Modeling Deployment of Enterprise Applications

Susanne Patig[1]

[1] University of Bern, IWI, Engehaldenstrasse 8, CH-3012 Bern
susanne.patig@iwi.unibe.ch

Abstract: Deployment comprises installing, activating and updating applications. The applications to be deployed usually require certain conditions that can refer to hardware capabilities, other software (dependencies), physical artifacts or configuration. Deployment planning aims at satisfying these prerequisites without violating the hardware's capabilities. This paper presents the domain-specific language *ADeL (Application Deployment Language)* that was designed to describe and validate deployment plans. The ADeL metamodel was implemented within the Eclipse Modeling Framework (EMF) and contains a set of OCL constraints (implemented with the tool *Topcased*) to enable the automatic validation of deployment plans.

1 Introduction

As a result of mergers, acquisitions and evolving business needs, the applications and the *IT infrastructure* (hardware, system software and network) of a company change. Typical *Enterprise architecture (EA)* approaches do not trace in detail the applications to the used IT infrastructure [2]. Such tracing information, however, is needed for IT consolidation, dependency analysis and the management of application portfolios [2].

This paper tries to close the gap between applications and IT infrastructure by dealing with deployment planning in data centers. *Deployment* comprises all activities that make some released software ready for use, namely installation, activation and updating [5]. During deployment planning, applications must be assigned to a given IT infrastructure in such a way that the assignment is valid. The approach presented here is capable of modeling such assignments and checking their validity.

Section 2 summarized the requirements of deployment planning in data centers; Section 3 sketches the relevant existing approaches. In Section 4, a new domain-specific language called ADeL (*A*pplication *De*ployment *L*anguage) is proposed and applied to a real-world case. The last section contains critical reflections and an outlook.

2 Requirements of Application Deployment

The requirements of planning the deployment of complex applications in data centers are derived from two real-world cases, namely the installation of SAP SCM in the SAP UCC in Magdeburg and the installation of the content management system openCMS

(http://www.opencms.org/) in the VLBA Lab Magdeburg[1]. During requirements elicitation, particular system's instances as well as documents related to installation were analyzed, and people involved in the installation process were interviewed. The complete description of the cases can be found in [16]; for brevity, only the elicited requirements are listed in the following. In detail, an approach that supports the deployment of complex applications in data centers must be capable to express:

[Rq1] The available *hardware* and its technical characteristics (*capabilities*). The most important technical characteristics are CPU type (restricting the operating system) and CPU count as well as the sizes of RAM and HD.

[Rq2] All that is to be deployed and has certain prerequisites, i.e., application components, system software or installation media. The prerequisites can refer to *hardware* capabilities, other *software* (i.e., dependencies), physical *artifacts* (e.g., executables, configuration files) or *configuration* activities (defining ports, IP addresses etc.). The objects to be deployed are called *requirement units*.

[Rq3] The direct or indirect *assignment* of requirement units to hardware; indirect assignments involve intermediate requirement units.

[Rq4] Deployment *constraints*, e.g., whether or not some software units are allowed to run on the same server.

[Rq5] *Choices* in realizing some functionality (e.g., 'database functionality') by distinct software products (e.g., Oracle, DB2, MaxDB).

Interviews with staff involved in the installation of complex applications made it clear that the expressive power reflected by the requirements [Rq1] to [Rq5] should be realized by a *modeling language* [Rq6] that is *SI*mple, *E*xtensible and *G*eneral; I call this the *SIEG principle*. *Simplicity* [Rq7] means that only a small set of well separated concepts should be used because the cognitive capacity of humans is limited [3]. *Extensibility* [Rq8] enables the adaptation of the new approach to specific deployment situations (by adding metamodel elements) and unanticipated usage scenarios (model-driven development by adding (meta-) model transformations). As real-world application landscapes and hardware are heterogeneous, the new approach should be *general* [Rq9], i.e., independent of particular hardware, software and software architecture. Finally, checking modeled deployment plans for their validity [Rq10] prior to installation was rated as an important benefit of modeling.

