Institute of Architecture of Application Systems

Formalizing the Cloud through Enterprise Topology Graphs

Tobias Binz, Christoph Fehling, Frank Leymann,
Alexander Nowak, and David Schumm

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{lastname}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings {INPROC-2012-18,
    author = {Tobias Binz and Christoph Fehling and Frank Leymann and Alexander
       Nowak and David Schumm},
    title = {{Formalizing the Cloud through Enterprise Topology Graphs}},
    booktitle = {Proceedings of 2012 IEEE International Conference on
       Cloud Computing},
    publisher = {IEEE Computer Society Conference Publishing Services},
    month = {June},
    year = {2012}
 }
```


Universität Stuttgart
Germany

Formalizing the Cloud through Enterprise Topology Graphs

Tobias Binz, Christoph Fehling, Frank Leymann, Alexander Nowak, David Schumm
Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

***Abstract*—Enterprises often have no integrated and comprehensive view of their enterprise topology describing their entire IT infrastructure, software, on-premise and off-premise services, processes, and their interrelations. Especially due to acquisitions, mergers, reorganizations, and outsourcing there is no clear 'big picture' of the enterprise topology. Through this lack, management of applications becomes harder and duplication of components and information systems increases. Furthermore, the lack of insight makes changes in the enterprise topology like consolidation, migration, or outsourcing more complex and error prone which leads to high operational cost. In this paper we propose *Enterprise Topology Graphs* (ETG) as formal model to describe an enterprise topology. Based on established graph theory ETG bring formalization and provability to the cloud. They enable the application of proven graph algorithms to solve enterprise topology research problems in general and cloud research problems in particular. For example, we present a search algorithm which locates segments in large and possibly distributed enterprise topologies using structural queries. To illustrate the power of the ETG approach we show how it can be applied for IT consolidation to reduce operational costs, increase flexibility by simplifying changes in the enterprise topology, and improve the environmental impact of the enterprise IT.**

Keywords-enterprise topology; enterprise topology graph; cloud; formalization; search; IT consolidation.

I. INTRODUCTION

IT costs as well as the complexity of IT landscapes are increasing rapidly [1] due to reorganizations, new technologies, mergers, acquisitions, and outsourcing. The constant change often causes a loss of insight into the enterprise topology which hinders its continuous improvement and makes an overall IT strategy and governance hard to realize. The lack of a consistent and machine-readable enterprise topology slows down adaptation to new requirements and market demands, integration, and IT consolidation. All of these are major challenges for IT departments and management alike, crucial challenges to stay competitive [2]. For example, research has shown that especially for mergers and acquisitions integration of IT is one of the key success factors [3][4]. IT consolidation, which has a high potential for cost savings, is today widely realized by introducing server virtualization in datacenters. The scope of consolidation can even be broadened to include all on-premise and off-premise infrastructure, middleware, and application components. However, due to the high complexity and number of dependencies new approaches are required, not only to IT consolidation, but also to make complex IT landscapes manageable and flexibly changeable.

In this paper we define the *Enterprise Topology Graph* (ETG) which enables different stakeholders to capture and manage all relevant IT components, spanning private, community, public, and hybrid clouds, within an organization. We propose the ETG as a graph-based model comprised of nodes and edges of arbitrarily customizable types, holding an extensible set of properties. IT components of an enterprise are employed as nodes that are linked together with their logical, functional, and physical relationships represented by edges in the graph. In this paper we show that exploiting graph theory through ETG has the potential to solve a wide range of problems in cloud research and enterprise computing. Approaches defined as formal graph or set algorithms can be applied efficiently to the ETG, which reflect a consistent representation of the realities in the enterprise topologies. To illustrate the ETG approach, we present a methodology for consolidation in enterprise topologies to support IT departments mastering IT consolidation in a generic, reusable, and efficient way.

We applied the following research design: We started with a literature survey which is summarized in Section V. The survey showed that existing topology models lack formality and focus on modeling applications. Based on summarization and abstraction of existing concepts we defined a first formalization of an Enterprise Topology Graph. We then used this formal model to define, implement, and evaluate an exemplary application: ETG-based IT consolidation. This practical application demanded to also define a search algorithm on ETG and it guided us to further refine the graph model. The resulting main contributions of this paper are therefore (i) a formalized, graph-based model for enterprise topologies which enables the practical application of graph theory to cloud and enterprise computing problems. (ii) An efficient search algorithm, which is able to locate structures in the ETG. (iii) Based on this foundation, we present a methodology for ETG-based IT consolidation, which addresses consolidation on infrastructure, platform, and software layer.

The remainder of this paper is structured as follows: We present Enterprise Topology Graphs along their formal definition and ETG example in Section II. In Section III, we show an efficient search algorithm on the ETG. As exemplary application of ETG we present the consolidation methodology in Section IV. The literature survey we conducted is reviewed in Section V. In the summary and

outlook in Section VI we describe future work to increase the applicability of ETG, e.g., specifying reusable operations.

