Designing Models and Systems to Support IT Management: A Case for Multilevel Modeling

Ulrich Frank

University of Duisburg-Essen, Germany
ulrich.frank@uni-due.de
https://www.wi-inf.uni-due.de/FGFrank/

Abstract. Refering to the domain of IT management, this paper demonstrates conceptual strengths and economic benefits of multilevel modeling. In the past, IT management was primarily focussed on technical aspects of IT infrastructures. In recent years, more and more organizations became aware of the pivotal relevance their IT infrastructures has for staying competitive. Therefore, IT managers are expected to not only provide IT services, but also to justify investments and costs with respect to the benefit the IT creates. On the one hand this responsibility demands for methods that allow for reducing the complexity of both, the IT infrastructure and the business. On the other hand, it is required accounting for interdependencies between these two areas. Specialized frameworks for IT management provide guidelines for specifying, managing and controlling IT services. Enterprise architectures or enterprise models support the alignment of IT and business by integrating models of the IT infrastructure and of enterprise software systems with models of the organizational action system. Against this background, specific requirements of modeling IT infrastructures will be looked at in more detail. It will then be demonstrated that a multilevel language architecture and a corresponding meta-modeling and programming facility represent a powerful foundation for the development of advanced tools for IT management.

Keywords: IT Management, Monitoring Tools, Models at Runtime, DSML, Multilevel Modeling

1 Introduction

Multilevel modeling offers a number of clear advantages over the traditional modeling paradigm. In general, it enables additional abstractions that foster reuse and flexibility. It also allows for using concepts that correspond more directly to the technical terminology in certain domains instead of overloading two-level languages, which is likely to jeopardize system integrity by causing accidental complexity [1]. Given these undisputed advantages, it is not surprising that multilevel modeling was embraced by some with great enthusiasm. A considerable number of approaches to multilevel modeling have been published so far, e.g., [2], [3], [4], [5][6]. However, it has not evolved into a major research topic so

far. In practice, it has received only, if at all, marginal attention. In part, this sobering situation may be explained with the well-known reluctance, if not resistance, in academia to turn toward a new paradigm [7]. Similarly, decision makers in practice are likely to adhere to mainstream solutions and standards in order to protect investments and for legitimation reasons as well. In addition to those obstacles, there are two other aspects that may have prevented multilevel modeling from becoming more popular. Most publications are focused on foundational aspects such as metamodels or language architectures. Only little attention has been paid to applying multilevel modeling to particular domains in order to analyse specific economic benefits that could arouse interest both in researchers and practitioners. Furthermore, the dissemination of multilevel modeling may be hindered by the lack of corresponding programming languages, which does not only imply the loss of semantics when models are mapped to code, but also creates an additional challenge for the synchronization of models and code.

This paper addresses both obstacles. First, it will be shown that the domain of IT management is in need of conceptual models that support the analysis of a increasingly complex subject and that foster sense-making and communication between different stakeholders. While IT management faces some particular challenges, its need for models is not idiosyncratic, but applies to any developed professional domain. Certain aspects of IT management are especially suited to illustrate modeling problems that cannot be satisfactorily solved within the traditional paradigm. Against this background, it will be demonstrated that a family of multilevel DSMLs is suited to offer a solution that promotes model integrity, modeling productivity and economies of scale. For this purpose a meta-modeling language is presented that allows for an unbounded number of meta levels and for the specification of so called intrinsic features. Second, an extension of a meta-modeling and -programming environment will be presented that is based on a recursive and reflective language architecture. It allows for the common representation of models and code and, thus, enables versatile decision support tools for IT management. They are suited to not only foster user empowerment by combining (meta) modeling features with customizable monitoring capabilities, but also to better integrate IT management with other management functions to make it more effective and more efficient.

2 IT Management: The Need for Models

In the early days of data processing, IT departments in many companies had a predominant technical focus. Software development and maintenance as well as dealing with the intricacies of hardware components were top priorities. As a consequence, the typical employee of an IT department had a technical background, often enough without an advanced qualification in software engineering. Over the years, many IT infrastructures developed into a “zoo” of heterogeneous applications. While the “horrors of the past” and the ever increasing complexity of IT infrastructures kept IT departments busy, IT did not only turn into the backbone of business operations, but became more and more a potential

enabler of future business models and, hence, a matter of survival. At the same time, IT budgests were still growing, while IT projects often did not deliver and the benefit of investments was hard to tell. Against this background, IT departments underwent a substantial change. Most companies abandoned inhouse development, which required employees who were able to deal with vendors and external service providers. Line managers who realized the potential of IT to improve their operations, developed business cases that required the adaption of IT systems. The growing need for collaboration between business people and IT professionals, the ever growing relevance of IT for the business and the complexity of IT infrastructures led to the emergence of a specialized management function. IT management should not only provide the required support to the business. It should also identify new opportunities of using technology to improve the business. Furthermore, it is supposed to control IT costs and organize IT departments to become more efficient. At the same time, IT management should overcome tradtional cultural barriers between IT experts and business professionals. The transition to IT management requires new organizational structures, new skills, and methods that support the introduction of appropriate structures and processes. To address the wide range of IT management tasks, methods and tools are needed that account for both, the peculiarities of IT infrastructures and the needs of businesses that will often have to adapt quickly to a changing environment. With regard to the pivotal economic relevance of IT management, it is not suprising that various vendors and initiatives responded do those needs. In the following we shall look at different approaches that are aimed at supporting IT management. This is to serve two purposes. On the one hand, it will underline the pivotal relevance of concepts and conceptual models. On the other hand, it will help to identify requirements that are not sufficiently satisfied so far.

2.1 Focus on Databases

The vast amount of software artefacts and hardware systems that form an IT infrastructure creates the need for tools that support inventory management, version management, configuration management, software installation, etc. A database that represents all IT resources together with relevant inderdependencies, often referred to as "configuration management database (CMDB)", is an obvious choice to provide a common foundation for such tools. However, designing a database schema for that purpose requires skills and resources that go beyond the capabilities of many organizations. Furthermore, individual solutions would impede the use of tools that rely on a database. Therefore, it seems a reasonable approach to define a common schema for CMDBs and to standardize it in order to foster protection of investment. The "Common Information Model (CIM)", promoted by an industry consortium of large hardware and software vendors, the "Distributed Management Task Force (DMTF)" is the most relevant attempt of this kind. It defines a schema that is documented as a set of UML class diagrams which are structured into three layers. The "core model" includes basic classes that are referred to in the "common model" which includes models

for representing application systems, hardware devices, networks, etc. Finally, "extension schemas" are included to define possible extensions to the common models. Figure 1 shows an excerpt of the CIM that is focused on application systems.

