Bus Inf Syst Eng 59(5):315–329 (2017)
DOI 10.1007/s12599-017-0482-4

RESEARCH PAPER

A Multilevel Model of IT Platforms for the Needs of Enterprise IT Landscape Analyses

Monika Kaczmarek-Heß · Sybren de Kinderen

Received: 8 November 2016 / Accepted: 4 April 2017 / Published online: 21 June 2017

Abstract Conceptual modeling supports analyses of IT artifacts and the enterprise action system they are embedded in. However, in this paper it is argued that for IT landscape analyses existing modeling approaches fall short due to, among others, (a) problems with accounting for specifics of the IT domain, e.g., an elaborate technical terminology with various hierarchy levels, and (b) inadequate support for automated analyses within and across those different levels. In this paper, the authors discuss how a designed multilevel model of IT platforms created using the multilevel modeling language FMML^x can help overcome these problems. To this end, limitations of IT platform models created with conventional, two-level modeling languages are shown. Furthermore, benefits resulting from the application of the selected multilevel modeling language are discussed.

Keywords IT landscape analyses · Multilevel modeling · FMML^x

1 Introduction and Motivation

Enterprise modeling supports sense-making of an enterprise by providing abstractions over its enterprise action system (e.g., business processes, goals) and enterprise information system, (cf. Frank 2014a; Sandkuhl et al. 2014). One of the aims of enterprise modeling is to enable enterprise-wide analyses, among others, IT infrastructure analyses (Lankhorst 2013; Antunes et al. 2015). A model-driven IT infrastructure analysis concerns assessment of IT infrastructure with a particular purpose in mind. Although model-driven analyses steadily increase in importance, practical applications show serious limitations when the analyses rely on conventional two-level modeling methods, (cf. Frank 2016; Schmidt et al. 2014). In line with our own observations, these limitations are: (1) a conflict between reuse and productivity, (cf. Frank 2014b)[1], (2) problems with accounting for specifics of the IT domain, specifically, an elaborate technical terminology with various hierarchy levels (cf. Frank 2016), (3) inadequate support for automated analyses, and (4) a lack of mechanisms linking models and operational-level data (Schmidt and Möhring 2016).

These observations lead us to consider an alternative language paradigm, namely multilevel modeling (Atkinson and Kühne 2001; Frank 2014b), which promises, among others, to alleviate the conflict between language reuse and productivity, and to account for the elaborate hierarchies of the IT domain. In this paper, we investigate the suitability of multilevel modeling in general, and of a selected multilevel modeling language Flexible Meta-Modeling and Execution Language (FMML^x) (Frank 2014b, 2016) in particular, for model-driven IT infrastructure analyses. In

Accepted after two revisions by Prof. Dr. Matthes

M. Kaczmarek-Heß (✉) · S. de Kinderen
Information Systems and Enterprise Modeling, University of Duisburg-Essen, Essen, Germany
e-mail: monika.kaczmarek@uni-due.de

S. de Kinderen
e-mail: sybren.dekinderen@uni-due.de

[1] The semantically richer the modeling concepts are, the higher the potential *productivity* gain as domain-specific concepts do not have to be reconstructed from scratch. However, increasing the semantic richness of the modeling concepts lowers the range of their applicability across different contexts. Hence, semantic richness lowers language reuse. Conversely, the more generic the modeling concepts are, the wider their range of *reuse*. However, this genericity implies a lower semantic richness and hence, a lower productivity of the modeling process.

order to identify requirements that a modeling approach should meet to support IT infrastructure analyses, we consider different types of analysis scenarios and reported users' needs (cf. Malavolta et al. 2013; Lago et al. 2015). Then, taking an IT platform as a point of departure, we discuss the fulfillment of the identified requirements by conventional modeling languages and show the limitations thereof. Finally, we contrast the identified limitations with the promises of the selected multilevel modeling approach.

In this paper, we focus on the concept of an IT platform, as it plays an important role in IT landscape analyses (cf. Kaczmarek and de Kinderen 2016). This is because it forms the foundation on which software applications and the corresponding business capabilities build. Therefore, for the needs of IT landscape analyses, we contribute a multilevel model of IT platforms that: (1) accounts for provided functionalities and imposed constraints (in line with our prior findings, Kaczmarek and de Kinderen 2016), (2) reflects the idea that the functionality and constraints on one IT platform type serve as a foundation for another platform, (3) supports multiple hierarchy levels of IT platforms, and, thanks to specific features of FMML^x, (4) supports automated analyses within and across different hierarchy levels, and finally, (5) allows for linking models to the operational-level data.

We follow a design-oriented research strategy (Österle et al. 2011) and construct a multilevel model of IT platforms, which addresses the needs of users with respect to enterprise IT landscape analyses (cf. Malavolta et al. 2013; Lago et al. 2015). We follow an iterative process of building a multilevel model, applying it, learning, and modifying/enhancing the model. We evaluate the obtained result against the identified requirements. To check the applicability of the proposed model, we implement it in the modeling/programming tool XModeler (Frank 2014b) and apply it to selected analysis scenarios.

This paper is structured as follows. First, we introduce model-driven analyses and formulate requirements towards a modeling approach. Next, we confront the identified requirements with existing IT modeling approaches and discuss resulting limitations. Next, we describe multilevel modeling and the modeling language FMML^x, and apply it to create a multilevel model of IT platforms. Then, we show the benefits of the model-driven analyses facilitated by FMML^x. Finally, we discuss our findings and conclude with final remarks.

2 Model-Driven Analyses and Resulting Requirements

A model-driven IT infrastructure analysis concerns IT infrastructure assessment with a particular purpose in mind (Johnson et al. 2007), be it the use of models for cost/benefit analysis of an IT portfolio, IT resource load analysis, or otherwise (Lankhorst 2013).

2.1 Types of Analysis Scenarios and the Role of Modeling

To show the role modeling plays in analyses, we focus on a typology of analysis scenarios by Niemann (2005) and Bucher et al. (2006). They roughly classify analysis scenarios into seven types, out of which we discuss two: dependency analysis and cost/benefit analysis.

Dependency analysis investigates the dependencies (including imposed constraints and offered functionalities) that exist among elements of IT infrastructure and between IT artifacts and business concerns (Bucher et al. 2006; Hanschke 2010). Considering the specific types of dependencies that occur between different elements, an important question for IT management is: what impact would changing an IT artifact (e.g., replacing a technology platform) have on the enterprise action system? Or vice versa: if something were to change in the action system (e.g., the introduction of a new business process), then how would this affect the IT infrastructure and its performance? Enterprise models support such dependency analyses. They have the ability to precisely articulate IT artifacts and different elements of the enterprise action system (e.g., business processes), and have the ability to express dependencies between IT artifacts and the different organizational perspectives (Frank 2014a).

Cost/benefit analysis pertains to (1) an analysis of the IT artifact itself, e.g., of the cost of creating and maintaining it, as well as (2) an analysis of the surrounding action system, e.g., of how effective and important an IT artifact is to the execution of a business process (Bucher et al. 2006). As an example of model-driven cost/benefit analysis, Quartel et al. (2010) and Lankhorst (2013, p. 206) propose a model-driven analysis method for balancing the investment in IT artifacts for an organization. In this method, the modeling language ArchiMate (The Open Group 2013) is used in conjunction with the Bedell method (a method used for IT portfolio evaluation). First, it uses ArchiMate to relate business processes to IT artifacts. Then, the ArchiMate model is marked up with importance values for the business processes and effectiveness values of IT artifacts in supporting business processes. Thereafter, using Bedell, model-driven calculations are done to advice on IT investments, essentially by balancing the effectiveness score of an IT artifact against the importance of the business processes that require it. Considering in addition the portfolio analysis, as proposed by Quartel et al. (2010) and Lankhorst (2013), we observe the role that enterprise modeling can play: it allows for identifying specific IT artifacts for evaluation, as well as relating these IT artifacts

to specific organizational concepts (such as individual tasks in a business process). Such detailed analyses are hardly possible with more generic methods such as Bedell, which in their analysis capability often operate on a higher level of abstraction by focusing on units of analysis such as "information system", and that assume a one-to-one mapping between an information system and a business process (Lankhorst 2013, p. 207).