II. The Enterprise Topology Graph

An *Enterprise Topology Graph* (ETG) is a graph-based model for enterprise topologies capturing all entities of enterprise IT and their logical, functional, and physical relationships. The conceptual model in Figure 1 depicts generic nodes and edges, which can be typed and refined to precisely define their semantics. Nodes and edges are both entities which can have an arbitrary number of properties. An ETG is a set of entities and their mapping onto types. Node types and edge types are structured in trees which are defined globally, i.e., they are not part of a definition of a particular ETG, but referenced in it and extensible. Segments are subgraphs of an ETG, representing a certain connected part of an ETG by holding only a subset of its entities.

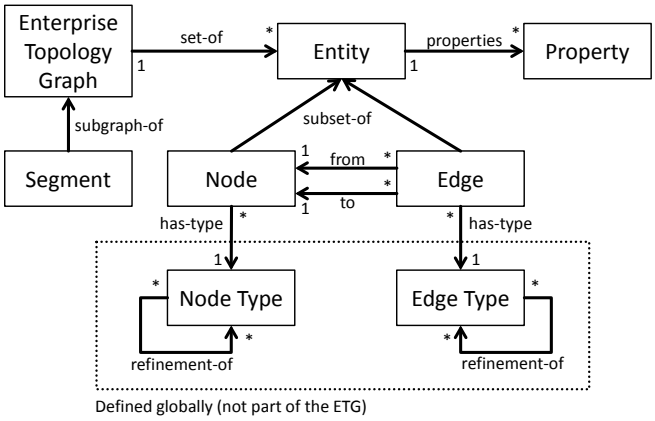

Figure 1. Conceptual Model of Enterprise Topology Graph

Representing enterprise topologies as formal graph enables the application of all knowledge in graph analysis and processing to improve the enterprise architecture. Without having done a detailed evaluation about this, we expect ETG to have hundreds of thousands of nodes. The ETG definition presented in our work is mainly influenced by TOSCA, the OASIS *Topology and Orchestration Specification for Cloud Applications* [5]. As of today, TOSCA seems to be the most complete, non-proprietary specification for describing applications and their management. ETG generalize TOSCA concepts to extend their purpose of describing application *models* towards the representation of enterprise topology *instances*. TOSCA represents a blueprint of a particular application, whereas on instance level ETG reflect many different (TOSCA) application instances present in the enterprise topology.

A. Node Types and Edge Types

Each entity is typed to bring domain-specific knowledge to the generic ETG entities. To establish a taxonomy between different levels of abstraction in types, we propose to structure them in trees, as shown in Figure 2. In [6] we successfully applied type trees to assigning functionalities applicable to certain types of entities to the type tree. When

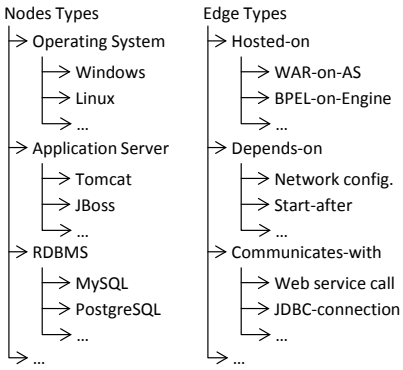

Figure 2. Exemplary extract of node type tree and edge type tree

applicable to a complete subtree the functionality is assigned to the respective node, e.g., start and stop to the *Application Server* node in Figure 2. Furthermore, type-specific aspects are assigned to leaves in the tree, e.g., port configuration to the *Tomcat* node in Figure 2. Based on our research we identified three fundamental edge types: *hosted-on*, *depends-on*, and *communicates-with*. For nodes, the types are much more diverse.

The usage of types enforces certain properties of their entities. This includes, for example, common functionality (like being an application server, hosting applications of a certain type, or a relational database, managing data in tables), attributes (like id, name), or offered operations with a predicable behavior to the outside (like start, stop, suspend). These properties are not explicitly modeled in the ETG, but tools using an ETG can rely on the fact that entities of a certain type follow the requirements of this type.

Additionally, this typing system helps to generalize and refine the graph. Generalization, for example, could replace detailed types with more general ones (e.g., replace type *Linux* with *Operating System*) or aggregate certain nodes and edges to reduce the ETG's complexity. For instance, an ETG can be made more abstract by aggregating all nodes from platform level and below. Refinement adds more details to the ETG by retrieving more information about an entity or assigning a more detailed type. In general, mechanisms to adjust the level of information abstraction depending on the application scenario are ongoing work.

Definition I (Node Types and Edge Types)

The sets $NodeTypes$ and $EdgeTypes$ contain the available types for nodes and edges respectively.

Definition II (Relation of Types)

Types are structured as trees, whereas each type may be present in multiple branches. This structure of types is denoted in the function $parentTypes$, whereas the inverse function is $childTypes$. The functions for $EdgeTypes$ are defined analogical.

$$parentTypes: NodeTypes \rightarrow 2^{NodeTypes}$$

$$childTypes: NodeTypes \rightarrow \{x \mid x \in NodeTypes\},$$
$$p \mapsto \{c \mid parentTypes(c) = p\}$$

B. Entities

The Enterprise Topology Graph is a directed, possibly cyclic graph that is constructed out of two basic entities, nodes and edges. The graph is directed to denote in which direction the semantic meaning introduced through edge types has to be interpreted. The ETG may be cyclic due to the ability to capture a wide variety of semantic information.