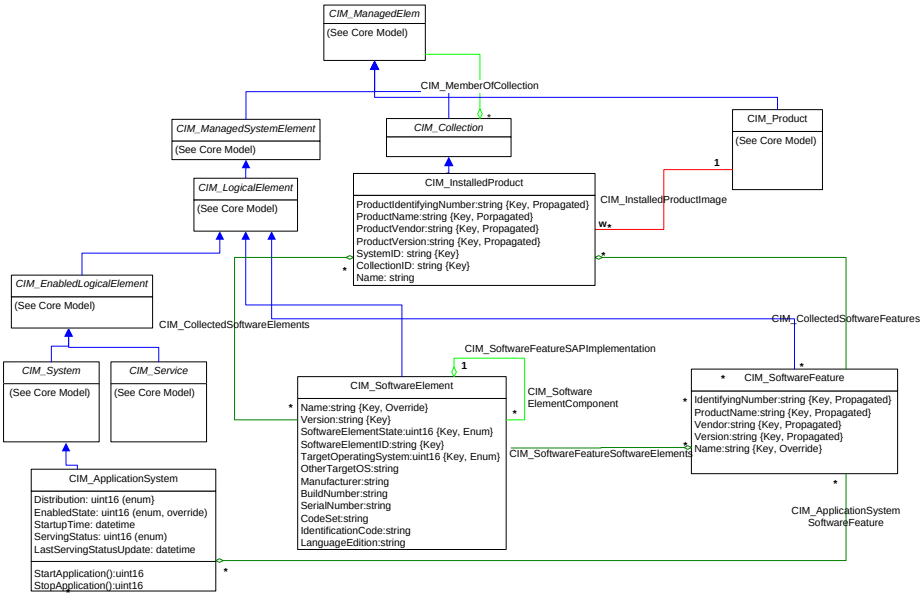

Fig. 1. Excerpt of CIM

To enable adaptations of schemas that go beyond predefined extensions, the CIM is supplemented with a metamodel (see figure 2) which can be used to define customized schemas. If tools are not only constructed to handle the CIM, but also instances of the metamodel, user may define new schemas and still use a standard tool. However, apparently the metamodel is generic as it includes general purpose concepts only. Therefore, it is not possible to build tools that can make sensible use of any possible instance of the metamodel. The DMTF seems to be aware of this limitation, because the range of feasible instances is restricted to "*new* conformant models" (http://www.dmtf.org/standards/cim).

A reliable industry standard that defines a schema for configuration management databases promises effective support for managing IT infrastructures. It also promotes protection of investment and vendor independence. At the same time, it may create a problematic lock-in effect.

2.2 Management Frameworks

The introduction of IT management requires the definition of responsibilities, services, processes and controls. Two frameworks for establishing IT manage-

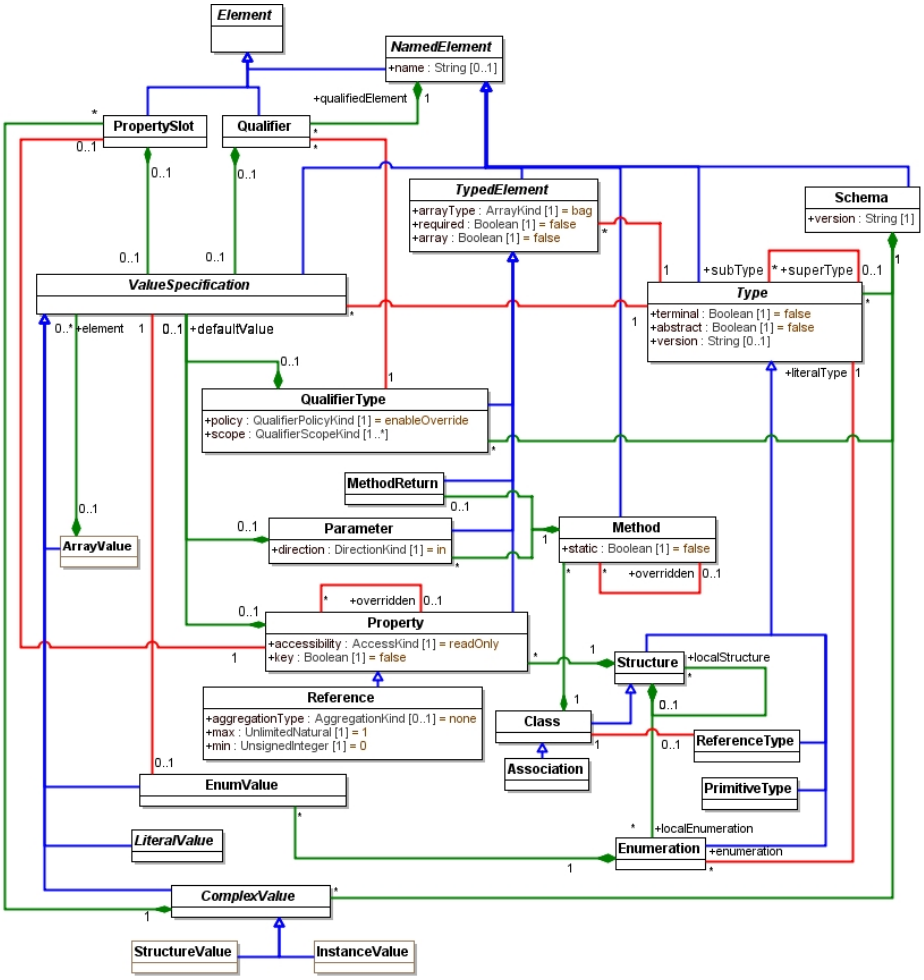

Fig. 2. Metamodel of the CIM ([8], p. 30

ment have achieved a remarkable dissemination. They both emerged from practice. ITIL ("Information Technology Infrastructure Library") orginates from a project launched by the British government. It is supposed to describe "the organisational structure and skill requirements of an information technology organisation and a set of standard operational management procedures and practices to allow the organisation to manage an IT operation and associated infrastructure." (http://www.itlibrary.org/). To this end, ITIL proposes a managerial perspective on IT that rests on two main pillars. On the one hand, the IT infrastructure is encapsulated toward the business with services. The specification of services is supported by comprehensive guidelines. On the other hand, the design of an IT organisation is supported by the definition of core functions such as

incident management, configuration management, change management, etc. For each of these functions, subjects, roles, processes and checklists are defined that provide hand-on guidelines for establishing IT management practices. ITIL offers various certificates that enable organizations and employees to demonstrate their IT management maturity.

"Control Objectives for Information and Related Technology (COBIT)" is a framework that originates in auditing. Similar to ITIL it is supposed to support companies with establishing a professional IT management. However, it has a different focus. Its main emphasis is on the introduction and continuous improvement of IT governance. For this purpose, COBIT provides a framework of responsibilities, rules and high level activities that aim among other things at improving the alignment of IT and business, at improving the qualitiy of IT services and at more reliable predictions of IT costs. The framework also includes the definition of corresponding metrics that serve to measure and control IT management practices. Like ITIL, COBIT offers training courses and certificates. There is one further aspect that is shared by both frameworks. Even though they do not include any conceptual model that was designed with an explicit modeling language, the backbone of both framework consists of a conceptual foundation in the form of technical languages that allow to structure the subject and the responsibilities of IT management in a purposeful way. Figure 3 shows a reconstruction of concepts defined in ITIL in comparison to those that are used in COBIT. The colour green marks concepts that are shared by both approaches, that are, however, clearly more comprehensively defined in ITIL. Blue indicates that corresponding concepts are defined in similar detail in both frameworks, while red marks those concepts that are not accounted for in COBIT. Yellow marks common concepts that are described in more detail in COBIT.