The next section analyses whether or not the existing approaches in the field of deployment satisfy the elicited requirements.

3 Existing Approaches in the Field of Deployment

Software deployment has been largely neglected in academic discussion. Fig. 1 arranges the existing *approaches* (i.e., tools, modeling languages and standards) in a portfolio: The axes reflect the focus of the approaches and the kind of support they offer, respectively. The sizes of the bubbles illustrate the covering realized by each approach.

[1] All names of products are trademarks, service marks or registered trademarks of the respective companies.

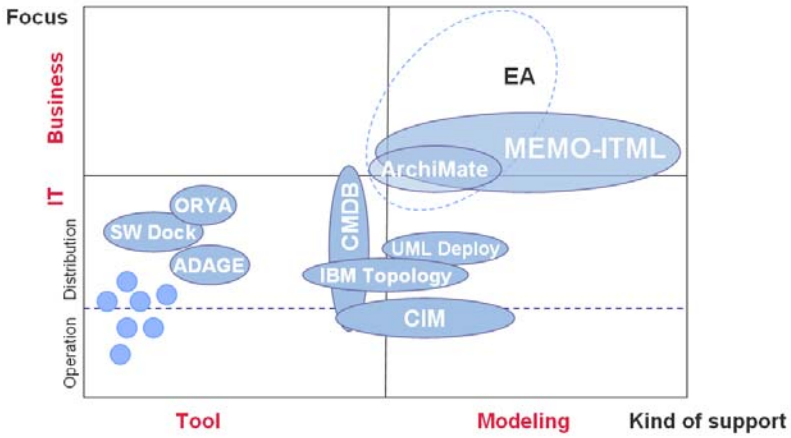

Fig. 1. Existing approaches in the field of deployment.

The lower left quadrant of the portfolio contains tools that automate all deployment activities (ORYA [12], Software dock [8]) or only installation (ADAGE [10]); the unlabeled bubbles represent proprietary tools. In this paper, these approaches are neglected as they do not satisfy the modeling requirement [Rq6]. Moreover, ADAGE and the proprietary tools are not general [Rq9].

The approaches that model deployment focus on business or IT. The business focus is typical for approaches stemming from the field of EA (ArchiMate [11]) or go even beyond (MEMO ITML [28]). Both approaches provide constructs to express applications and system software [Rq2] as well as hardware [Rq1] and the corresponding assignments [Rq3]. However, as enterprise architecture aims at aligning business and IT, the resulting models are hardly extensible for purposes beyond description [Rq8], constraints [Rq4] and choices [Rq5] cannot be represented, and the validity of the modeled deployment scenarios [Rq10] cannot be checked.

UML deployment diagrams [14], IBM topologies [13], [16] and the Common Information model (CIM) [6], which is implemented in configuration management databases (CMDB), represent the IT view on deployment. These approaches satisfy the requirements [Rq1] to [Rq3] related to expressive power (minor restraints refer to the representation of artifacts) as well as the requirement of extensibility [Rq8]. However, gaps exists for the other requirements: The UML relies on the OCL [11] to specify any kind of *deployment constraints* [Rq4], whereas IBM topologies support a limited set of constraint types by particular constructs [13]. Deployment choices [Rq5] are not covered by the existing approaches, except for an indirect modeling with IBM topologies (see [16]). Measured by the number of constructs, none of the approaches is simple [Rq7]. Because of being standards, UML deployment diagrams and CIM are general [Rq9], IBM topologies and CMDBs are not. Only IBM topologies include a (restricted) way to check the validity of deployment plans prior to installation [Rq10].

To sum it up, the main deficiencies of the existing approaches are missing simplicity [Rq7] as well as lack of support for deployment choices [Rq5], constraints [Rq4] and checking the validity of deployment models [Rq10]. The domain-specific language ADeL proposed in the next section was designed to overcome these deficiencies.

4 ADeL – The Application Deployment Language

4.1 ADeL Metamodel

The ADeL metamodel consists of the abstract syntax depicted in Fig. 2 as well as a set of OCL invariants.