Definition III (Nodes)

The set N contains all the nodes of an ETG. A node represents everything that is part of an enterprise topology. Nodes represent the building blocks that applications need to operate sufficiently, for instance a specific workflow, a Web service, a user interface, a middleware component, or infrastructure element. Like in other modeling approaches, the granularity determining what is represented in the model is up to the modeler or extraction algorithm analyzing the IT landscape. We advocate that ETG should contain all information available and thus should be as fine-grained and detailed as possible. The more fine-grained an ETG is modeled the better results may be achieved during search, consolidation, and other tasks executed on ETG.

The function $nodeType$ associates one node with a specific type from the set of $NodeTypes$.

$$nodeType: N \rightarrow NodeTypes, n \mapsto t$$

Definition IV (Edges)

The set $E \subseteq N \times N$ contains directed relationships between two nodes. The type of an edge defines its semantics, for example, denoting that one node is 'hosted-on' another node. We do not constrain the use of edges between nodes, thus, some ETG may have cycles as described above. For usability reasons we define the function E which returns the set of incoming and outgoing edges of a Node. Additionally we define $incomingEdges$ and $outogingEdges$ analogous if only edges with this specific direction are of interest.

$$E: N \rightarrow N \times N, n \mapsto \{(f,t) | (f,t) \in E \wedge (f = n \vee t = n)\}$$

$$incomingEdges: N \rightarrow N \times N, \\ n \mapsto \{(f,t) | (f,t) \in E(n) \wedge t = n\}$$

The function $edgeType$ associates one node with a specific type from the set of $EdgeTypes$.

$$edgeType: E \rightarrow EdgeTypes, e \mapsto t$$

Definition V (Entities)

The set $Entities = N \cup E$ holds all the nodes and edges of an ETG. This set is used by the following definitions.

C. Properties

Properties capture the domain-specific knowledge of ETG entities as key-value-pairs. They are used to represent properties of the entity, information augmented by tools or algorithms, implementation artifacts, or non-functional requirements. For instance, an ETG can be augmented with runtime information indicating workload gathered from monitoring. The different types of properties are uniquely identified and distinguished by URIs, which specify the structure and data type of the value.

Definition VI (Property Keys)

As property keys we use URIs [7] which enable both, hierarchical structuring and extensibility. Properties can be grouped using the structure of the URI.

$$PropertyKeys = \{uri \mid uri \in RFC3986\}$$

Definition VII (Property Values)

Valid attribute values are represented by the set $PropertyValue$ which explicitly includes structured strings, for example, XML documents or URIs, and binary data like a Java Web Archive (WAR). Multiple values can be condensed into one value by using a type-specific list format.

Definition VIII (Properties)

The generic properties function returns the attributes of an entity. The different kind of data stored in properties is not distinguished by using different functions. Property keys define the semantic of the value and what data type can be expect from the value.

$$properties: Entities \rightarrow EntityProperties \\ \subseteq PropertyKeys \times PropertyValues$$

D. Final Definitions

Based on the preceding definitions the Enterprise Topology Graph and its Segments are defined as follows:

Definition IX (Enterprise Topology Graph)

The Enterprise Topology Graph consists of the set of nodes, edges, and their type mappings. The ETG only includes the assignment function of node types and edge types ($nodeType$ and $edgeType$), but not the set of types ($NodeTypes$ and $EdgeTypes$) because they represent a global set which is not bound to a specific ETG.

$$ETG = \{N, E, nodeType, edgeType\}$$

This definition not only allows the application of algorithms based on graph theory, but also algorithms with low complexity operating on the sets of nodes and edges.

Definition X (Segment)

A segment is a part of an ETG, which is called subgraph in graph theory. The number of entry and exit nodes is not limited, i.e., it may be greater than one. A segment's nodes must be connected.

$$Segment = \{SN \subseteq N, SE \subseteq E, nodeType, edgeType\}$$

Segments are the concept to refer to parts of an ETG and the corresponding edges. Segments can be used to reduce complexity of algorithms, whose runtime typically decreases with the number of processed entities. Also for human readers, interested only in a particular aspect included in the ETG, work is made easier through segmentation.

E. ETG Example

Figure 3 illustrates the ETG approach by depicting one segment of a larger ETG. The ETG segment contains two virtual machines hosting nodes of the type Tomcat

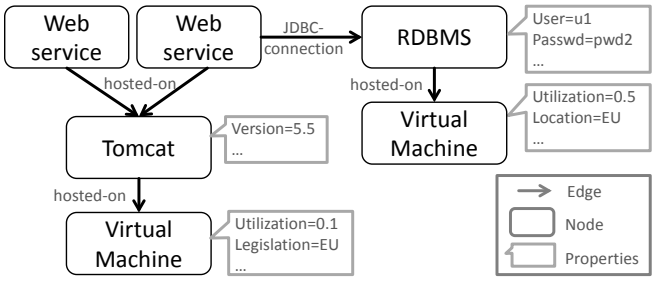

Figure 3. Exemplary ETG

application server and relational database management systems (RDBMS). On the application server two Web services are hosted with one of them connecting to the RDBMS. Some nodes have properties, for example, the virtual machines are located in the European Union (EU) and are augmented with utilization. A graphical notation for ETG is not explicitly defined yet. Therefore we use an intuitive notation explained in the legend of Figure 3.