The lack of precise metamodels may be related to the business models behind ITIL and COBIT. Both promote training and consulting to help users develop appropriate interpretations that fit their needs. At the same time, both frameworks were not designed to serve as a foundation for building software tools.

2.3 Enterprise Architecture Management

IT management is more and more regarded as a pivotal management function that does not only manage IT resources, but supports the business more directly and contributes to the strategic development of a firm. As a consequence, IT managers are supposed to have a clear understanding of how IT should support the business and how it may contribute making a company more competitive. At the same time, they should be able to efficiently communicate with users, line managers and top management, similar to consultants. The idea of an enterprise architecture, which was introduced by Zachman, an IBM sales representative who aimed at improving communication with customers, intends to provide a representation of the enterprise and its IT infrastructure that reflects the basic building blocks and relevant dependencies. An enterprise architecture mainly targets management audiences. For this reason, it stresses a high level

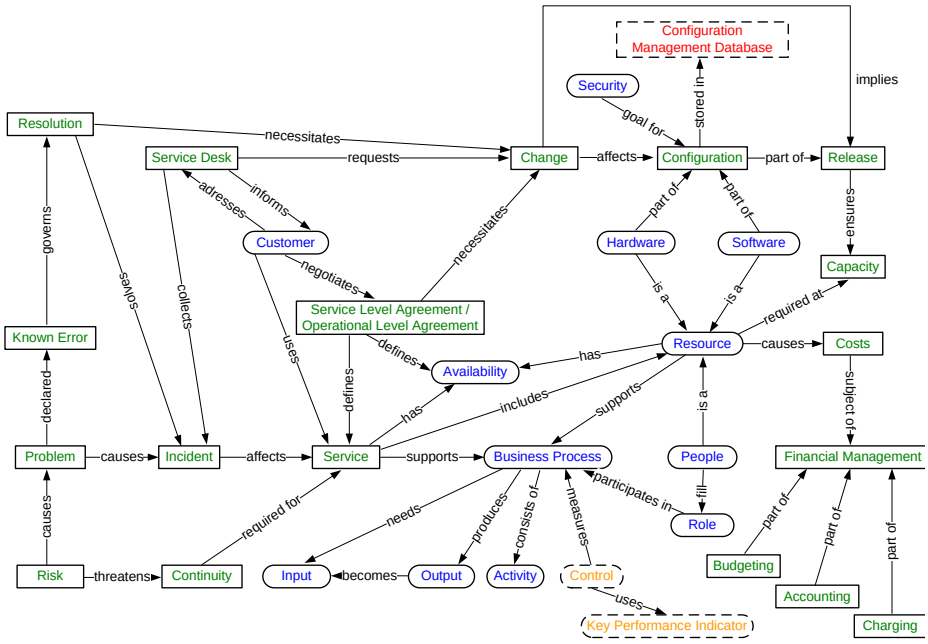

Fig. 3. Conceptual Reconstruction of ITIL and COBIT (copied from [9], p. 135)

of abstraction ("ballpark view") and does not account for specific details. It is supposed to represent essential aspects of the business, such as goals and business processes, and to integrate them with those aspects of an IT infrastructure that are relevant for managerial decision making. It is regarded as a pivotal tool for promoting IT business alignment, for making IT more efficient, and for supporting strategic (IT) management [10]. To emphasize the importance of developing, documenting, communicating and enforcing an enterprise architecture, it has been proposed to introduce a dedicated management function. Enterprise architecture management is not intended to replace IT management, but is rather seen as one of the core functions of IT management. While enterprise architectures are still often created with generic drawing tools only, there are various attempts to provide a more formal foundation for designing enterprise architectures (e.g., [11], [12]). One language to model enterprise architectures has gained remarkable attention, probably also because it has become part of TOGAF (http://www.opengroup.org/subjectareas/enterprise), a framework for enterprise architecture management that is characterized by similar claims as ITIL and COBIT. Archimate [13] includes a rather generic metamodel and various so called viewpoints that consist of concepts to model certain views of the enterprise. It is unclear whether the viewpoints are intended as metamodels. In any case, the corresponding models, which are presented in the same notation as the one used for the models that represent particular views, remain on a high level of abstraction. For example, they do not include attributes or multiplicites.

Figure 4 illustrates how modeling concepts are defined and used. The language is designed for a high degree of adaptability. For example, every concept allows for adding any kind of "property", which does not, however, mean to refine its semantics. Instead, the extension happens on the instance level and is specified as a tuple of a name and a corresponding value. Apart from the unusual conceptualization, the downside of this kind of flexibility is obvious: Archimate allows for creating models that are wrong in the sense that they are counter-factual, e.g. an ERP system could be modeled as being part of a DBMS, or that do not make any sense, because properties were added that are meaningless.

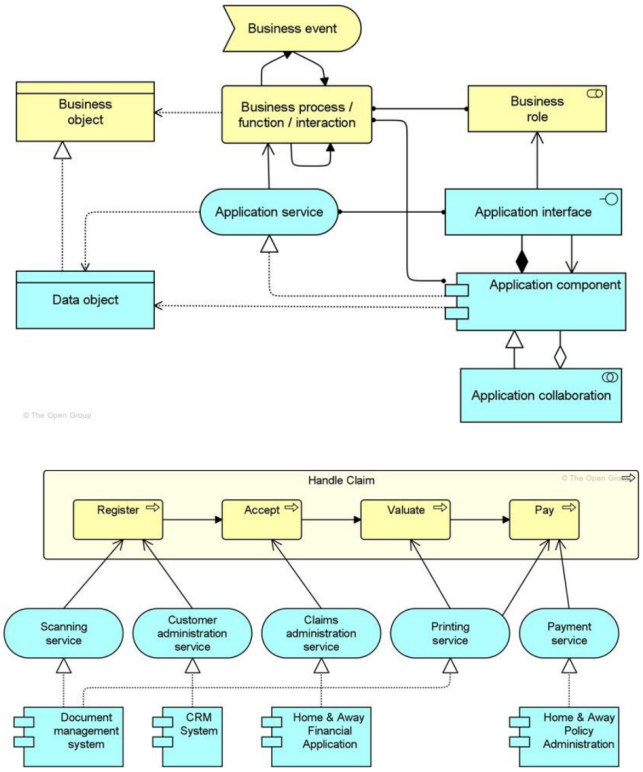

Fig. 4. Illustration Archimate Viewpoint and Corresponding Model

2.4 Intermediate Conclusion

The different approaches to support IT management share two common properties. First, they emphasize a pragmatic approach, either by using mainstream technology such as CIM, or by preferring sketchy definitions over elaborate conceptualizations such as ITIL, COBIT, and to a lesser extent, Archimate. Second,

they allow for adaptations and, in that respect, put more emphasis on flexibility than on integrity. While Archimate claims to be a modeling language, it hardly qualifies as a DSML. The viewpoint definitions are not metamodels that allow for specifying types. Instead, viewpoints and the models that correspond to them are characterized by a subtle overloading of concepts. The lack of sufficiently clear semantics makes it virtually impossible to use Archimate models for generating code. Although there is some overlapping between the different approaches, integrating them into one comprehensive, multi-perspective approach would be benefitial. However, the lack of clear conceptual foundations makes it difficult to accomplish such an integration.