The core ADeL metamodel elements are `units`; each `unit` can be linked (`isLinked`) to an arbitrary number of other `units`. As an abstract super class, a `unit` defines the common properties of both `RUnits` (*requirement units*, see [Rq2] in Section 2) and `hardware`: the `name`, an identifier `id` (if units cannot be recognized from their names), the type of CPU (`CPU_type`), the total number of CPU cores (`CPU_count`), the sizes of hard disk (`HD`) and random access memory (`RAM`). All properties except for the name are optional.

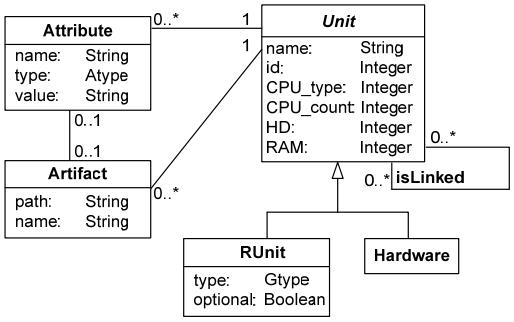

Fig. 2. Abstract syntax of the ADeL metamodel.

Units of the subtype `hardware` represent physical capabilities [R1] to host some `RUnit`(s). Basically, the prerequisites for `RUnits` can refer to hardware, software, physical artifacts or configuration (see [Rq2] Section 2). Hardware prerequisites are expressed by the properties listed above and paths to hardware [Rq3], whereas software prerequisites (dependencies) correspond to links (`isLinked`) between `RUnits`.

The predefined properties of `units` express standard deployment needs. Unforeseen prerequisites or capabilities can be modeled by `attributes` [Rq8]. A `unit` may be associated with an arbitrary number of `attributes`.

`RUnits` have the additional properties `type` and `optional`. The property `type` indicates whether a `RUnit` is elementary (`GType = E`), which is the default, or groups other `RUnits`. Groups of `RUnits` are either conjunctive (`GType = A`), disjunctive (`GType = O`) or exclusive (`GType = X`), i.e., all/at least one/one and only one of the grouped `RUnits` is to be deployed. Often such groups are conceptual, i.e., they structure ADeL models or prepare deployment choices [Rq5]. The property `optional` describes whether or not some `RUnit` must be deployed at all.

Physical artifacts are needed for deployment execution, IT operations or result from configuration activities (e.g., configuration files, start profiles). They can be represented by the metamodel element `artifact`. A `unit` can be linked to any number of `artifacts`. The location of an `artifact` must always be given (property `path`), whereas the property `name` as well as associations to `attributes` are optional.

The OCL invariants of the ADeL metamodel are independent of deployment, namely: (1) Exactly one root node of the type `RUnit` must exit. (2) Identical `hardware` units agree in the values of their capabilities (CPU type and count as well as the sizes of RAM and HD).

The ADeL metamodel was implemented within the Eclipse Modeling Framework EMF 2.4.2 [4] and Eclipse 3.4 Ganymed. The current concrete ADeL syntax corresponds to the graph provided by the EMF.Edit framework [4]; see Fig. 4 in Section 4.3.

4.2 Deployment Constraints

An instance of the ADeL metamodel, i.e., an ADeL model, corresponds to a *deployment plan* that successively assigns the `RUnit` of the root node (which is to be deployed) to hardware (leaf nodes). Only valid deployment plans can be effectuated. To be *valid*, a deployment plan (ADeL model) must satisfy all the `RUnits`' prerequisites (*deployment constraints*) without interfering with the hardware's capabilities (*hardware constraints*). Both groups of constraints are specified as OCL invariants [11] and explained in the following. Due to space limitations, the OCL statements are not given here, but can be requested from the author of this paper.

Deployment and hardware constraints rely on deployment paths, which exploit the association `isLinked` between `units`: A *deployment path* always starts at a `RUnit` and terminates at a unit of the types `hardware` or `RUnit`, respectively. In the first case, the start node is said to be *deployed* and *undeployed* otherwise.