III. Search in Enterprise Topology Graphs

An essential algorithm required frequently for the management of large ETG is *search*. The approach for search in ETG we present in the following does not only locate certain nodes or properties by comparing strings, but it allows using ETG segments as queries. Consequently, the algorithm enables finding complex structures in the ETG. For example, we can locate all nodes hosted on outdated Tomcat application servers. As shown in Figure 4 the query contains a node of the type *Tomcat* with the property *version=5.5*. This query could be implemented in code, but a generic search algorithm specifying this query as segment, i.e., in the same model the ETG is already defined, provides a comprehensible, reusable, and generally applicable solution. To increase flexibility, we offer wildcards for the types and properties of entities, indicating to the search algorithm that any value is valid. Figure 4 depicts the aforementioned query visually, including one wildcard as type of the node hosted-on the Tomcat node. When applying this query to the ETG example in Figure 3, the two results would be the two Web services in the upper left of the figure.

A. Search Algorithm

The input of the algorithm is an ETG (cf. *definition IX*) to search on and a query in form of an ETG segment (cf. *definition X*). The output is a set of zero or more segments representing the nodes and edges in the graph that match the search query.

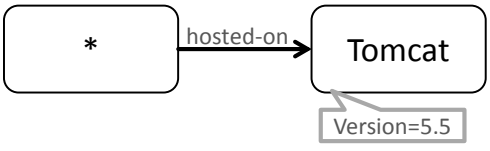

Figure 4. Search query defined as ETG segment locating arbitrary typed nodes which are hosted on Tomcat servers with version 5.5

The described ETG problem can be reduced to the problem of *graph isomorphism* evaluating the equality of two graphs. *Subgraph isomorphism* decides if a graph S (the search query) is isomorph to a subgraph of G (the ETG). This is the case if any node in S can be mapped to one node in G and each edge existing between two nodes in S also exists between these nodes in G [8]. Graph isomorphism is well-researched and we list corresponding algorithms in Section V. We apply subgraph isomorphism to searching on ETG through identifying all subgraphs, i.e., search results, of G matching S. Our implementation is built upon the *VF2 algorithm* presented by Cordella et al. in [9] with some extensions discussed in the following. VF2 solves the problem of (sub) graph isomorphism for directed graphs and was proven to be efficient for large graphs [9] which qualifies it for usage in ETG. Core of the algorithm is a recursive function deciding if an isomorphism was found or if the current state can be extended towards an isomorphism. Special care is given to the recursive function regarding spatial complexity. The search tree is pruned by a set of syntactical feasibility rules which are the main contribution of VF2 compared to its predecessor.

In the following we describe how we implemented and optimized VF2 to suit ETG requirements:

(i) In ETG edges are typed and may have properties which must be inspected during search. VF2 does not regard edges and evaluates them always together with the two connected nodes. To overcome this we represent ETG edges as nodes and introduce *plain* directed edges between nodes representing ETG nodes and nodes representing ETG edges. With this, we gain the ability to formulate complex searches towards edges without changing the core graph algorithm.

(ii) VF2 checks the validity of states with a set of 5 syntactical feasibility rules. To verify the semantics of the respective domain the function F_{sem} is defined. For our use case we implement F_{sem} to check if the type and properties on entities match the search query. For a specific type a node being a semantic match must be of this type *or* any subtype of the given type, i.e., $\bigcup_{i=1}^{\infty} childType^{i}\big(nodeType(s)\big)$. If no restriction on the type of the entity should be applied a wildcard type matching all other types is used. If properties are provided in the search query their values must *exactly* match to be valid. Additionally, a wildcard can be used as value which is used to enforce the existence of a property. Otherwise, if a property is not defined on a node of S, it may or may not exist on the respective node of G. Formally F_{sem} is defined as follows:

$$F_{sem}: N_S \times N_G \to \{true, false\}, (s,g) \mapsto$$
$$nodeType(g) \in \{WILDCARD \cup nodeType(s) \cup$$
$$\bigcup_{i=1}^{\infty} childType^{i}\big(nodeType(s)\big)\} \wedge \forall (k,v) \in$$
$$properties(s): v = WILDCARD \vee (k,v) \in properties(g)$$

In future work we would like to extend F_{sem} to support more complex conditions on property values than wildcard and equality, for example, to state that a value starts with a certain string or matches a certain regular expression.

(iii) Instead of returning the first result found and stopping the algorithm we implemented our search algorithm to store the result to a list and continue search for further hits.

(iv) VF2 generates a large (i.e., $|N_S| * |N_G|$) set of node pairs (the *candidate pair set*) to which the feasibility rules are applied. Many unnecessary computations are done with mappings which cannot be extended to a result. This can be improved by choosing a start node out of S. The key property of the search query is that each node contained in the graph S must exist in each of the results. Based on this we select one node in S as start node and apply the semantic feasibility rules only to the node pairs created by combining the start node with each node in the ETG. This results in a much smaller number of initial mappings to process ($|N_G|$).