3 Design and Use of DSMLs for IT Management: Prospects and Obstactles

Our work on DSMLs for modeling IT infrastructures was at first motivated by our long-standing research on enterprise modeling. In the early days of enterprise modeling, it was assumed that companies need support for software development. Accordingly, there was emphasis on integrating software design languages, such as object-oriented modeling languages. Later, it became apparent that software development was of less concern for most companies than managing the often huge IT infrastructures that have emerged over the years. Therefore, we decided to add a language for modeling IT infrastructures [9] to the range of already existing DSMLs. Second, our work was inspired by a project with the global market leader for data center management tools. The company board had realized that focusing on tools to support the technical management of IT only would not be sufficient in the long run. Therefore, they were looking for solutions that put more emphasis on IT business alignment and gave users more flexibility in adapting tools to their particular needs.

3.1 Vision: Integrated Modeling, Monitoring and Decision Support for IT Management

It took some time to convince the seasoned software developers in the company that a DSML that is integrated with other DSMLs for enterprise modeling could serve as a powerful foundation for future IT management tools. However, a modeling tool alone would not be sufficient for that purpose. Apart from conceptual support for analyzing the IT and the business, IT managers also need to know the state and the state history of particular objects, for example the availability of a particular server or the maintenance costs of an application system. They also need to be informed about certain events like attempts from external intruders to get access to sensitive data. In addition, decision making requires various kinds of statistical analysis on sets of data. Most companies appreciate general frameworks for IT management, because they do not want to start from scratch, but simulaeously, they do not want to be restricted too much. Against this background a vision of a future tool was developed that was based

on three main components. First, an existing method and toolset for creating and managing enterprise models should be used to address the need for integrating representations of the IT with those of the organizational action system. Second, an existing DSML for modeling IT infrastructures that was already part of the enterprise modeling method should be refined to better satisfy the needs of prospective customers [14]. Third, a versatile dashboard system should support managers with customizable representations of aggregate data and with notifications of relevant events. Fourth, and most appealing, the enterprise modeling environment should be integrated with the dashboard system [15]. An IT manager could study a certain model of the IT infrastructure, navigate to associated representations of organizational goals or of business processes, and, if needed, drill down to data representing particular objects.

3.2 Obstacles and Challenges

The enthusiasm we had developed for the vision of an advanced IT management tool was soon contrasted by the sobering insights that an implementation of the vision was confronted with serious obstacles. They include general problems with the design of DSMLs that became especially apparent with the design of a language for modeling IT infrastructures, and more specific problems that are related to peculiarities of the subject. In general, the design of a DSML is confronted with the decision whether a certain concept should become part of the language or rather be defined with the language. Take, for example, the following terms: *Software, Operating System, ERP-System, DBMS, RDBMS, Middleware, Printer, Laser-Printer*. How could one decide whether to take the more generic or the more specific term—or both? Criteria to support this decision have been proposed in [16]. For being incorporated into a language, a concept's semantics should be invariant throughout the range of intended applications. Furthermore, its instances should be perceived as types intuitively. Unfortunately, these criteria are not sufficient to enable clear decisions. The popular advice that the decision depends on the purpose of a language does not help much either, because a clear definition of a particular purpose will often be hardly possible and, furthermore, the goals to be addressed with a DSML may be in conflict. An essential conflict with the design of DSMLs results from the fact that it will usually be reasonable to design a language for a wide range of (re-) use, because that will promote economies of scale and, hence, the economic benefit of a language. However, at the same time, making a language more specific promotes the productivity of its use in those cases where it fits. Figure 5 illustrates this conflict that requires a trade-off which will be unsatisfactory in many cases.

A further, related problem is suited to cause frustration, too. The analysis of domain concepts often results in a hiearchy of concepts where more general, but still domain-related terms are refined step by step to more specific terms. Figure 6 shows a typical example of this problem that seems to be rather the rule than an exception. Obviously, such a hierarchy leads to the question whether the refinement represents an instantiation relationship or rather a specialization relationship. Looking at the hierarchy of terms on the left only may inspire an

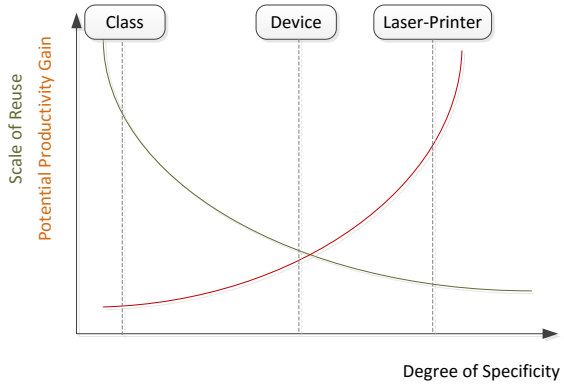

Fig. 5. Illustration of Essential Design Conflict

ontological discussion with an uncertain outcome. One can also take a more pragmatic approach by adding properties such as attributes and operations that are characteristic for the pictured terms. The hierarchy on the right represents a corresponding view. Its analysis leads to a sobering result. Even though it is an iron law of conceptual modeling that there is a clear dichotomy between instantiation and specialization, the example indicates that it may make sense to combine both. We know that every peripheral device has a sales price. This is the case for any printer, too. Therefore, the attribute salesPrice should be inherited to Printer. However, the operation numberOfModels is to be used by Printer, which would suggest to instantiate it from PeripheralDevice. Similarily, the attribute resolution in Printer is instantiated in classes that represent particular models, while the attribute serialNumber should be inherited from Printer to classes that represent a model. One could argue that it is not appropriate to assign an attribute like salesPrice to PeripheralDevice, because it is instantiated only with specific models. However, that would result in conceptual redundancy, because it would imply to introduce a corresponding attribute for each kind of peripheral device again and again. There are some obvious lessons to be learnt from examples like that. First, there are cases where a strict dichotomy of instantiation and specialization (or rather inheritance) is dissatisfactory. Second, instantiation of attributes may sometimes be deferred to a lower level. Note that this may be the case for operations, too. An operation like numberOfDevices could be specified with PeripheralDevice already, which would replace redundant operations like numberOfPrinters in the example. Third, classes as well as metaclasses may have a state and may execute operations. Therefore, they should be regarded as objects. Unfortunately, all of these lessons are in clear contrast to the dominant modeling paradigm, which means that they cannot be expressed accordingly in a traditional one level (M_1) object model.