2.2 Resulting Requirements

Let us consider which requirements an IT infrastructure modeling approach should fulfill to support the model-driven analyses. Already from the aforementioned two types of analysis scenarios, we observe that IT artifacts should be described in relation to the surrounding enterprise action system (R1 – *ability to relate IT artifacts to the enterprise action system*). Also we observe that they require a detailed description, from different points of view, of IT artifacts and existing dependencies (R2 – *offering semantically rich concepts and relations*). For example, in the case of the portfolio analysis we need to not only precisely articulate which IT infrastructure element supports which business process, but also how effective this support is. Moreover, the different types of analyses require information about the characteristics of IT artifacts on different hierarchy levels, e.g., on types, models, editions or specific exemplars/instances. One may be interested, e.g., in how many different types of software platforms we have, or one may be interested in the type and state of all components being part of it as well as the actual software configuration. Thus, the IT artifacts are managed and analyzed on different hierarchy levels (Hanschke 2010), which requires a modeling approach to *account for different hierarchy levels of IT artifacts and their level-specific characteristics* (R3). In addition, models are expected to be able to support managing IT artifacts by allowing to, e.g., compare design time and run time level data (e.g., Bodenstaff et al. 2008). Thus, models should provide abstractions over the large collections of operational IT infrastructure data, so that this data can be used within various types of analysis scenarios (R4 – *accounting for the operational-level data within the model*) (Schmidt et al. 2014; Schmidt and Möhring 2016).

In turn, if we consider existing studies on business needs for IT modeling languages (e.g., Malavolta et al. 2013; Lago et al. 2015), we find the following contradiction. Business users demand a modeling language to be "*simple and intuitive enough to communicate the right message to the stakeholders*" (R5) (Malavolta et al. 2013, p. 871). To foster intuitiveness, it is suggested to keep language concepts close to the domain-specific professional terminology (Frank 2014a). In turn, to foster simplicity, Malavolta et al. (2013, p. 871) advocate avoiding or at least hiding semantic richness of the language concepts. This however, is in apparent opposition to R2 and partly to R3. Avoiding semantic richness is also postulated in the context of demanded *support for reuse* (R6) (cf. Malavolta et al. 2013). This means that the modeling language should provide modeling concepts that can be easily reused across different scenarios. This requires the application of more generic concepts with a generic set of properties, which again is in opposition to R2 and R3.

Moreover, two additional requirements of business users regarding the language mechanism should be mentioned here. Firstly, it is not possible to properly foresee all relevant changes in information needs/analysis needs of stakeholders or in the domain itself (e.g., regulatory, market). Therefore, a mechanism is needed that would *allow users to modify a language specification on demand* (R7) (cf. Malavolta et al. 2013), of course, without loosing the possibility to use supporting modeling tools. Secondly, business users demand more *support for automated analyses* (R8) than is currently offered by existing modeling environments (cf. Malavolta et al. 2013). From the description of the analysis scenarios it follows that the automated analyses should account for different hierarchy levels (cf. R3). Also they should support different calculations based on the values of selected properties of the model elements, both within and across different hierarchy levels.

3 Existing IT Modeling Languages, Fulfillment of Requirements and Observed Limitations

We now discuss to what extent the stated requirements are fulfilled by existing IT modeling languages. Following this, focusing on one language, namely the IT Modeling Language (ITML), we discuss limitations of approaches that follow the conventional two-level paradigm.

3.1 Evaluation of Selected Modeling Languages

Several enterprise modeling languages exist that support modeling of IT infrastructure in the context of an enterprise action system (cf. R1). Those languages adhere (with a few exceptions) to the traditional meta modeling paradigm (corresponding to the meta object facility (MOF)) and offer modeling tools with the semantics of the mainstream object-oriented programming languages. Although conceptual overlaps between the existing enterprise modeling approaches can be assumed, they differ substantially in terms of the domain coverage and semantic richness of offered concepts. In the following, we focus on two distinctive ones, namely ArchiMate (The Open Group 2013)

and Multi-Perspective Enterprise Modeling (MEMO) (Frank 2014a). We have selected these specific modeling approaches considering, on the one hand, the extensive capacity to relate perspectives and popularity of the language (ArchiMate, as shown, e.g., by Malavolta et al. 2013), and on the other hand, the expressiveness of the IT domain (MEMO ITML, as shown, e.g., by Kaczmarek and de Kinderen 2016). Whereas ArchiMate favors a concise language design, MEMO offers an extensive language design in the form of a family of domain specific modeling languages (DSMLs). These DSMLs, among them, the IT modeling language (ITML) (Heise 2013), offer comprehensive reconstructions of the technical languages that domain experts are familiar with (cf. Frank 2014a). The difference between both languages is clearly visible, if we consider how they account for software and hardware artifacts and dependencies between those (cf. Table 1).

As can be seen (cf. Table 1), ArchiMate provides generic concepts (with no attributes) (cf. R5), thus it facilitates reuse (cf. R6). However, as the concepts are generic and do not possess attributes, there is not enough information to conduct most of the analysis scenarios (cf. R2–R4). Therefore, the relevant information needs to be added at a later stage in order to perform a selected analysis type (cf. Florez et al. 2014). In addition, the created models are not fit for (automated) quantitative analyses (cf. R8), therefore intermediate translation steps (e.g., a normalization step, cf. Lankhorst 2013, p. 201), or transformation into a different formalism (cf. Johnson et al. 2007) are required. In addition, the offered concepts do not provide relevant abstractions over operational-level data (cf. R4).

In turn, ITML provides more differentiated elements (cf. R2). These express a variety of types of dependencies between different types of software and hardware artifacts, thus ITML is more geared to support productivity, but not reuse (cf. R6). However, although different analysis scenarios are supported by the offered tool (cf. Bock and Frank 2016), the level of support for the performed analyses is still deemed unsatisfactory. This especially concerns different hierarchy levels, which are not accounted for in the language (cf. Kaczmarek and de Kinderen 2016). Similar to ArchiMate, analyses of enterprise models are characterized by a lack of information from the actual systems and, although the meta modeling language used allows for accounting for the instance level (Frank 2014a), this is not accounted for in the supporting modeling tool (Bock and Frank 2016). Thus, R3–R6 and R8 remain unfulfilled.

Finally, when it comes to R7, both languages offer different mechanisms allowing to modify a language specification: meta model customization (in the case of MEMO) and language-built-in mechanisms (in the case of ArchiMate). However, as explained in detail by Atkinson et al. (2015), those mechanisms are not satisfactory and cause notable problems. For instance, in the case of meta model customization, if it is at all supported by a given modeling environment (as often a meta model is 'hard-wired' in the tool, cf. Atkinson et al. 2015), in the standard meta modeling environment it requires a recompilation/redeployment step so that the change can be reflected into the tool.

We argue that the lack of the fulfillment of the identified requirements by the existing modeling languages results from the limitations of the currently dominant language architecture and as such, cannot be addressed by, e.g., modifying or extending them. To illustrate this, in the next section we undertake an attempt to extend ITML (as it already offers a rich set of concepts) with the required aspects.

3.2 Model-Driven Analysis with ITML

To discuss, using the example of ITML, limitations of the conventional modeling paradigm, we focus on analyses in the context of IT platforms.

As we have shown in our previous research (cf. Kaczmarek and de Kinderen 2016), different understandings of an IT platform exist that range from a hardware-oriented to a software-oriented interpretation, down to architecture/industry/technology platforms, or a platform as an enabler of innovation. As a response to this diffuse understanding we have proposed, based upon Sun et al. (2015) and

Table 1 The selected concepts supporting modeling of an IT Landscape – an overview

Approach	Software	Hardware	Relation types	Attr.	Constr.
ArchiMate	Artifact, Node, SystemSoftware, ApplicationComponent, ApplicationInterface, App. service, Infrastr. service	Node, Device	Access, association, used by	–	–
ITML	Software, Architecture, License, SoftwareProduct, SoftwareInterface, ComunicationProtocol, Data, FormatType	Computer, PhysicalDataMedium, NetworkComponent, Network, Printer, Scanner, Fax	Multiple domain relations, e.g.: used in, requires, runs on	Multiple attributes defined for each concept	Numerous OCL constraints

Sangiovanni-Vincentelli and Martin (2001), to perceive an IT platform as "a baseline piece of hardware or software that enables the development of, and imposes constraints on, other software or hardware. In turn, the other software or hardware can by itself be an IT platform" (Kaczmarek and de Kinderen 2016, p. 77). Here a key characteristic is that a platform *enables* and *constrains* development of further hardware and software, by offering, e.g., a certain instruction set (in the case of hardware), a software interface, or a set of methods. Thus, on the one hand, a platform defines constraints on an application, because it is written for a specific platform. On the other hand, a platform provides functionalities that developers can take advantage of. Furthermore, the functionality of, and constraints on one type of IT platform serve as a foundation for another type of a platform (cf. Table 2). Finally, IT platforms come with different hierarchy levels, i.e., there exists a hierarchy of concepts, where more general concepts are refined into more specific ones (e.g., Software platform, Software Server, Web Server, Apache Server, TomCat – cf. Fig. 3).