We argue that the probability that extending the mapping fails is higher for nodes in S with many restrictions. This is the case when a type is defined instead of using a wildcard type, restrictions on properties are imposed, and the node has lots of edges. To let unnecessary search branches fail fast, instead of processing them, we propose using a heuristic to choose a start node with many restrictions. We define the function which evaluates the nodes in the search segment based on the number of restrictions they impose. The node with the highest rating is selected and used as start node.

$$rating: Nodes \rightarrow int, n \mapsto 5 * (nodeType(n) \neq WILDCARD) + |poperties(n)| + 2 * (|incomingEdges(n)| + |outgogingEdges(n)|)$$

The last optimization (iv) highlights the advantage that algorithms on ETG can switch between using the sets of entities and the graph structure of the entities.

The complexity of our search algorithm based on VF2 is $\mathcal{O}(e! * e), e = |Entities_G|$ in the worst case [9] and $\mathcal{O}(e)$ in the best case. We improved the best case which is finding a single node in the ETG by introducing the selection of the start node (iv). In this case the start node or nodes in G are already the search results and only one pass over all ETG nodes is required.

B. Evaluation Results

We evaluated our search algorithm with a set of ETG samples to analyze its performance characteristics for different types of search queries. Both, the formal ETG definition and search algorithm, have been implemented using Java. Table 1 shows the runtime in seconds we achieved on a standard notebook.

TABLE I. RUNTIME IN SECONDS OF ETG SEARCH QUERIES

Search Query	**ETG samples**			
#nodes	***100***	***1,000***	***10,000***	***100,000***
# edges	***250***	***2,500***	***25,000***	***250,000***
(i) Node of arbitrary type with a certain property	0.003	0.003	0.007	0.019
(ii) Arbitrary typed node hosted on typed node (Figure 4)	0.005	0.005	0.015	0.020
(iii) Complex structure of typed nodes and edges (11 entities)	1.625	1.692	1.710	1.760
(iv) Same as query (iii) but nodes are not typed	0.130	0.499	4.257	40.423

These measurements show that even for complex structured queries the runtime of the search algorithm only increases slowly with the size of the sample ETG, as long as there are not that many wildcards contained in the query. Query (iv), with wildcards used for all node types, shows a different characteristic. Due to the low number of restrictions it is closer to subgraph isomorphism than the other queries.

IV. ETG APPLICATION EXAMPLE: CONSOLIDATION OF IT INFRASTRUCTURES

Consolidation in general is the process of aggregating or merging multiple similar entities into one. Consolidation in the context of cloud computing is currently mainly focused on virtualization, i.e., the consolidation of multiple physical servers as virtual machines on one physical server [10]. However, performing consolidation on the enterprise topology enables to broaden the scope of consolidation to consider infrastructure, platform, and software layer.

The methodology we propose *identifies* consolidation possibilities and provides pluggable extension points to automate the actual consolidation through type-specific implementations. A generic automated consolidation is not possible due to type-specific configurations or requirements. For example, one cannot generally assume that the migration of an application running on application server A onto application server B, which hosts similar applications, is generally possible. The information needed to decide this may be contained in the ETG, but the evaluation must be done by a domain expert or a domain-specific algorithm.

All domain-specific consolidation knowledge is captured in a so-called *consolidation strategy*, in reference to the strategy pattern in programming [11]. As depicted in Figure 5, a consolidation strategy consists of an extraction query, a target query, and a strategy implementation, which may be performed by a human expert or automation code. The *extraction query* identifies candidate segments in the ETG that a consolidation strategy is able to consolidate. Beyond the identification of candidate segments in this example, further properties like low utilization, outdated or not supported software or hardware, insufficient quality of service are applicable. The *target query* locates candidate segments in the ETG where the consolidation candidates might be migrated to. Based on type-specific knowledge and evaluation the strategy implementation migrates the

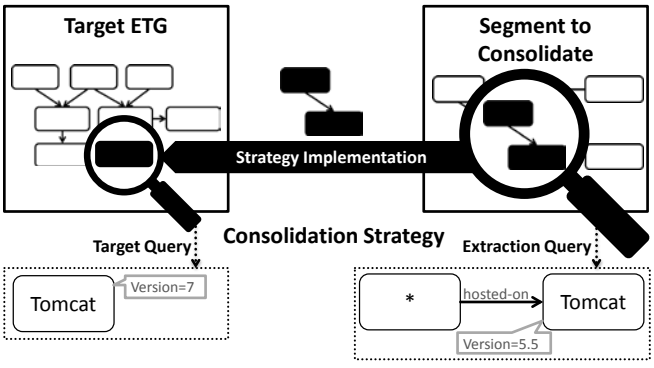

Figure 5. Exemplary consolidation strategy consisting of two queries and an implementation.

consolidation candidates to the identified target and adapts the environment accordingly, for example, by reconfiguring or removing nodes not used anymore.

A. Consolidation Methodology

Our consolidation methodology supports two scenarios: First, consolidate a segment, *segment to consolidate* in Figure 5, into the target ETG, as shown in Figure 5. This may be applied during IT consolidation in reorganizations, mergers, and acquisitions to integrate IT without creating a high degree of duplication. Second, both queries can be applied to one ETG to optimize the enterprise topology. The exemplary consolidation strategy to illustrate our methodology aims to consolidate outdated Tomcat application servers from an acquired partner company whose IT should be integrated into the existing enterprise IT.