A further challenge results from the idea to integrate a modeling environment with dashboard software. A dashboard is supposed to present data related to concepts specified in models. Integration implies among other things that

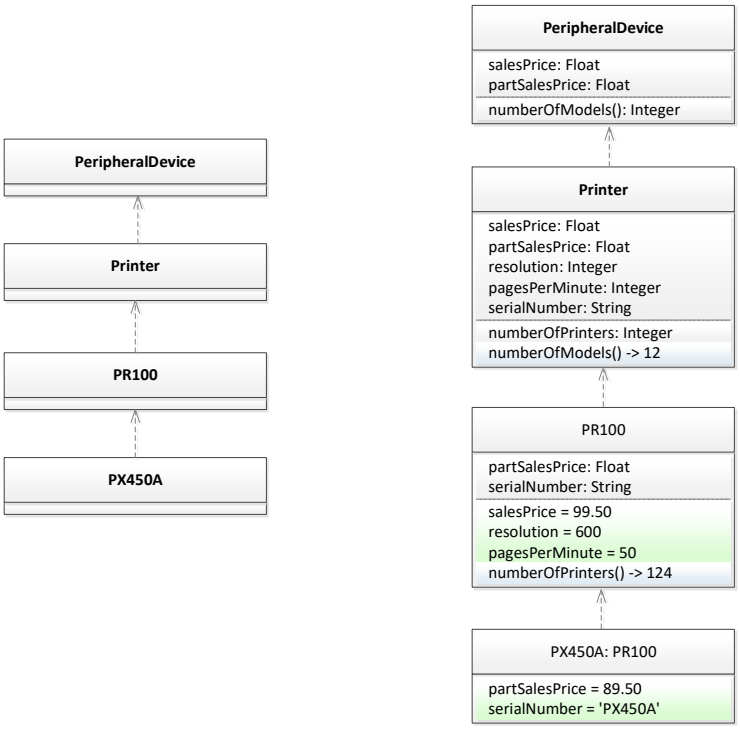

Fig. 6. Instantiation or Specialization?

it should be possible to navigate from the dashsboard to a model et vice versa. However, as shown in fig. 7, there is a serious obstacle that prevents a satisfactory integration. Classes that represent a model (or a metamodel respectively) have to be represented as objects on M_0 in a modeling environment, because mainstream object-oriented programming languages do not allow to treat classes as objects, nor do they allow for metaclasses. Hence, "classes" in a model editor cannot be instantiated any further. Therefore, the approach of choice is to generate code from models. Unfortunately that approach is hardly satisfactory. It would require synchronizing models and code, which is a notorious problem that does not allow for a satisfactory solution. It would also require a sophisticated middleware system to establish references between the modeling environment and the dashboard system that enable navigation at runtime. Furthermore, generating code from a model that was specified with a DSML to a general-purpose programming language can be a demanding task.

4 Outline of a Solution

To cope with the obstacles outlined above, we supplemented our MOF-like language architecture ([17], [18]) with various "workarounds", but that did not re-

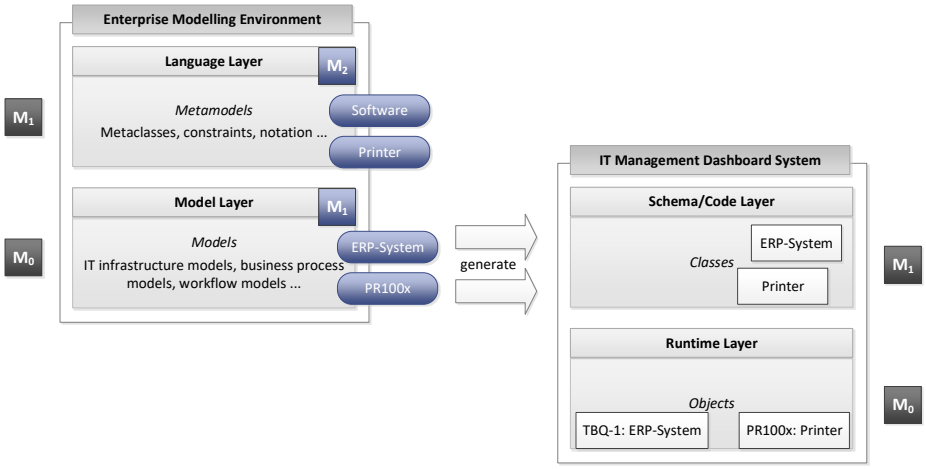

Fig. 7. Obstacles of Integrating Models and Code

sult in a satisfactory solution. Instead, it became more and more apparent that the challenges we experienced cannot be overcome without leaving the traditional paradigm. While multilevel languages seemed to be much better suited as a foundation for a convincing solution, implementing multilevel models for being used at runtime was not possible without painful trade-offs. The solution that is presented below became possible only through the availability of a multilevel language engineering facility.

4.1 Foundation: XMF

The idea of recursive language architectures has probably become popular by the "golden braid" metaphor Hofstadter used for his sophisticated praise of recursion [19]. It is based on the idea that a class can be regarded as an object, too, which is instantiated from a meta class, which in turn can be seen as an object instantiated from a higher level class ("everything is an object"). Only very few programming languages are based on a "golden braid" architecture (e.g., object-oriented extensions of Lisp, Smalltalk, Ruby). Among these languages, Smalltalk is especially appealing. It treats classes as objects and is available within powerful development environments. Unfortunately, Smalltalk is not suited for our purpose, since it does not allow for defining metaclasses above M_2. Furthermore, it does not feature metaclasses as classes of many classes, since each class is assigned exactly one default metaclass only. XMF ("executable Metamodeling Facility") is not limited by this constraint ([21], [20]). It is a language execution engine that features a recursive metamodel, called XCore ([20], p. 43). Every language that is specified in XCore can be executed by XMF. XMF allows accessing and modifying its own specification and its runtime system. Hence, there is no clear distinction between the language and a respective meta language: XMF is

reflective. Furthermore, it includes tools for building compilers for further languages. Therefore, XMF is a meta programming facility that allows to execute code written in different languages in the same runtime system. Furthermore, XMF allows for modifying XCore to satisfy specific requirements of designing and using DSMLs.

The golden braid architecture in general, XCore in particular are based on concepts that may be perceived as confusing, since they seem to violate well known principles of meta-modeling. XCore makes use of a circular relationship: The central metaclass Class, which is associated with a meta attribute, a meta operation and a meta association is an instance of itself. At the same time, class inherits from Object. Hence, every class is an object and can be executed. Object itself is instantiated from Class. Furthermore, every instance of Class can inherit from every other instance of Class. For a more detailed description see [20], especially p. 23 ff. The lean recursive structure of XCore allows for creating an unbounded number of classification levels but not without effort. As a default, every class that is instantiated from Class is located on M_1. If a metaclass on a higher level of classification is required, one would specialize a class that was instantiated from Class from Class at the same time. That would result in lifting it up to M_2, because it would inherit the instantiation capability of the original metaclass. Instantiating the new class and having the resulting instance inherit from the original metaclass would result in lifting the instance up to M_2 and, as a consequence, its classification level, which was previously M_2 up to M_3. Since the lifting procedure can be repeated indefinitely, language hierarchies with any number of classification levels can be realized.

4.2 The $FMML^x$

XMF provides all the flexibility that is required for developing a satisfactory solution for the implementation of the outlined IT management support tool. In particular, it allows for a common representation of models and code. Classes, no matter on what level, are objects at the same time. They can be represented in the tool on the intended classification level. They can also be instantiated and executed within the Xmodeler. However, XCore lacks two features that are useful for modeling and implementing multilevel DSMLs. First, XCore does not provide direct support for deferred instantiation. Second, from a conceptual viewpoint it is important to know the classification level of a class, because it makes a clear difference whether a class is on level n or on level m with $n \neq m$. However, classes in XMF are not explicitly assigned to a classification level upon instantiation. Furthermore, the level of a class may be contingent in the sense that the level may change during the lifetime of the class. The level of a class at a particular point in time can be determined dynamically through step-by-step instantiation only. It is also possible that a class is level-agnostic, which means that it may be interpreted as being on different levels at the same time. This is in particular the case for Class the instances of which may reside on any level above M_0 which implies that Class may virtually reside on many levels above M_1 simultaneously. While being level-agnostic is a prerequisite for the positioning of

classes on multiple classification levels, it is hardly acceptable from a conceptual point of view, because it would remain unclear what kind of concept the class represents in the end. Conceptually, contingent classification is questionable, too, because the classification level is a pivotal aspect of a concept's semantic.