Deployment constraints comprise the invariants `[deployed]` and `[choice]`. The invariant `[deployed]` requires that all `RUnits` that are not optional must be either linked to another unit (the child, which can be hardware) or belong to a non-elementary `RUnit`. The deployment of non-optional, non-elementary `RUnits` is guarded by the invariant `[choice]`: If the group type (`GType`) of a non-elementary `RUnit` is `A/O/X`, then for all/at least one/exactly one non-optional member(s) of the group a deployment path ending at a hardware unit must exist.

Hardware constraints, which are specified by the invariants `[HD]`, `[RAM]`, `[CPU_count]` and `[CPU_type]`, guarantee that the aggregations of prerequisites along *all* deployment paths that target at the *same* hardware unit observe the hardware's capabilities. Consequently, these invariants must be specified in the context of `hardware`, and navigation occurs along the reverse deployment path, i.e., from the leafs of an ADeL model to its root. Reverting the deployment path is achieved by iterating over all instances of the type `RUnit` and selecting parent `RUnits` that are linked with the corresponding (child) `RUnit`; see, e.g., the invariant `[HD]`:

```
context Hardware
inv HD:  RUnit.allInstances()->select(r|r.isLinked->exists(m|m.name=self.name))->
    collect(u|u.aggrHD(u.HD))->sum()<=self.HD
```

All invariants of hardware constraints rely on help functions for specific *aggregations* along the deployment path, i.e., (1) to sum up the required HD size (help function `aggrHD()`, (2) to find the maximum required RAM size or CPU count or (3) to check the equality of the required CPU type.

These predefined OCL invariants are implemented with the tool *Topcased* (http:// www.topcased.org) and must be evaluated for each ADeL model (see Fig. 3). Topcased can also be used to implement additional, deployment-specific OCL constraints.

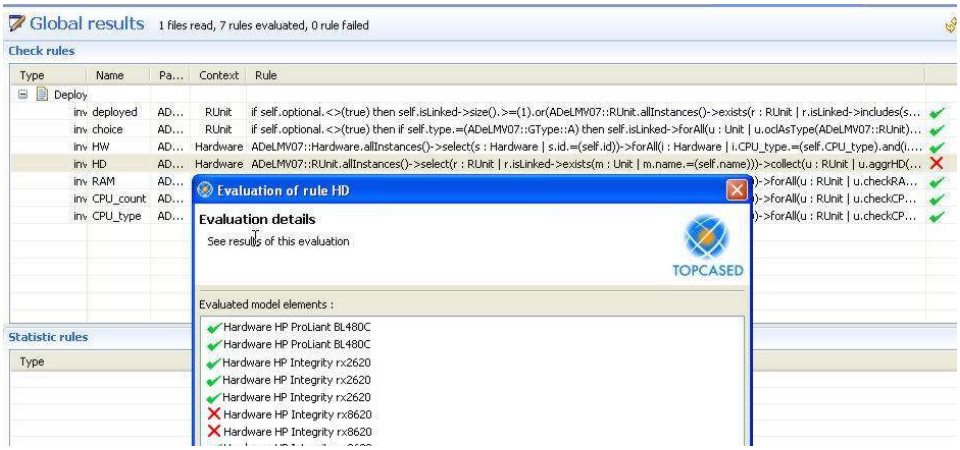

Fig. 3. Evaluation of the ADeL OCL constraints for the ADeL model of Fig. 4.

4.3 Application Example

Fig. 4 depicts the ADeL model for the deployment of SAP SCM (a real-world case investigated in [16]). SAP SCM, the root node, consists of several `RUnits` (the 'SAPKernel', a database instance 'DBSID', the LiveCache 'LID' and the optional optimizer 'OptID'). The `RUnit` 'Install' expresses installation prerequisites (installation media, JRE). The 'SAPKernel' is a conjunctive `RUnit` (`GType = A`) since both the global instance 'SAPSID' and the central instance 'DBSID' as well as the C++ Runtime environment must be installed. The software products that realize the `RUnits` 'DBSID', 'LID' and 'OptID' must be chosen from a set [Rq5]; thus, these `RUnits` are exclusive groups.