The following methodology shows which steps a consolidation strategy follows:

(i) Similarity – The extraction query, bottom right of Figure 5, is used to find possible consolidation candidates in the segment which should be consolidated. In our exemplary consolidation strategy we would like to locate Tomcat nodes of version 5.5. The target query, bottom left of Figure 5, identifies possible consolidation targets in the ETG. In the example we aim at improving the enterprise topology and therefore restrict the Tomcats located by the target query to version 7. Both queries are executed on the respective ETG using the search algorithm presented in Section III.

(ii) Compatibility – The strategy implementation uses domain-specific knowledge regarding functional and non-functional aspects to determine which of the resulting segments are compatible with respect to consolidation. In the example the configuration and non-functional requirements, e.g., the legislative area the infrastructure is located in, of the Tomcat nodes must be compared to decide if an application can be moved. In addition, the utilization of the infrastructure, e.g., memory or CPU, and software, e.g., number of connections allowed per license, should be considered to evaluate if the nodes in the target segment are capable to host the nodes to be consolidated. For this, future utilization must be estimated to ensure that none of the nodes is over-utilized after the consolidation.

(iii) Environment – Components cannot be handled independent of their environment and relations to other components. In the ETG example in Figure 3, the environment of the right *Web Service* would be the *RDBMS*. In general, the environment includes aspects like networking or physical locations, which determines accessibility, security, and legislative implications of the component. Additionally, there are a number of relations to other components which must be analyzed, for example, databases or Web services. These relations must be reestablished after the consolidation, which might for instance require a network reconfiguration, or that corresponding nodes are also migrated by recursively invoking another consolidation strategy. In any case, the references to the nodes must be changed, i.e., the connection string to the database or Web service must be adapted. This is processed by the consolidation strategy, because the place this information is stored depends on the node type.

(iv) Execution – After the consolidation is planned and evaluated it must be translated into technical actions to perform the actual consolidation. In our example the respective applications and their data are migrated to the target environment, dependencies to and from the environment are updated, the application is started in the target segment, and shut down and deprovisioned in the old one. The details of this process depend on the used cloud management system. The generation of automated workflows out of the high level decision how to consolidate certain components is an interesting field of future research.

(v) Progress – The consolidated parts are now marked and the process is repeated using the available consolidation strategies until everything has been consolidated or no further consolidation is possible anymore.

B. Discussion

The methodology proposed in this section describes the consolidation of components from a conceptual viewpoint in order to illustrate how ETG can be used as a basis to tackle challenges in cloud and enterprise computing.

Due to availability requirements some nodes must not be hosted on the same physical infrastructure or must even be located in different geographic regions. These and similar optimization criteria are captured through edges with a dedicated edge type. To document this non-functional requirement the ETG is augmented with additional edges, to make this information available to consolidation strategies.

One reason for the increased efficiency in the cloud is multi-tenancy, which is realized through sharing resources between multiple tenants. When sharing a physical machine by running multiple virtual machines on it, the operating system and middleware are duplicated for each tenant. If the tenants are sharing the middleware, the overhead is reduced. On the other side the isolation decreases accordingly, which must then be enforced using appropriate mechanisms. Consolidation strategies introduce resource sharing on the highest possible layer, according to the cloud layers software, platform, and infrastructure [12], to maximize the savings of the consolidation. Another way to improve consolidation is the use of so-called *adapter* which allows hosting nodes on other nodes to which they have not been compatible before, as described in [6].

V. Related Work

The results of our literature survey are structured into two sections: Section V.A observes how topologies and enterprise architectures are currently modeled. Section V.B presents the related work regarding search algorithms and discusses current research in IT consolidation.

A. Models for Enterprise Topology

Enterprise Architecture Management (EAM) defines the layers business, process, integration, software, and infrastructure [13]. Two of the most important aspects are (i) a holistic view with respect to all enterprise architecture layers and (ii) the alignment of business and IT [14]. ETG

support this by representing nodes in the integration, software, and infrastructure layer, and their dependencies. Despite capturing technical details the presented search algorithm and consolidation methodology show how ETG can be used to achieve business goals. Fran et al. [15] describe a domain-specific language based on the Meta Modeling Language [16]: The IT domain-specific Modeling Language (ITML) contains a fixed set of entities like hardware, software, services, and processes, which can be refined if needed, and a fixed set of relations. ETG are more generic and enable modeling on multiple levels of granularity while still providing strong typing through structured node types and edge types, as presented in Section II. Schweda [17] presents how to create tailored languages capturing enterprise-specific aspects of enterprise architectures by defining best practice building blocks. These building blocks are on a high level of abstraction and may be technically realized by a number of ETG entities in the enterprise topology.

There are data models expressing dependencies of IT infrastructure elements [20], these are unsuitable and incomplete to express the whole enterprise topology. The generality of the ETG, however, enables the mapping of these data to ETG entities. Therefore, existing management tools supporting CIM [20], SNMP [21], and so on may be used to extract an ETG from a company's infrastructure.