Against this background, we decided to specify a meta-modeling language, the $FMML^x$ ("Flexibile and Executable Meta-Modeling Language") that, like XCore, enables an unbounded number of classification levels. However, for every model created with the $FMML^x$, each class is assigned a level. In addition to that, the $FMML^x$ supports intrinsic features. An intrinsic feature is an attribute, an association, or an operation that is defined on a level n, but is instantiated only on a level m, with m ¡ n - 1. The $FMML^x$ was created by extending XCore and by introducing a specific concrete syntax for representing multilevel models. Figure 8 illustrates the use of intrinsic features and the concrete syntax of the $FMML^x$. The level of a class is indicated by the colour of the background a class name is printed on. To indicate the class, a class is instantiated from, the name of the metaclass is printed above the classname. Intrinsic features are marked by the instantiation level that is printed in white on a black rectangle. Intrinsic associations may have two, possibly different instantiation levels assigned to them. The model in figure 10 includes a corresponding example: the class HardwareComponent on M_4 is associated with the class Location on M_2. Intrinsic features correspond widely to "deep instantiation" and "potency" ([1],[22]), except they are applicable not only to attributes, but also to operations and associations.

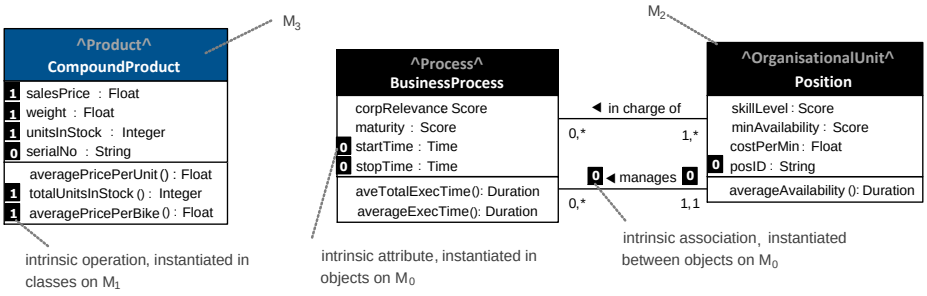

Fig. 8. $FMML^x$: Illustration of Concrete Syntax

As illustrated in figure 9, the $FMML^x$ is a monotonic extension of XCore. On the one hand, that means that existing models of XCore will not be affected by the extension. On the other hand, it allows to preserve the flexibility provided by contingent levels for those cases where it is needed. The extension comprises two parts. First, the meta-attributes isInstrinsic and instLevel are added to Attribute, CompiledOperation, and End. The meta-attributes allow for defining attributes, operations and associations as intrinsic. Second, a metaclass, CompiledOperation, is defined that allows to instantiate level-aware classes. For

this purpose, the instantiation method new() had to be overriden. It could not, however, be overriden in Class without side-effects on previous models of XCore. Therefore, an intermediate class, MetaAdaptor, was introduced. It includes the attribute level, which enables to assign a level to every class. It is also used to re-implement new(). A class can be instantiated from MetaClass on any level. Hence, the level of metaclass itself is contingent. It is, however, possible to define a level for Class, through the attribute level specified in MetaAdaptor, that is supposed to be invariant for a certain model.

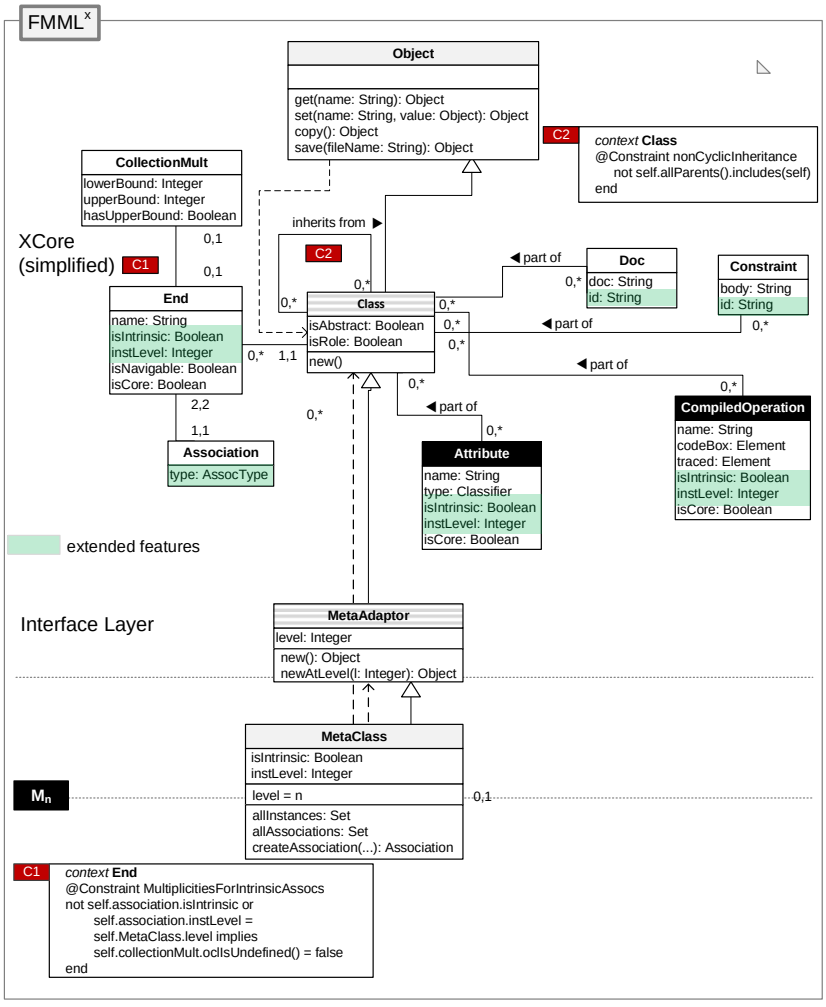

Fig. 9. $FMML^x$: Metamodel

A model editor for the $FMML^x$ was implemented within the Xmodeler. It allows to create models and modeling languages simultaneously. Every class that is created above M_2 is a language concept that is immediately added to the palette and can be subsequently used to create new instances (see figure 10).

4.3 A Multilevel IT Modeling Language

To demonstrate the potential of a multilevel approach to solve the problems discussed in 3.2, an existing language for modeling IT infrastructures (ITML) [9] was reconstructed in part with the $FMML^x$. The reconstruction resulted in a multilevel model the concepts of which qualify as DSMLs on different levels, where a language on one level refines, and, hence, reuses, concepts defined with a language on a higher level. Only on M_1 one would not speak of language concepts in the sense that they do not allow for further instantiations. In addition to enabling multilevel models, it is also possible to extend a model with objects on M_0. In other words, an application system can be integrated with its models and (meta) meta models at runtime. The highest level of classification used for the current multilevel version of the ITML is M_4. It applies, for example to the class HardwareComponent that is shown in figure 10.