To conceptualize IT platforms, as stated, we extend MEMO ITML. In conventional meta modeling, one describes domain concepts and their relations using a meta model (i.e., M_2) (language specification) and subsequently, this meta model can be instantiated on the type level (i.e., M_1) (language application). Therefore, we need to modify the ITML's meta model. As we extend ITML from the MEMO language family, we use MEMO's common Meta Modeling Language (MML) (Frank 2011). MML uses concepts common to meta modeling, such as meta types, attributes and associations. In addition, it provides the possibility to model intrinsic features and intrinsic relations (marked by the white literal "i" on a black background), which are to be instantiated only on the instance level (M_0) and not on the type level (M_1). In addition, MML allows for modeling so called Language Level Types – visualized with a black name of the concept on a grey background – which specify concepts that represent instances already on the type level (M_1) and cannot be instantiated further (Frank 2011, pp. 23–24).

A meta model excerpt, corresponding with the presented understanding of IT platforms, is provided in Fig. 1. Trying to find a balance between productivity and reuse, and taking into account the modeling constructs we can use, we model a meta type *Software* with three main specializations *OperatingSystem*, *TechnologyInfrastructureSoftware*, *ApplicationSoftware*. Also, we model two meta types: *HardwarePlatform* and *SoftwarePlatform*. To account for specific kinds of *HardwarePlatform* we use an enumeration attribute *caseType*. In addition, *HardwarePlatform* has a *ProcessorArchitecture* and finally, an *OperatingSystem* running on it. In turn, a *Software* offers a *SoftwareInterface* which in turn provides a *Method* covering a *Functionality*. Software can communicate (*SoftwareCommunicationRelation*) with some other *Software*. Due to the space limitations and to increase the meta model's understandability, only exemplary attributes, values of enumerations, and constraints specified in the Object Constraint Language (OCL) have been included (Fig. 1).

Analysis possibilities To illustrate model-driven analyses, Fig. 2 shows IT platforms of a fictitious insurance company called ArchiSurance, from Jonkers et al. (2012) and extended by Kaczmarek and de Kinderen (2016). They are modeled using the extended ITML with an exemplary concrete syntax. The diagram is divided into two main parts. First, the *IT Landscape* part of the diagram depicts different platforms running at ArchiSurance: a hardware platform, a system platform, a technology platform and an application platform combined with application software. This model is integrated with a business process model created using the MEMO Organisation Modeling Language (OrgML) (Frank 2014a). The created diagram allows developers and analysts, by browsing it, to e.g., (1) *Assess compatibility with existing legacy applications* – e.g., ArchiSurance has a legacy financial application written in COBOL, which clashes with the overall use of the Java platform; (2) *Assess business IT alignment* – we learn that the financial legacy application, which clashes with the Java platform, supports ArchiSurance's business process for paying out insurance claims; (3) *Assess the level of*

Table 2 Different types of an IT platform, based on Kaczmarek and de Kinderen (2016)

Type of platform	Understanding
Hardware platform	The type of a processor and/or other hardware on which a given operating system runs. It can also refer to the type of system in general (e.g., mainframe, workstation, desktop)
Software platforms:	
Operating system platform	An operating environment under which various smaller application programs can be designed to run. An operating system is installed on some dedicated hardware platform
Technology-oriented and infrastructure platform	A piece of software enabling the realization of applications, rather than software offering specialized functionality to an end user, e.g., a virtual machine, enterprise system platform
Software application platform	A piece of software under which various smaller application programs can be designed to run

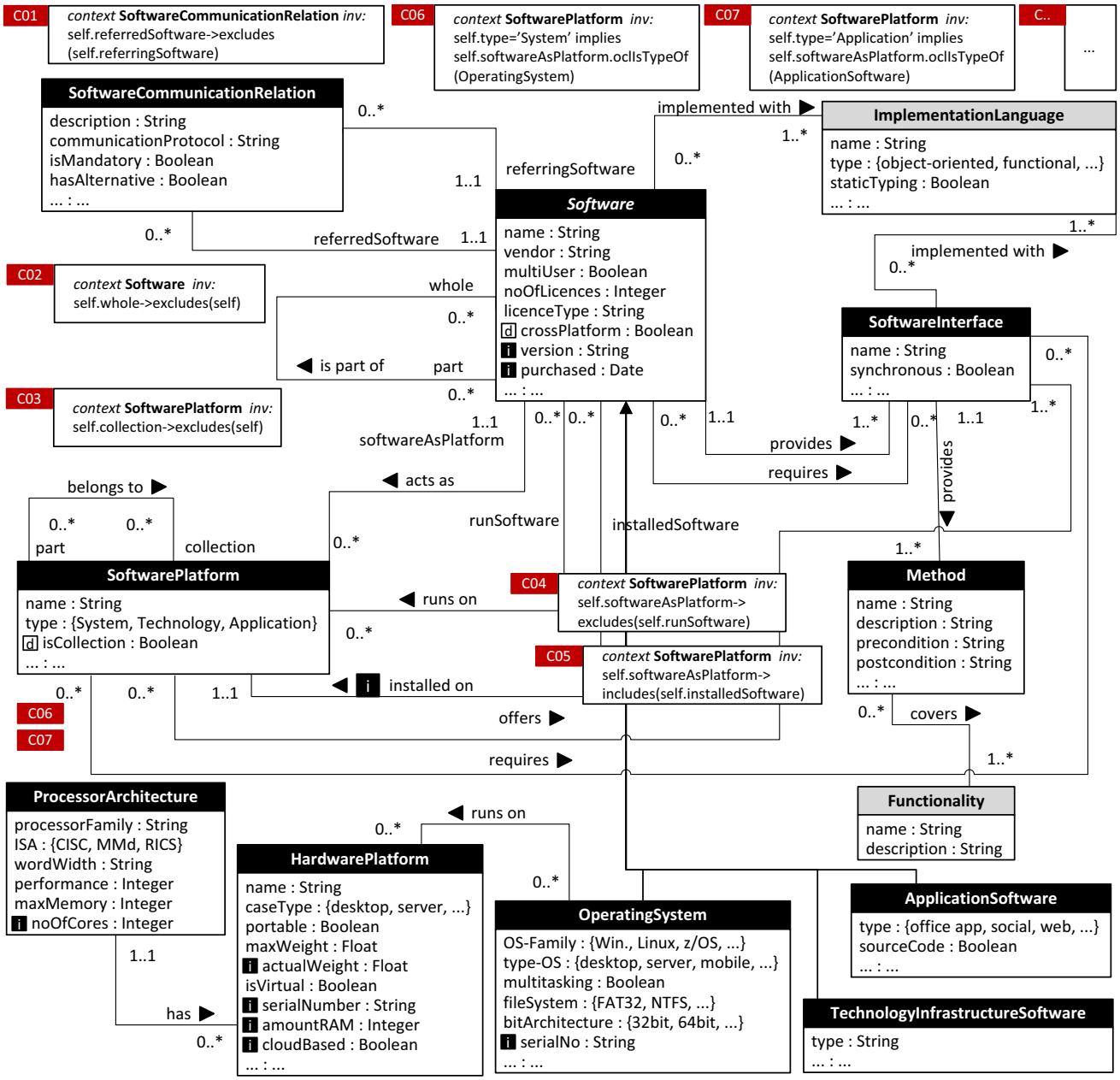

Fig. 1 Excerpt from the extended ITML meta model (Kaczmarek and de Kinderen 2016)

interoperability – e.g., ArchiSurance has different hardware platforms that impose constraints on the development of new applications.

Observed limitations As illustrated, we can use the created models to conduct additional analyses compared to the original ITML. However, the requirements are still unfulfilled as, among others, (1) the analyses need to be mostly performed by navigating different diagrams and relations between them (cf. R8), (2) to perform different analyses, still a different level of abstraction is required to draw meaningful conclusions (cf. R2 and R3), (3) flexibility is missing that would allow to move between different hierarchy levels (cf. R3), (4) the models do not account for operational-level data (cf. R4). As shown subsequently, extending the ITML's specification further (e.g., with the hierarchy of IT platforms), will not change this situation. We argue that this is due to limitations imposed by (a) mainstream object-oriented programming languages used to implement the corresponding modeling tools (R4, R7–R8), and (b) relying on the traditional two-level conceptual modeling approach (R2–R3, R5–R6).