There are different approaches describing a (composite) application: An application model which includes dependency and deployment relations between components is formalized in [22]. Mietzner [23] describes *cafe* which supports modeling and deployment of composite applications, formalized by a formal definition of an application model. Leymann et al. [24] broadens this typing model and adds labels (properties in the ETG definition) to optimize the distribution of applications between clouds. However, types are not structured, i.e., in contrast to our work there is no relation between different types. The *Topology and Orchestration Specification for Cloud Applications* [5] (TOSCA) is a recently initiated standardization initiative at OASIS to define the topology and management aspects of applications. The presented ETG definition is based on concepts of TOSCA, as detailed in Section II, and able to include instances of application models described in TOSCA. In essence, all these approaches describe application models. ETG however represent the enterprise topology including a number of instance of these models. [25] presents an UML-based approach to model application structures used in the scope of topology discovery. The *Service Component Architecture* (SCA) [26] composes applications out of services by defining functional relations. Other relations, e.g., where a service is deployed, are not captured. In software architecture, design, and development languages like the Acme architectural description language [18] and UML [19] are used. However, they target mostly application architectures and do not offer the formality we are looking for. To be able to build the code fragments of the described applications build tools like Apache Maven capture dependencies between code artifacts, but are not able to depict other relations, functional or logical. The presented approaches either do not have a broad enough typing system, which is crucial to bring different semantics found in enterprise topologies to generic entities, or they do not provide a formal model appropriate for enterprise topologies. However, information included in the application models may be used to augment the ETG.

B. Search and IT Consolidation

To solve the problem of (sub)graph isomorphism different algorithms exist: Messmer [27] compares different graph matching algorithms. The algorithm of Ullmann [28] was already published in 1976 and improved, for example, in [8]. McKay [29] solves the graph isomorphism problem by transforming the graphs into a canonical data format. In some cases though, its runtime is exponential. For search, i.e., subgraph isomorphism, we used the VF2 algorithm in [9], which argues to provide better performance in terms of time and spatial complexity than Ullmann. The concise listing of all these works shows how well-elaborated the concepts in graph theory are. We take significant advantage from that as these results are of enormous help for defining ETG-based applications.

In EAM, consolidation efforts are mostly in the focus of the business perspective. Buckl et al. [30], for instance, describes patterns to merge and harmonize business functionalities after mergers. Business IT alignment after mergers and acquisitions is also addressed in [31]. This approach states facts about the systems to be consolidated and rules as relations between facts. Based on this, predictions regarding the impact of business decisions to IT can be made. On enterprise topology level, Speitkamp and Bichler [32] describe an algorithm for server consolidation, as well as heuristics which can be used if the number of server is too high for the exact algorithm. [33] describes server consolidation through virtualization. Consolidation on platform or application level [12] is not investigated in detail so far. [34] is a case study of a software consolidation in the banking sector, merging three systems after multiple mergers into one. Therefore, our approach broadens the consolidation scope and additionally exploits the cloud characteristic of decreasing costs if resources are shared on higher layers, i.e., if an application server is shared the efficiency is higher than sharing only the physical server.

VI. Summary and Future Work

In this paper we presented Enterprise Topology Graphs that allow us to comprehensively capture enterprise IT landscapes. Based on the formal definition of ETG a generic search algorithm has been introduced that enables identifying structures in ETG. The presented consolidation methodology identifies possible consolidation candidates and targets using this search algorithm. Consolidation strategies are defined as part of the methodology to capture domain-specific knowledge required to evaluate consolidation possibilities and to identify concrete actions for realizing them.

Formalizing enterprise topologies enables the application of proven graph algorithms to problems in cloud and EAM research, for example, VF2 [9] was used for the search

algorithm in Section III. Furthermore, the consolidation methodology shows how a generic operation on the ETG (search) can be easily combined into higher level functionality (consolidation). Our vision is to have a broad set of basic operations for the ETG as reusable building blocks to simplify future research.

The areas to apply ETG are manifold: Future research will address further cloud challenges like analyzing the relation of business processes to the services represented in enterprise topologies, outsourcing, migration, ensuring compliance, modeling of new applications, and approaches to gather relevant information to augment the ETG. Due to the large number of entities in an ETG mechanisms have to be researched to adjust the granularity and level of abstraction of the graph. Depending on the problem domain the ETG needs to be structured accordingly to only include the required information.

ACKNOWLEDGMENT

This work was partially funded by the BMWi project CloudCycle (01MD11023) and the BMWi project Migrate! (01ME11055). D. Schumm would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

REFERENCES

[1] J. Garbani, T. Mendel, and E. Radcliffe, “The Writing on IT’s Complexity Wall,” Forrester Research, June 2010.

[2] A. Reichman, “Measuring The Cost Of IT Consolidation - Identifying the Elements that Contribute to Economic Analysis,” Forrester Research, Nov. 2007.

[3] D. Aponovich, “IT Integration seen as Key to Merger Success,” CIO Update, Mar. 2002.

[4] M. Mehta and R. Hirschheim, “A Framework for Assessing IT Integration Decision-making in Mergers and Acquisitions,” Proceedings of the 37th Hawaii International Conference on System Sciences, pp. 5–8, 2004.