The deployment of a `RUnit` is visible from the deployment path to a `hardware` unit. For example, the `RUnit` 'DBSID' is realized by the `RUnit` 'Oracle 10.2' (operating system 'HP-UX 11.23') and installed on the `hardware` unit 'HP Integrity rx8620'.

The additional `attribute` 'SWAP' related to the `RUnit` 'SAPKernel' expresses that additional 20 Gigabyte (GB) of SWAP space are needed. (All sizes are specified in GB in this paper). Moreover, the `RUnit` 'SAPSID' is associated with an `artifact` that specifies the location (`path`) of the directory `/sapmnt`.

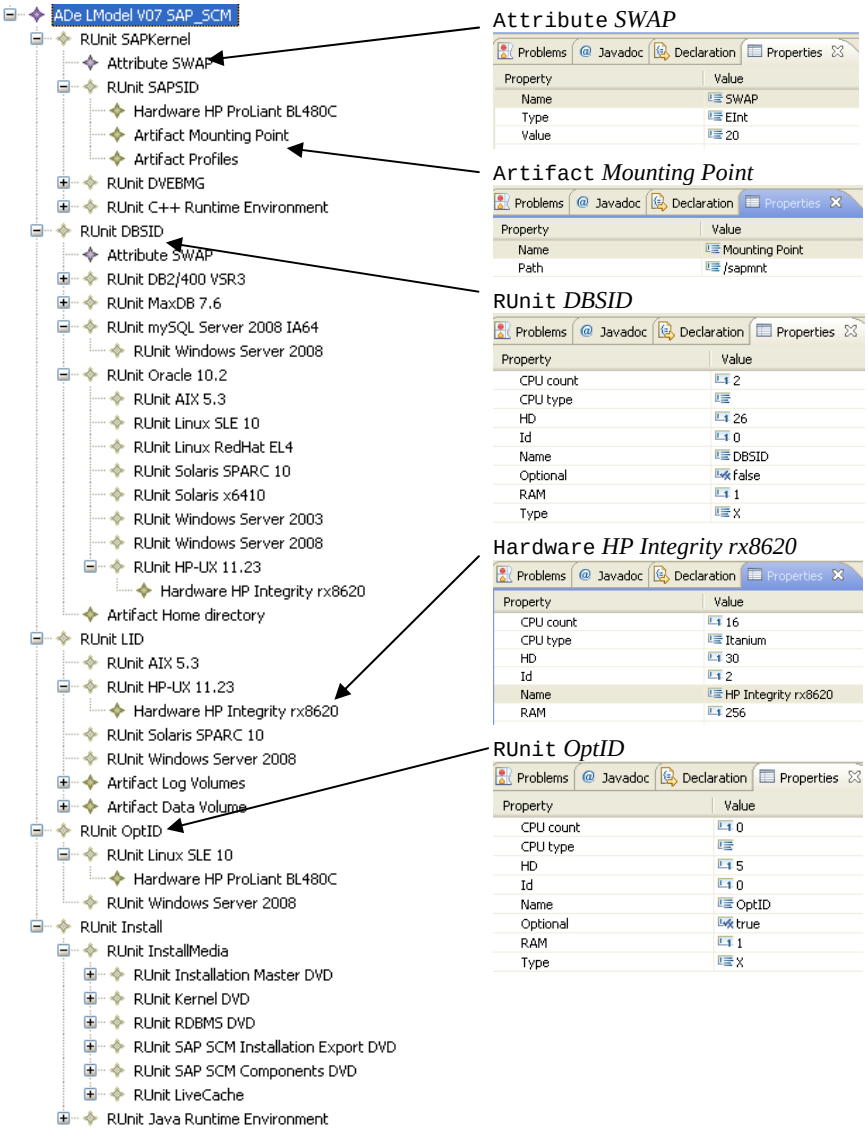

Fig. 4. Concrete syntax of the ADeL metamodel for the example of SAP SCM (extract).