First, mainstream object-oriented programming languages feature only one classification level. Therefore types or even meta types are represented as objects by

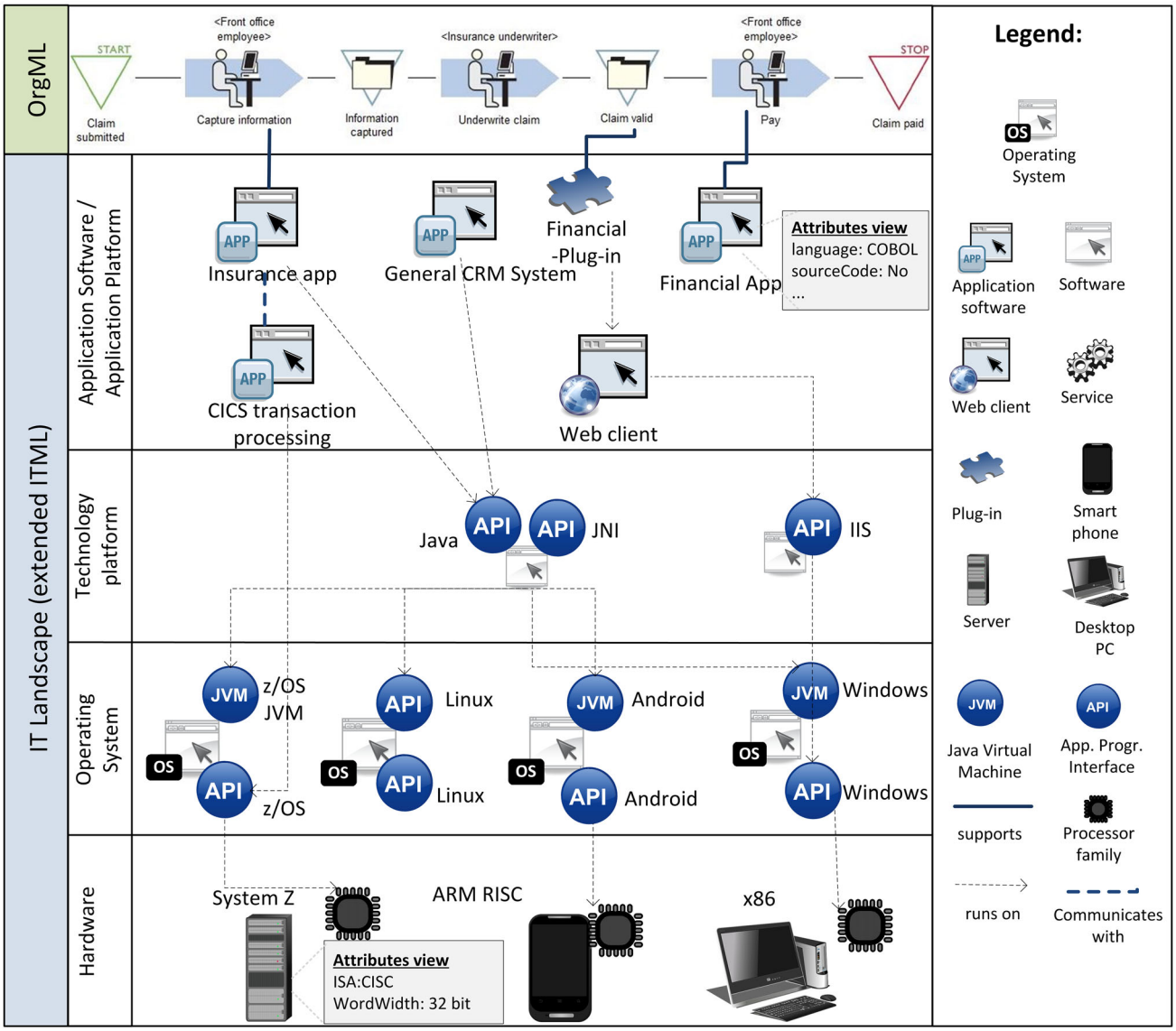

Fig. 2 An exemplary extended ITML diagram integrated with OrgML

overloading the M_0 level of a programming language. As a result a common representation of code and model is not possible (Frank 2014b). Thus, not only a recompilation of modeling tools is required whenever we want to change something in the language specification, but also equipping model elements with operations (to support automated analyses), or linking them with operational-level data, is hardly conceivable (cf. Frank 2014b, 2016).

Second, there are limitations that inherently come with using the two-level modeling paradigm. We illustrate them by referring to the hierarchy of IT platforms we want to account for.

A fixed number of classification levels: IT platforms exist in a remarkable variety of types (e.g., hardware platforms, technology platforms, cf. Table 2), each of them possessing a variety of type-specific attributes and further hierarchies (cf. Fig. 3).

Considering the identified requirements and with the aim to avoid conceptual redundancy, we are interested in making this hierarchy part of a language specification (M_2). Thus, we model, e.g., an additional meta type *SoftwareServer* and specialize it in, e.g., *WebServer*, which in turn may be specialized into specific products *ApacheHTTP*, *TomCat* and specific editions (e.g., *TomCatv8*) (cf. Fig. 3). However, by deciding to represent the refinement relations between the elements of the presented hierarchy as specializations, we would be dealing with a so called level mismatch problem (cf. Atkinson and Kühne 2008) as various domain levels/hierarchy levels are mapped onto exactly the same model level (i.e., M_2). Although

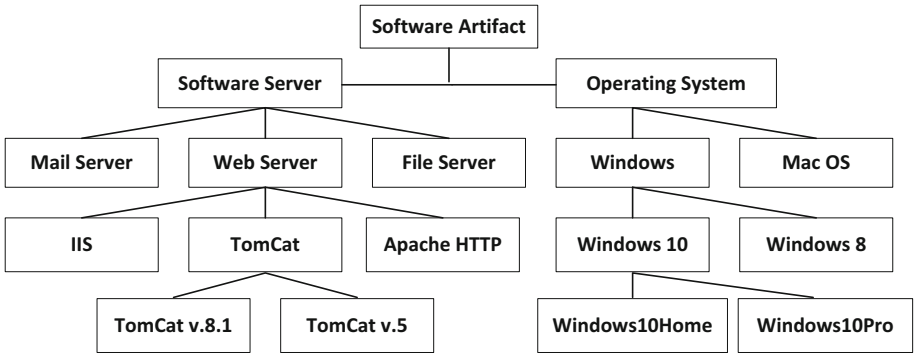

Fig. 3 Excerpt from an exemplary hierarchy of IT platforms

it is certainly *technically* possible to overload the (M_2) level, by relying on specialization, this would constitute a *workaround*. Such workarounds are necessary since ITML, based on the traditional two-level paradigm, natively does not offer constructs to mirror the hierarchies that naturally exist in the IT infrastructure domain.

Problems with incorporating relevant information: as we use the specialization relation to model the platform hierarchy, moving along it we are able to extend the definition of specialized meta types (e.g., adding information that a *WebServer* has additional attributes such as *kernelSpace*, *userSpace*). However, we face problems when trying to incorporate relevant information, in particular when trying to assign values to attributes of meta types. This is because in conventional meta modeling *meta types cannot have a state*. To illustrate this consider that we want to specify that *TomCat* is produced by the organization 'Apache' and made available on the terms of the Apache license. To do it we would need to assign values to inherited (via specialization) attributes: *vendor* and *licenceType* of the meta type *TomCat* (M_2). However, since meta types are stateless, this would not be possible. The only solution would be to define an OCL constraint that would state that each instance of *TomCat* (i.e., on M_1) needs to have, e.g., the value of attribute *vendor* set to 'Apache'. This however, not only increases the complexity of the model, but also such OCL constraints are not always supported by modeling tools.

No associations between objects on different levels: in conventional meta modeling the only relation allowed between different levels (e.g., M_2, M_1) is the instantiation relation (Atkinson and Kühne 2008). As a result, we cannot link a concept defined as part of language specification with a concept that is part of language application. To illustrate this, let us consider again *TomCat* being produced by 'Apache'. Instead of using an attribute *vendor*, we could decide to model a meta type *Vendor* (M_2) and define a relation that each *SoftwareServer* (and thus, also its specializations) has some *Vendor*. However, when we would like to relate *TomCat* (a specialization of a *WebServer*, M_2) with the relevant vendor, it turns out to be impossible. This is because the *Apache* organization would be in most cases defined as an instance of *Vendor*, so would be part of language application (M_1) and not language specification, as *TomCat*. Alternatively, we could define both *TomCat* and *Apache* on M_1 (i.e., *TomCat* would be an instance and not a specialization of *WebServer*) and then state the relation between them. This would however lead to *redundancy* in corresponding models, as the information not stated in the language specification would need to be always added during the use of the language.