Each class in the model shown in figure 10 is an object the operations of which can be executed. The enlarged representation of three selected classes in figure 11 illustrates this feature. The class Printer on M_2 executes two operations which are defined with its metaclass PeripheralDevice. Whenever the number of printer models changes, the operation is anewly executed and the resulting value is presented in the diagram. The Sister200 on M_1 stores the values that specify a particular printer model such as the resolution or the speed. It also executes operations that collect data from its instances, such as costs or pages printed in a certain period. Finally, particular printers are represented on M_0. The state of each object, no matter on what level of classification, can be changed interactively. Hence, a diagram turns into a multilevel representation. Furthermore, the specification of classes can be modified, too. If attributes or operations are added, they are, as a default, available not only with newly created instances, but also with already existing ones. However, other policies may be defined. Deleting attributes will, as a default, still allow access to existing slot values. This is similar for deleting operations which still allows executing those operations with existing objects, but of course not with those that were instantiated after the removal of an operation. Again, different policies for deleting properties may be defined. Adding and deleting intrinsic features requires more complex operations.

Apparently, a multilevel object model is at the same time a multilevel software system, which could be directly used to store and calculate all data required for a dashboard system. Users of a dashboard system will probably prefer other representations than diagrams. This can be accounted for by regarding the model as a model in a MVC architecture, which is in fact supported by the Xmodeler. Any kind of visualisation could be defined with one or more corresponding views

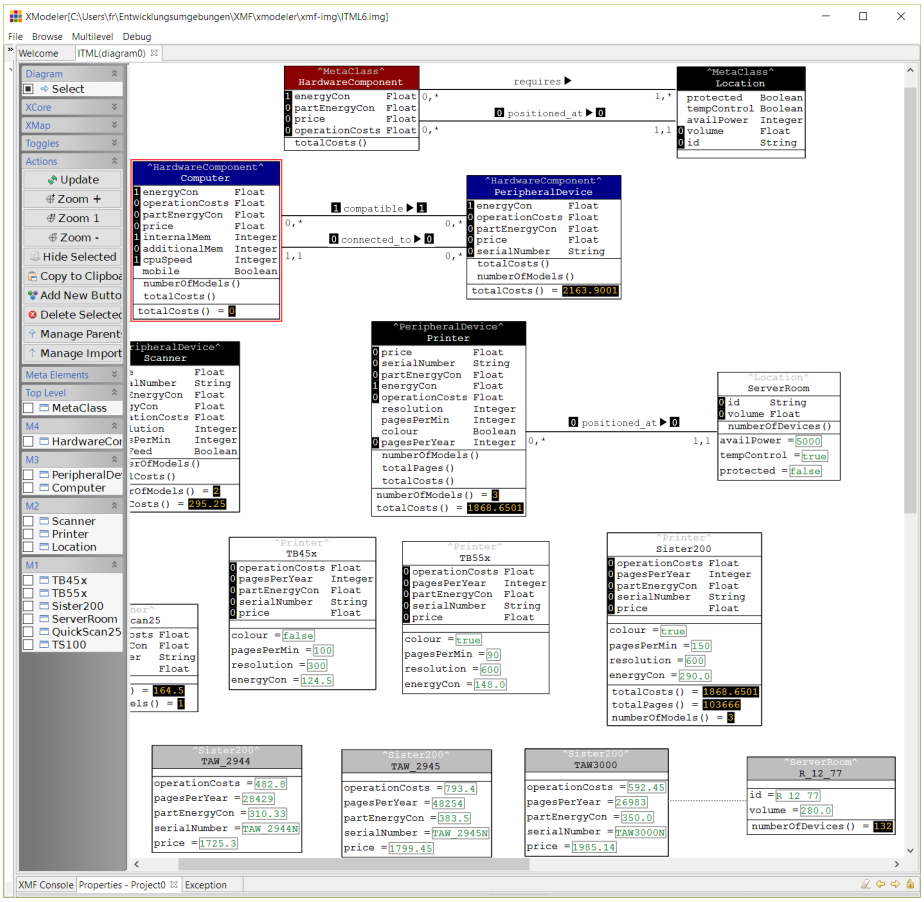

Fig. 10. Screenshot of Xmodeler with Excerpt of Multilevel ITML

that would support user-specific interfaces. Figure 12 shows a GUI of a dashboard system that represents a view of the multilevel model in figure 10.

5 Conclusions and Future Work

IT management is an example out of many domains that are characterized by the need for an elaborate technical terminology in order to support advanced decision making. In addition to that, it also requires the management of particular resources. While the use of DSMLs is promising to model domains of this kind, traditional language architectures with one classification level only are confronted with serious problems, both with respect to the design of proper conceptual models and the realization of corresponding software systems. In this paper it was demonstrated that a multilevel language architecture that integrates

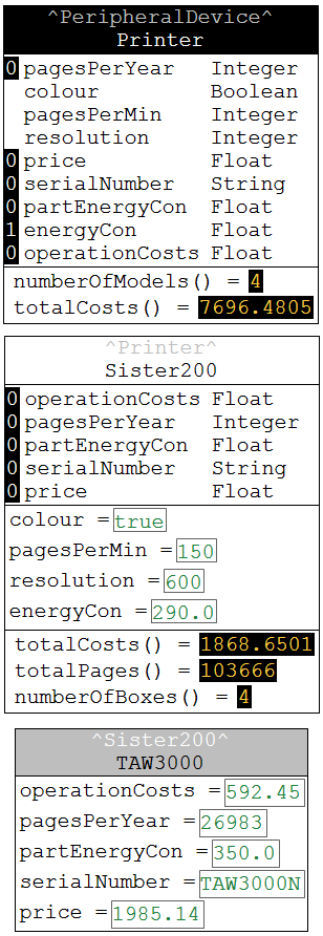

Fig. 11. Enlarged Excerpt of Multilevel Model

a (meta-) modeling environment with a (meta-) programming facility is suited to effectively address those problems. It does not only enable more expressive models that promote reuse and flexibility, it also provides the foundation of a new class of application systems that are integrated with conceptual models of themselves at runtime. Such self-referential systems are suited to empower users, because they do not only provide users on demand with information about the conceptual foundation of the system they work with, but may also enable users to modify the system by editing those parts of the models they are authorized to change.