[5] “Topology and Orchestration Specification for Cloud Applications (TOSCA),” OASIS, Oct. 2011.

[6] T. Binz, F. Leymann, and D. Schumm, “CMotion: A Framework for Migration of Applications into and between Clouds,” Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), Dec. 2011.

[7] T. Berners-Lee, R. Fielding, and L. Masinter, “RFC 3986, Uniform Resource Identifier (URI): Generic Syntax,” 2005.

[8] G. Valiente, “Algorithms on Trees and Graphs,” Springer, 2002.

[9] L. Cordella, P. Foggia, C. Sansone, and M. Vento, “A (Sub)Graph Isomorphism Algorithm for matching large Graphs,” Pattern Analysis and Machine Intelligence, IEEE Transactions, vol.26, no.10, pp. 1367–1372, Oct. 2004.

[10] M. Mishra and A. Sahoo, “On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach,” 2011 IEEE International Conference on Cloud Computing (CLOUD), pp. 275-282, July 2011.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software,” AddisonWesley Professional, Nov. 1994.

[12] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” Information Technology Laboratory, Jul. 2009.

[13] R. Winter and R. Fischer, “Essential Layers, Artifacts, and Dependencies of Enterprise Architecture,” Journal of Enterprise Architecture, pp. 7–18, 2007.

[14] S. Buckl, A. Ernst, J. Lankes, F. Matthes, and C. Schweda, “State of the Art in Enterprise Architecture Management,” Technische Universität München, Chair for Informatics 19 (sebis), 2009.

[15] U. Frank, D. Heise, H. Kattenstroth, D. Fergusona, E. Hadarb, and M. Waschkec, “ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management,” Proceedings of the 9th OOPSLA workshop on domainspecific modeling, 2009.

[16] U. Frank, “The MEMO Meta Modelling Language (MML) and Language Architecture,” ICB Research Report, University of Duisburg-Essen, 2008.

[17] C. Schweda, “Development of Organization-Specific Enterprise Architecture Modeling Languages Using Building Blocks,” Dissertation, Technische Universität München, 2011.

[18] D. Garlan, R. Monroe, and D. Wile, “Acme: An Architecture Description Interchange Language,” CASCON First Decade High Impact Papers, ACM, pp. 159–173, 2010.

[19] “OMG Unified Modeling Language (UML),” Specification, Object Management Group, 2007.

[20] “Common Information Model (CIM) Infrastructure,” Specification, Distributed Management Task Force, 2010.

[21] J. Case, M. Fedor, M. Schoffstall, and J. Davin, “RFC 1157: Simple Network Management Protocol (SNMP),” Internet Engineering Task Force, May 1990.

[22] T. Unger, R. Mietzner, and F. Leymann, “Customer-defined Service Level Agreements for Composite Applications,” Enterprise Information Systems, 2008 International IEEE Enterprise Computing Conference (EDOC), 2009.

[23] R. Mietzner, “A Method and Implementation to define and provision variable Composite Applications, and its usage in Cloud Computing,” Dissertation, University of Stuttgart, Aug. 2010.

[24] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, and S. Dustdar, “Moving Applications to the Cloud: An Approach based on Application Model Enrichment,” Intl Journal of Cooperative Information Systems, World Scientific, 2011.

[25] V. Machiraju, M. Dekhil, K. Wurster, J. Holland, M. Griss, and P. Garg, “Towards Generic Application Auto-discovery,” HP Laboratories Palo Alto, Jul. 1999.

[26] “Service Component Architecture (SCA),” Specification, OASIS, Mar. 2007.

[27] B.T. Messmer, “Efficient Graph Matching Algorithms for Preprocessed Model Graphs,” PhD Thesis, Instute of Computer Science and Applied Mathematics, University of Bern, 1996.

[28] J.R. Ullmann, “An Algorithm for Subgraph Isomorphism,” J. Assoc. for Computing Machinery, vol. 23, pp. 31–42, 1976.

[29] B. D. McKay, “Practical Graph Isomorphism,” Congressus Numerantium, vol. 30, pp. 45–87, 1981.

[30] S. Buckl, A. Ernst, H. Kopper, R. Marliani, F. Matthes, P. Petschownik, and C. Schweda, “EAM Pattern for Consolidations after Mergers,” Workshop on Patterns in Enterprise Architecture Management (PEAM 2009), Kaiserslautern, pp. 67–78, 2009.

[31] B. Srivastava and P. Mazzoleni, “Business Driven Consolidation of SOA Implementations”, 2010 IEEE International Conference on Services Computing (SCC), pp. 49-56, 2010.

[32] C. Speitkamp and M. Bichler, “A Mathematical Programming Approach for Server Consolidation Problems in Virtualized Data Centers," Services Computing, IEEE Transactions, vol.3, no.4, pp. 266–278, Dec. 2010.

[33] B. Braswell, M. Newman, and C. Wiberg, “Server Consolidation with VMware ESX Server,” IBM Redpaper, January 2005.

[34] P. Meinen and J. Overhoff, “Banking IT Consolidation in Time and on Budget,” European Conference on Software Maintenance and Reengineering, pp. 319–320, Mar. 2009.