Summarizing, we see that the identified requirements for IT infrastructure modeling languages cannot be satisfactorily addressed with a conventional language architecture. A satisfactory solution is understood as a solution that would promote model integrity, avoid model redundancy, and that would allow to express all relevant knowledge at the relevant abstraction level. Thus, as a response, we now introduce multilevel modeling and discuss to what extent this would be a suitable instrument for model-driven IT infrastructure analyses.

4 Multilevel Modeling and FMML^x

The need to deal with more than two levels of classification has been receiving increasing attention over the years (cf. Atkinson and Kühne 2001, 2008; Frank 2014b, 2016; Carvalho and Almeida 2016). A number of multilevel modeling approaches have been proposed, among them, a potency-based multilevel modeling (also called deep instantiation) (Atkinson and Kühne 2008), multilevel objects and multilevel relationships (m-objects and m-relationships) (Neumayr et al. 2009) and lately, the Flexible Meta-Modeling and Execution Language (FMML^x) (Frank 2014b) as well as Multilevel Theory (MLT) (Carvalho and Almeida 2016). Three features are shared by all multilevel approaches (cf. Table 3): (1) the support for arbitrary-depth classification hierarchies,

Table 3 Comparison of selected features of selected multilevel modeling approaches

Feature	Deep inst.	FMML^x	m-objects	MLT
(1) Multiple levels of classification	●	●	●	●
(2) Relaxed type/instance dichotomy	●	●	●	●
(3) Deferred instantiation	●	●	●	●
(4) Accounting for operations	○	●	○	○
(5) Common rep. of code and model	○	●	○	○

○ not supported/not offered, ● supported

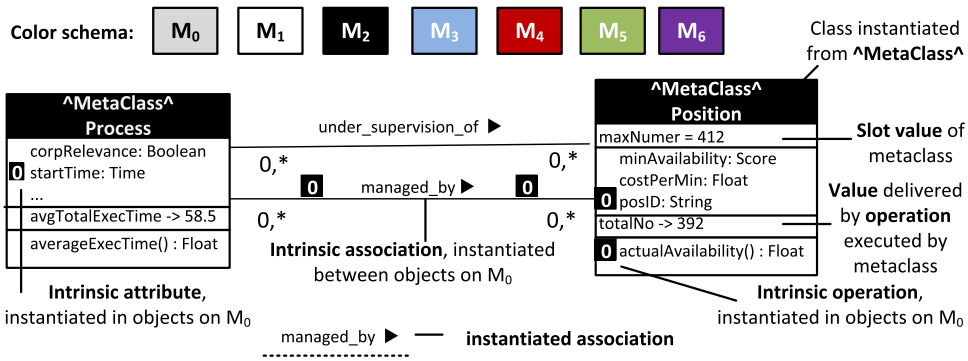

Fig. 4 FMML^x concrete syntax, cf. (Frank 2014b)

(2) relaxing the type/instance dichotomy, implying, among others, that types (classes) can have a state, i.e., that their attributes can be assigned with a value, and finally, (3) offering a deferred instantiation mechanism[2]. However, these languages differ when it comes to (1) the way these features have been implemented, (2) their applicability and expressiveness, and (3) additional mechanisms and tool support offered (cf. Neumayr et al. 2011; de Lara et al. 2014). For instance, to express a deferred instantiation, a deep instantiation approach uses a special construct 'potency' that may be assigned to attributes only, MLT uses the notion of 'regularity' attributes, whereas in the case of FMML^x the concept of 'intrinsic properties' is applied, which may be used together with attributes, operations and relations.

FMML^x exhibits distinctive features compared to other approaches (cf. Table 3), among others, it accounts for operations as well as offers a common representation of model and code. As those features support fulfillment of identified requirements, as explained subsequently, therefore, FMML^x becomes our language architecture of choice. Whereas a full description of FMML^x can be found in Frank (2014b), here we briefly explain its most relevant features.

FMML^x is based on an extension of XCore (Clark et al. 2008), which is the meta model of an executable meta modeling facility (XMF) (Clark and Willans 2013). XCore allows for an arbitrary number of classification levels, which is accomplished through a recursive and reflexive language architecture. In XCore a meta class *Class* inherits from *Object*. At the same time, objects (instances of *Object*) are instantiated from *Class*. As a result, every (meta) class is an object and can have state (Frank 2014b). In addition, the recursive language architecture allows for a common representation of code and model (Frank 2014b). This means that (1) model elements are classes, which allows for the definition of attributes and operations, and also (2) classes are objects, so we can assign values and execute operations (cf. R8).

In addition, the definition of attributes, operations and relations has been equipped with an additional property: *intrinsicness* (Frank 2014b). This property obtains an integer value that indicates the level on which a given attribute, relation or operation will be instantiated. For instance, while defining a meta class *Process* (on M_2) we can express that it has an attribute *startTime* which can only be instantiated (meaning: it obtains a value) on M_0 (cf. Fig. 4). The level of instantiation is indicated by placing a white number in the black box next to the definition of the element in question (cf. the FMML^x concrete syntax, Fig. 4).

Finally, FMML^x is supplemented by a meta modeling and programming environment, called XModeler (Frank 2014b). XModeler offers an environment that, among others, allows to integrate the external data sources into a model.

[2] Deferred instantiation allows to define on each level of abstraction invariant commonalities which are relevant for our purposes, and deferring their instantiation to some not directly proceeding lower level.

5 A Multilevel Model of IT Platforms

In this section, we apply FMML^x to model different aspects of IT platforms. We express it in terms of what we refer to as a *multilevel model*, so as to reflect the arbitrary number of classification levels of IT platforms. Furthermore, we remove the prefix 'meta' from model because in multilevel modeling the separation between language specification and application is blurred. Due to space restrictions, we discuss an excerpt of the multilevel model (Fig. 5). In order to increase its understandability, most of the model's properties are omitted, which is indicated by an additional line with "... : ...".

Following our understanding of an IT platform, on the most abstract level, we distinguish two main categories of a platform: hardware and software. In the multilevel model, this is reflected in a division into *HardwareComponent* and *SoftwareArtifact* (M_5). A set of (intrinsic) attributes and relations defined for each of them reflects our domain knowledge on this abstraction level. In addition, *HardwareComponent* as well as *SoftwareArtifact* have operations (mainly instantiated on lower levels), which allow for running desired analyses. This can, e.g., pertain to a calculation of the amount of categories (cf. *ComputingDevice*, M_4) or the total maintenance cost (cf. 3000 Euros for *WebServer*, M_3) for the needs of cost/benefit analysis. Finally, the multilevel model includes relevant relations, which in FMML^x can be cross-level. For instance, we know that each *HardwareComponent* (M_5) provides *Functionality* (M_2) and this relation will be instantiated on M_2 and M_1, respectively.