To further promote the field of multilevel modeling and multilevel software construction, it seems useful to develop languages, models and applications for further domains, and compare the results to traditional approaches (for a further

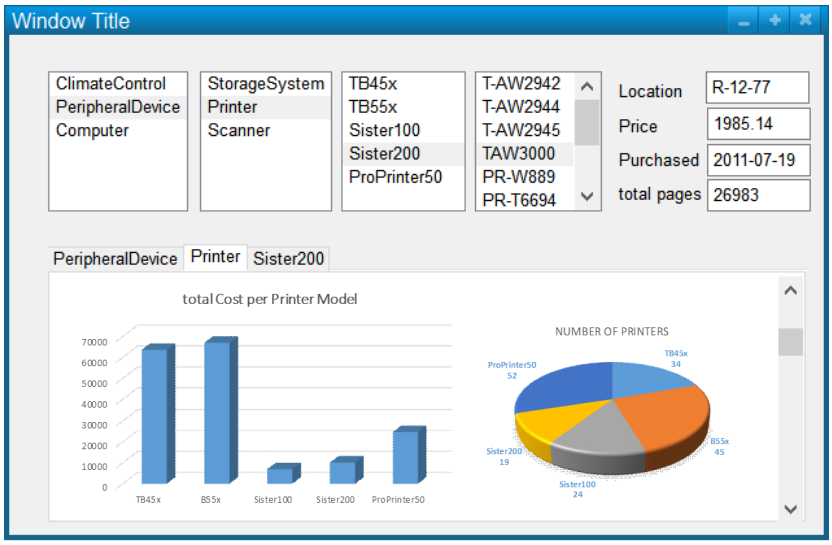

Fig. 12. Illustration of Dashboard

analysis of this kind see [23]). That does not only contribute to demonstrating the practical benefits of multilevel approaches, it also leads to new requirements for foundational meta-languages and language engineering facilities. The integration of an application system with multiple meta levels increases its flexibility substantially. At the same time, it adds complexity and raises specific challenges to integrity. Therefore, more research is needed on specifying and ensuring integrity constraints in multilevel systems. The design of multilevel domain models is in part still unknown territory. So far, there is only little methodical support ([24] and, in part, [25]). Therefore, future research needs to aim at the further development of specific design and analysis methods. So far, research on multilevel modeling was mainly restricted to static abstractions. A multilevel approach to process modeling may be suited to reduce the remarkable lack of abstraction and reuse in current process modeling languages [26].

References

1. Colin Atkinson and Thomas Kühne. Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359, 2008.
2. Colin Atkinson and Thomas Kühne. The essense of multilevel metamodeling. In Martin Gorgolla and Cris Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, volume 2185 of *Lecture Notes in Computer Science*, pages 19–33. Springer, Berlin and London, New York, 2001.
3. Manfred A. Jeusfeld. Metamodeling and method engineering with conceptbase. In Manfred A. Jeusfeld, Matthias Jarke, and John Mylopoulos, editors, *Metamodeling for Method Engineering*, pages 89–168. MIT Press, Cambridge, 2009.

4. Thomas Kühne and Daniel Schreiber. Can programming be liberated from the two-level style: multi-level programming with deepjava. In Richard P. Gabriel, David F. Bacon, Cristina Videira Lopes, and Guy L. Steele, editors, *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications (OOPSLA '07)*, volume 42,10 of *ACM SIGPLAN notices*, pages 229–244, New York, 2007. ACM Press.
5. Colin Atkinson, Matthias Gutheil, and Bastian Kennel. A flexible infrastructure for multilevel language engineering. *IEEE Trans. Software Eng.*, 35(6):742–755, 2009.
6. Bernd Neumayr, Katharina Grün, and Michael Schrefl. Multi-level domain modeling with m-objects and m-relationships. In Markus Kirchberg and Sebastian Link, editors, *Conceptual Modelling 2009*, pages 107–116. Australian Computer Society, 2009.
7. Thomas S. Kuhn. *The Structure of Scientific Revolutions*, volume 159 of *Phoenix books*. Univ. of Chicago Press u.a, Chicago, 1964.
8. DMTF. Common information model (cim) metamodel.
9. Lutz Kirchner. *Eine Methode zur Unterstützung des IT-Managements im Rahmen der Unternehmensmodellierung*. Logos, Berlin, 2008.
10. Frederik Ahlemann. *Strategic enterprise architecture management: Challenges best practices and future developments*. Management for professionals. Springer, Berlin, 2012.
11. Stephan Aier, Stephan Kurpjuweit, Jan Saat, and Robert Winter. Enterprise architecture design as an engineering discipline. *AIS Transactions on Enterprise Systems*, 1(1):36–43, 2009.
12. Sabine Buckl, Florian Matthes, Sascha Roth, Christopher Schulz, and ChristianM Schweda. A conceptual framework for enterprise architecture design. In Erik Proper, Marc M. Lankhorst, Marten Schönherr, Joseph Barjis, and Sietse Overbeek, editors, *Trends in Enterprise Architecture Research*, volume 70 of *Lecture Notes in Business Information Processing*, pages 44–56. Springer, Berlin and Heidelberg, New York, 2010.
13. The Open Group. *Archimate 2.1 Specification*. Van Haren Publishing, Zaltbommel, 2012.
14. Ulrich Frank, David Heise, Heiko Kattenstroth, Donald Ferguson, Ethan Hadar, and Marvin Waschke. Itml: A domain-specific modeling language for supporting business driven it management. In Matti Rossi, Jeff Gray, Jonathan Sprinkle, and Juha-Pekka Tolvanen, editors, *Proceedings of the 9th OOPSLA workshop on domain-specific modeling (DSM'09)*, Helsinki, 2009. Helsinki Business School.
15. Ulrich Frank, David Heise, and Heiko Kattenstroth. Use of a domain specific modeling language for realizing versatile dashboards. In Juha-Pekka Tolvanen, Matti Rossi, Jeff Gray, and Jonathan Sprinkle, editors, *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, Helsinki, 2009. Helsinki Business School.
16. Ulrich Frank. Outline of a method for designing domain-specific modelling languages.
17. Ulrich Frank. The memo meta modelling language (mml) and language architecture: 2nd edition.
18. Ulrich Frank. Multi-perspective enterprise modeling: Foundational concepts, prospects and future research challenges. *Software and Systems Modeling*, 2013.
19. Douglas R. Hofstadter. *Godel, Escher, Bach: An eternal golden braid*. Basic Books, New York, 1979.

20. Tony Clark, Paul Sammut, and James Willans. *Applied Metamodelling: A Foundation for Language Driven Development*. Ceteva, 2 edition, 2008.
21. Tony Clark and James Willans. Software language engineering with xmf and xmodeler. In Marjan Mernik, editor, *Formal and Practical Aspects of Domain-Specific Languages*, pages 311–340. Information Science Reference, 2012.
22. Bernd Neumayr, Manfred A. Jeusfeld, Michael Schrefl, and Christoph Schütz. Dual deep instantiation and its conceptbase implementation. In Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff, editors, *Advanced Information Systems Engineering: 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, pages 503–517. Springer International Publishing, Cham, 2014.
23. Alessandro Rossini, Juan De Lara, Esther Guerra, and Nikolay Nikolov. A comparison of two-level and multi-level modelling for cloud-based applications. In Gabriele Taentzer and Francis Bordeleau, editors, *Modelling Foundations and Applications: 11th European Conference, ECMFA 2015*, pages 18–32. Springer International Publishing, Cham, 2015.
24. Juan De Lara, Esther Guerra, and Jesús Sánchez Cuadrado. When and how to use multilevel modelling. *ACM Trans. Softw. Eng. Methodol.*, 24(2):12:1–12:46, 2014.
25. Ulrich Frank. Domain-specific modeling languages - requirements analysis and design guidelines. In Iris Reinhartz-Berger, Aron Sturm, Tony Clark, Yair Wand, Sholom Cohen, and Jorn Bettin, editors, *Domain Engineering: Product Lines, Conceptual Models, and Languages*, pages 133–157. Springer, 2013.
26. Ulrich Frank. Specialisation in business process modelling: Motivation, approaches and limitations.