5 Criticism and Future Research

Probably the main objection to the ADeL approach is over simplification, as the abstract syntax is a graph of linked units. However, recent research on enterprise architecture has shown that linked units are the basis to generate any kind of EA visualization and to exchange EA models between tools [9]. Moreover, an extensive type vocabulary becomes burdensome when using the OCL: Deviating types of units must be casted;

recursive navigation along deployment paths is not possible if the link types are distinct. For that reason ADeL does not differentiate between link types.

Though the OCL is a natural choice to express constraints [Rq4] in the field of model-driven development, its appropriateness for the purpose can be doubted: First, as explained above, it affects the ADeL abstract syntax. Secondly, the ADeL hardware constraints require reverse navigation along deployment paths. This can only be achieved by the predefined operation `allInstances()`, which increases the worst case complexity of OCL evaluation [1]. The latter argument can be mitigated by the fact that the number of instances of each type within ADeL models is small, even in real-world deployment scenarios. Nevertheless, a goal of my future research consists in replacing the OCL invariants by another formalism that is capable of handling constraints, e.g., constraint solving techniques. Other topics for future research are the implementation of a more sophisticated editor as well as the definition of transformations to generate installation guidelines and system configurations from ADeL models.

References

[1] Altenhofen, M., Hettel, T., Kusterer, S.: OCL support in an industrial environment. In: Proc. Workshops and Symposia at MoDELS 2006, pp. 169-178. Springer (2007)

[2] Aier, S., Riege, C., Winter, R.: Unternehmensarchitektur-Literaturüberblick und Stand der Praxis. WIRTSCHAFTSINFORMATIK 40, 292-304 (2008)

[3] Anderson, J.R.: Cognitive Psychology and its Implications. 5th ed., Worth, New York (2000)

[4] Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework: A Developer's Guide. Boston et al., Addison-Wesley (2004)

[5] Carzaniga , A. et al.: A Characterization Framework for Software Deployment Technologies. Techn. Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, (1998).

[6] DMTF: Common Information Model (CIM) Standards. CIM Schema: Version 2.24.0 http://www.dmtf.org/standards/cim/cim_schema_v2240/

[7] Frank, U. et al.: ITML: A domain-specific modeling language for supporting business-driven IT Management. DSM Forum 2009 http://www.dsmforum.org/events/DSM09/Papers/Heise.pdf

[8] Hall, R.S., Heimbigner, D., Wolf A.L.: A Cooperative Approach to Support Software Deployment Using the Software Dock. In: Proc. 21st Int. Conf. on Software Engineering (ICSE 1999), ACM Press (1999)

[9] Kruse, S. et al.: Decoupling Models and Visualisations for Practical EA Tooling. In: Proc. of the 4th Workshop on Trends in EA Research (TEAR 2009). Springer (2010)

[10] Lacour, S., Pérez, C., Priol, T.: Generic Application Description Model: Toward Automatic Deployment of Applications on Computational Grids. Rapport de Recherche No 5733, INRIA, Rennes (2005)

[11] Lankhorst, M. et al.: Enterprise Architecture at Work: Modeling, Communication, Analysis. Berlin et al.: Springer (2005)

[12] Lestideau, V., Belkhatir, N.: Providing highly automated and generic means for software deployment process. In: Proc. of the 9th European Workshop Software Process Technology (EWSPT 2003). Springer (2003)

[13] Makin, N.: Deployment modeling in Rational Software Architect Version 7.5, Part I & II. http://www.ibm.com/developer¬works/rational/library/08/{1202|1230}_makin/

[14] Object Management Group (OMG): Unified Modeling Language: Superstructure, Version 2.2, formal/2009-02-02, http://www.omg.org/ (2009)

[11] OMG: UML 2.0 OCL Specification, formal/2006-05-01, http://www.omg.org/ (2006)

[16] Patig, S., Herden, S., Zwanziger, A.: Modeling Deployment of Enterprise Applications - Cases and Conclusions. Preprint No. 224, University of Bern, IWI (2009)