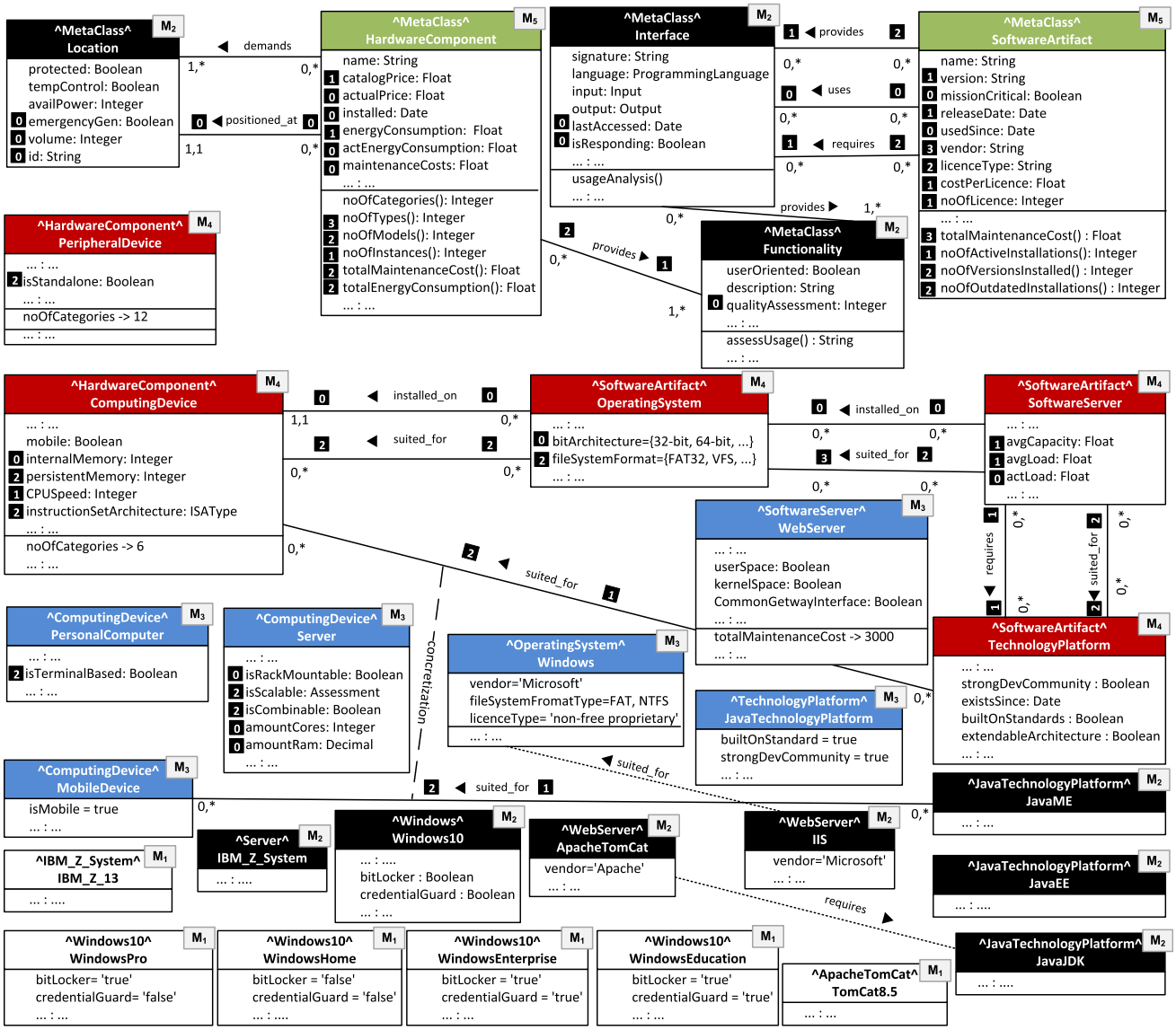

Fig. 5 Excerpt from the multilevel model, covering M_5 to M_1

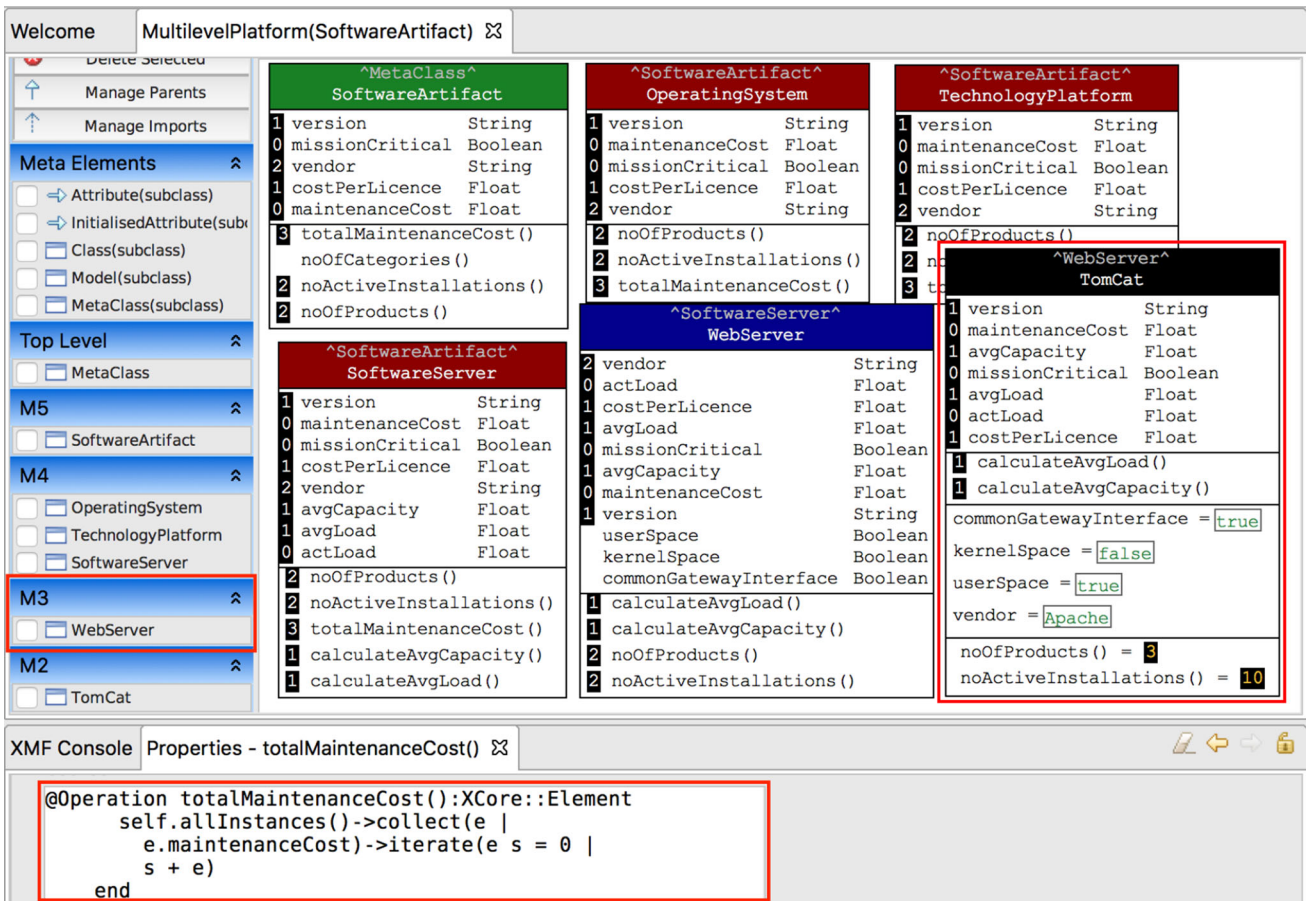

Fig. 6 Screenshot of the multilevel model implemented in XModeler

HardwareComponent and *SoftwareArtifact* are instantiated, among others, into different types of hardware and software platforms. If we move along the created hierarchy, for each class we can define additional attributes and operations for expressing particular phenomena. Furthermore, we can instantiate attributes and relations defined on higher levels. For example, the class *Windows* can define the attribute value for *vendor* as 'Microsoft' (Fig. 5). Also, already defined relations can be concretized. For example, the relation 'suited for' between the classes *ComputingDevice* and *TechnologyPlatform*, can be concretized in the relation 'suited for' between *MobileDevice* and *JavaME*.

In turn, Fig. 6 presents a screenshot from the XModeler tool with an excerpt of the multilevel model. It shows the class *SoftwareServer* (M_4) (being an instance of a *SoftwareArtifact*, cf. Fig. 5) and its instance, a class *WebServer* (M_3). A *WebServer* class specifies additional attributes such as *userSpace*, *kernelSpace*, or *CommonGatewayInteraface*. A *WebServer* is further instantiated into different types, namely *TomCat*, *ApacheHTTP* and *IIS*. Please note that if a user wants to model a different type of a server, a user can instantiate the *WebServer* class using the modeling palette of XModeler (cf. Fig. 6). A *SoftwareServer* meta class defines a set of operations that can be executed at the corresponding level. For instance, there are operations allowing to calculate average load or average capacity of servers on M_1. Moreover, it is possible to obtain the information regarding the number of categories or the maintenance cost by executing the relevant operations (defined at the M_5 level within the *SoftwareArtifact* meta class). Various analyses regarding the performance in the selected time horizon may be also implemented as methods. The types of WebServers may be further instantiated into specific versions and the additional information may be added, e.g., that TomCatv8 requires at least Java v7. We do it by instantiating the relation defined on the upper level. Finally, on M_0 we deal with a specific server version running on a specific platform. This illustrates that XModeler both (1) accounts for abstractions over the relevant operational-level data, and (2) we could feed it with data coming from the operational systems.

6 Multilevel Model-Based Dependency Analysis

In this section, by referring to the extension of the ArchiSurance scenario (Sect. 4), we show additional

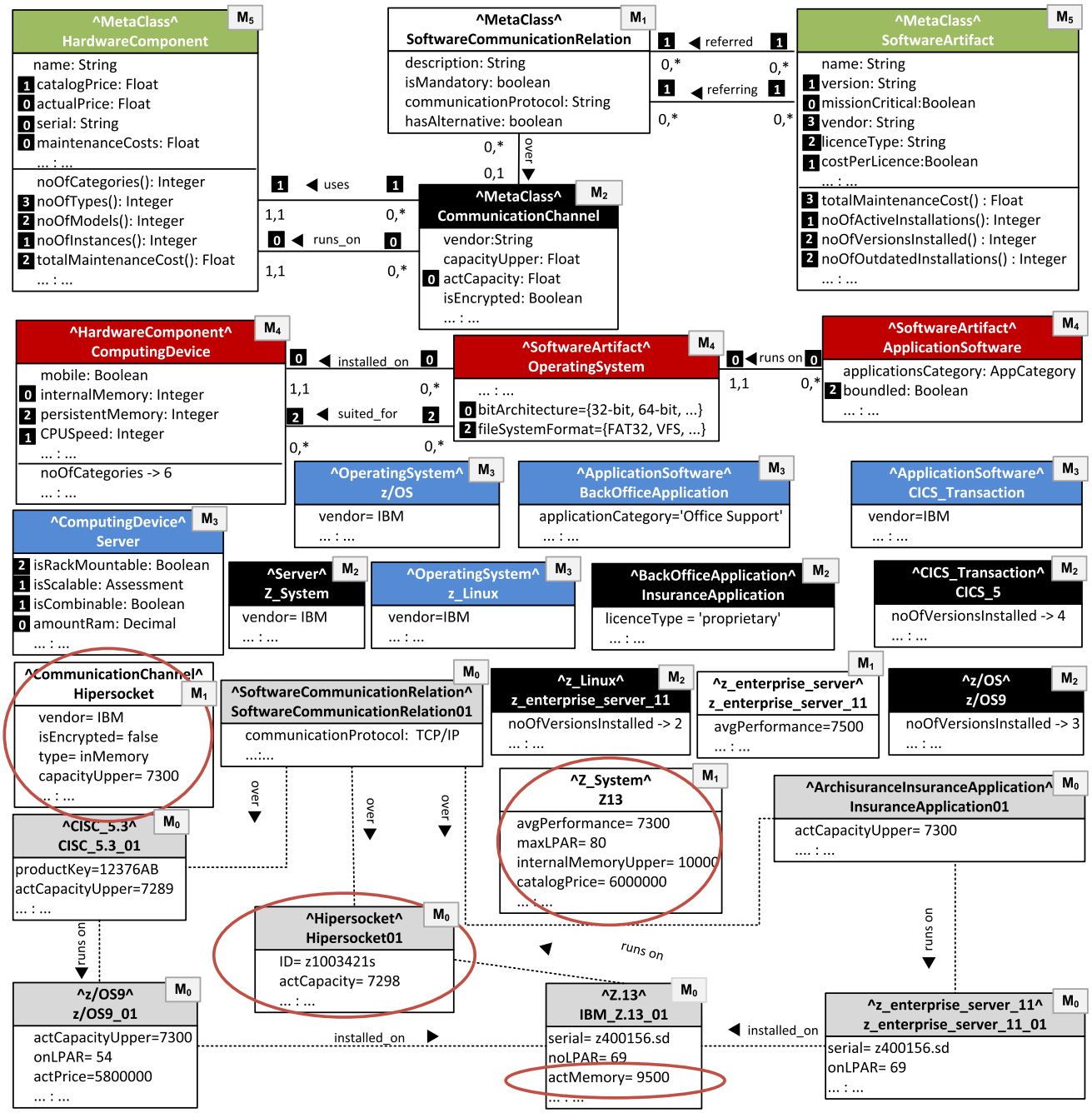

Fig. 7 Multilevel model of a mainframe – an excerpt with highlighted aspects

mechanisms and enhanced analyses possibilities enabled by FMML^x.

In particular, we consider the software communication relation between the insurance application running on Linux, and the CICS transaction processing middleware running on z/OS. This communication takes place over a communication channel.

Suppose that ArchiSurance decides to transition its insurance application to a windows-based machine, to reduce its dependence on the (legacy) mainframe. After the transition, ArchiSurance notices a significant drop in throughput over the communication channel, turning this into a bottleneck for the number of transactions that the insurance application can process. To find the cause of this bottleneck, ArchiSurance uses a multilevel model as an instrument to inspect dependencies in its IT infrastructure, cf. Fig. 7. It learns that prior to the transition to Windows, the Linux-based insurance application was running on the

same hardware as the z/OS-based CICS middleware – by means of virtualization[3], this hardware was split into multiple partitions. As a result the communication channel used for communication between the insurance application and CICS, while functioning the same as an ordinary communication channel (thus allowing, e.g., the use of a TCP/IP connection), was actually *virtualized* in the RAM memory of the same machine. This virtualization offers performance advantages compared to an actual network connection, whereby data needs to take a 'detour' over hardware networking infrastructure.

Considering the use of multilevel modeling, first, we observe how we benefit from relaxing the type/instance dichotomy. In particular, while moving down the level of abstraction for the communication channel, we can automatically derive/obtain specific values. In the mainframe multilevel model, this is particularly visible for the class 'Hipersocket' (Fig. 7). Once it is known that the communication channel is actually a Hipersocket, we obtain four attribute values. Namely that the vendor is IBM, and that a Hipervisor operates inMemory, meaning that it is by definition a virtualized communication channel. Also, we can notice that the communication channel is not encrypted, encryption being unnecessary with the communication taking place over a virtualized communication channel. Finally, taking into account that we now also know the maximum memory capacity of the server that the hipersocket runs on (the Z13 server), we can also infer an upperbound to the capacity of this communication channel (7300 transactions per second).

Second, to compare actual performance of the communication channel between a physical network connection and the hipersocket, we benefit from the common representation of model and code. We can thus update the model with instance level organizational data, coming from the running hipersocket (7298, for the particular performance snapshot shown in Fig. 7), as well as for an actual network connection (not shown due to space restrictions).

Third, the flexibility to move between different hierarchy levels allows us to 'zoom in' on the ArchiSurance scenario. In this case, it has allowed us to observe that the communication channel for applications running on different partitions of the mainframe is virtualized, resulting in a performance increase compared to traditional network set-ups. Thus, multilevel modeling allows us to gain a better *understanding* of what is going on underneath, if so desired.

[3] Virtualization of hardware resources is a common feature of mainframes, in particular it allows (1) to offer redundant (virtual) partitions, which helps with fault tolerance, and (2) to help distributing the ample hardware capabilities offered by a mainframe into smaller, more manageable pieces.

In the multilevel ArchiSurance scenario, the gained understanding leads to the decision to move the insurance application back to the Linux OS. This way, ArchiSurance can capitalize on the virtual network connection capability that is possible only because CICS and the insurance application are running on two virtualized partitions of what is essentially the same hardware.

7 Discussion

The application of FMML^x to model IT platforms leads us to the following observations. By allowing multiple classification levels, FMML^x makes it possible to use concepts that correspond directly with the hierarchy levels of the concepts used by domain experts (R3 and R5). We can choose the level of abstraction (or level of details) we want to deal with (cf. R5) and take advantage of the defined operations to satisfy information needs of different stakeholders (R1–R3). Furthermore, by offering a flexible number of levels, multilevel modeling has the potential to offer a user exactly the knowledge required at a particular level of abstraction. For instance, to have all knowledge available on 'Servers', but equally on 'Apache Servers'. As a result, multilevel modeling actually alleviates the conflict between reuse and productivity present in the traditional two-level paradigm. It accounts for *reuse* (R6) by providing, at each classification level, all knowledge relevant for a given context, while at the same time accounting for *productivity* by providing the possibility to express semantically rich concepts at each classification level (R2).

A common representation of model and code in an integrated modeling and programming environment (XModeler) avoids the cumbersome model and code synchronisation problem, and treating classes as objects allows to define relevant operations supporting different analysis scenarios (R7). Thanks to the recursive architecture of XMF and dynamic typing (Frank 2016), the introduction of changes to the language definition in the accompanying software tool does not require an additional recompilation step. This eases modification of a language (R7). In addition, XModeler as a programming environment allows to link to external applications. Thus, we can ensure that enterprise models are up-to-date by linking them to existing data sources/applications (R8).

However, we experience several challenges. Firstly, new 'principles' need to be followed to use multilevel modeling, because it imposes a paradigm change (e.g., no type/instance dichotomy, an arbitrary number of levels). This requires a change in the mindset for those used to modeling with the conventional paradigm. Secondly, as there is no limitation regarding the classification levels, a language designer requires the ability to think in terms of multiple

classification levels. On the one hand, the application of multilevel modeling allows for a straightforward statement of domain phenomena. Thus, this leads to the creation of models without overloading levels. However, the possibilities offered by this new modeling paradigm (e.g., using various classification levels, equipping the model elements with behavior) lead to models that are more complex and semantically richer than their conventional counterparts would be. Therefore, depending on the scenario, we should equip supporting tools with visualization and customization mechanisms that would facilitate fading out irrelevant parts of a model. Finally, there is a need to define methods to support the design of multilevel models and their application.

8 Conclusions

In this paper, we used the concept of an IT platform and corresponding analyses to (1) discuss limitations of the conventional two-level modeling, and (2) show the benefits resulting from the application of the multilevel modeling language FMML^x. We have shown that multilevel modeling indeed better addresses the identified requirements due to, among others, (1) introducing an arbitrary number of abstraction levels, (2) relaxing the type/instance dichotomy, and, in the case of FMML^x, (3) offering a common representation of model and code. Also, considering that the IT domain intrinsically features more than two classification levels, and that it has a considerable amount of coded enterprise level data, FMML^x seems to be particularly suited to this end. Although the application of multilevel modeling in general, and FMML^x in particular, has its challenges and requires additional research, our experience indicates that it is well suited for enabling more powerful model-based analyses than the instruments currently offered.

References

Antunes G, Barateiro J, Caetano A, Borbinha J (2015) Analysis of federated enterprise architecture models. In: 23rd ECIS Conference, AISNet

Atkinson C, Kühne T (2001) The essence of multilevel metamodeling. In: Proceeedings of the 4th Int. Conf. on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. Springer, London, pp 19–33

Atkinson C, Kühne T (2008) Reducing accidental complexity in domain models. SoSyM 7(3):345–359

Atkinson C, Gerbig R, Fritzsche M (2015) A multi-level approach to modeling language extension in the enterprise systems domain. Inf Syst 54:289–307

Bock A, Frank U (2016) Multi-perspective enterprise modeling - conceptual foundation and implementation with ADOxx. In: Karagiannis D, Mayr H, Mylopoulos J (eds) Domain-specific conceptual modeling: concepts, methods and tools. Springer, Cham, pp 241–267. doi:10.1007/978-3-319-39417-611

Bodenstaff L, Wombacher A, Reichert M, Jaeger MC (2008) Monitoring dependencies for SLAs: the MoDe4SLA approach. In: IEEE International Conference on Services Computing, vol 1, pp 21–29

Bucher T, Fischer R, Kurpjuweit S, Winter R (2006) Analysis and application scenarios of enterprise architecture: An exploratory study. In: 2006 10th IEEE EDOCW, pp 28–28. doi:10.1109/EDOCW.2006.22

Carvalho VA, Almeida JPA (2016) Toward a well-founded theory for multi-level conceptual modeling. SoSyM (online first) pp 1–27. doi:10.1007/s10270-016-0538-9

Clark T, Willans J (2013) Software language engineering with XMF and XModeler. In: Mernik M (ed) Formal and practical aspects of domain-specific languages: recent developments. IGI Global, Hershey, pp 311–340

Clark T, Sammut P, Willans J (2008) Applied metamodelling: a foundation for language driven development. Ceteva, Sheffield

de Lara J, Guerra E, Cuadrado JS (2014) When and how to use multilevel modelling. ACM TOSEM 24(2):12:1–12:46. doi:10.1145/2685615

Florez H, Sánchez M, Villalobos J (2014) Extensible model-based approach for supporting automatic enterprise analysis. In: EDOC conf. 2014, IEEE Computer Society, Washington, EDOC '14, pp 32–41. doi:10.1109/EDOC.2014.15

Frank U (2011) The MEMO meta modeling language (MML) and language architecture, 2nd Edn. ICB-Research Report 43, University of Duisburg-Essen

Frank U (2014a) Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. SoSyM 13(3):941–962. doi:10.1007/s10270-012-0273-9

Frank U (2014b) Multilevel modeling – toward a new paradigm of conceptual modeling and information systems design. BISE 6(6):319–337

Frank U (2016) Designing models and systems to support it management: A case for multilevel modeling. In: Atkinson C, Grossmann G, Clark T (eds) Multi@models, ceur-ws.org, pp 3–24

Hanschke I (2010) IT landscape management. Springer, Heidelberg, pp 105–217. doi:10.1007/978-3-642-05034-34

Heise D (2013) Unternehmensmodell-basiertes IT-Kostenmanagement als Bestandteil eines integrativen IT-Controllings. Logos, Berlin

Johnson P, Lagerström R, Närman P, Simonsson M (2007) Enterprise architecture analysis with extended influence diagrams. ISF 9(2):163–180. doi:10.1007/s10796-007-9030-y

Jonkers H, Band I, Quartel D (2012) The ArchiSurance case study. White paper, The Open Group, Spring

Kaczmarek M, de Kinderen S (2016) A conceptualization of IT platform for the needs of enterprise IT landscape modeling. In: 18th Conference on Business Informatics (CBI), vol 01, pp 74–83. doi:10.1109/CBI.2016.17

Lago P, Malavolta I, Muccini H, Pelliccione P, Tang A (2015) The road ahead for architectural languages. IEEE Soft 32(1):98–105. doi:10.1109/MS.2014.28

Lankhorst M (2013) Enterprise architecture at work: modeling, communication and analysis, 3rd edn. Springer, Heidelberg

Malavolta I, Lago P, Muccini H, Pelliccione P, Tang A (2013) What industry needs from architectural languages: a survey. IEEE TSE 39(6):869–891

Neumayr B, Grün K, Schrefl M (2009) Multi-level domain modeling with m-objects and m-relationships. In: Proc. of the 6th Asia-Pacific Conference on Conceptual Modeling, Australian Computer Society, Darlinghurst, Australia, pp 107–116

Neumayr B, Schrefl M, Thalheim B (2011) Modeling techniques for multi-level abstraction. In: Kaschek R, Delcambre L (eds) The evolution of conceptual modeling. Springer, Berlin, pp 68–92. doi:10.1007/978-3-642-17505-34

Niemann KD (2005) Von der Unternehmensarchitektur zur IT-Governance: Bausteine für ein wirksames IT-Management. Vieweg+Teubner

Österle H, Becker J, Frank U, Hess T, Karagiannis D, Krcmar H, Loos P, Mertens P, Oberweis A, Sinz EJ (2011) Memorandum on design-oriented information systems research. EJIS 20:7–10

Quartel D, Steen MW, Lankhorst M (2010) IT portfolio valuation-using enterprise architecture and business requirements modeling. In: 14th IEEE International Conference on Enterprise Distributed Object Computing Conference (EDOC), pp 3–13

Sandkuhl K, Stirna J, Persson A, Wißotzki M (2014) Enterprise modeling: tackling business challenges with the 4EM method. Springer, Heidelberg

Sangiovanni-Vincentelli A, Martin G (2001) Platform-based design and software design methodology for embedded systems. IEEE Design Test Comput 6:23–33

Schmidt R, Möhring M (2016) Enterprise architecture analytics and decision support. In: El-Sheikh E, Zimmermann A, Jain LC (eds) Emerging trends in the evolution of service-oriented and enterprise architectures. Springer, Heidelberg, pp 201–218. doi:10.1007/978-3-319-40564-3_11

Schmidt R, Wissotzki M, Jugel D, Möhring M, Sandkuhl K, Zimmermann A (2014) Towards a framework for enterprise architecture analytics. In: Proc. of the 2014 IEEE 18th EDOCW, IEEE Computer Society, Washington, pp 266–275. doi:10.1109/EDOCW.2014.47

Sun R, Gregor S, Keating B (2015) Information technology platforms: conceptualisation and a review of emerging research in the IS discipline. In: The 26th ACIS, Adelaide

The Open Group (2013) ArchiMate 2.1 Specification: Open Group Standard. The Open Group Series, Van Haren, Zaltbommel