Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 9–20
DOI: 10.15439/2024F686
ISSN 2300-5963 ACSIS, Vol. 39

Multi-Level Language Architectures as a Foundation for Advanced Enterprise Systems

Ulrich Frank
Department of Computer Science
University of Duisburg-Essen
Germany
0000-0002-8057-1836

***Abstract*—Enterprise systems are the backbone of many companies. Most operational activities are usually not feasible without them. In addition, enterprise systems may also constitute remarkable competitive advantage – or turn out to be a threat to competitiveness, depending on their quality. Enterprise systems in general, ERP systems in particular, have been around for some decades. During this time, they have undoubtedly undergone a maturing process. However, hardly any significant progress has been made regarding foundational architectures and corresponding functions. Based on an analysis of widely undisputed objectives and corresponding shortcomings of current enterprise systems, this paper presents an advanced architecture that enables the construction of *self-referential enterprise systems* (SRES). SRES promise substantial progress with respect to various essential objectives of enterprise systems. The proposed architecture is based on a multi-level language architecture. Among other things, it allows for the integration of enterprise models and corresponding software at run-time. Thus, it does not only boost reuse and adaptability, but substantially fosters user empowerment, too.**

***Index Terms*—integration, reuse, adaptability, conceptual model, enterprise model, self-referential enterprise system, DSML**

I. Introduction

TODAY'S enterprises depend on software systems. Among others, software systems are of pivotal relevance for resource management, for running business processes and for decision making. Among a plethora of specific systems, there are a few software systems that are of general relevance for a wide range of companies, e.g., systems for human resource management, for customer management, for stock management, to name a few only. The most prominent, not to say prototypical example of enterprise software are enterprise resource planning (ERP) systems. In the following, I will subsume these enterprise software systems under the umbrella term of *enterprise systems*, with specific emphasis on ERP systems.

Over the past decade, there have been various technological trends that have had an impact on the development and use of business software. Driven by the availability of non-volatile RAMs at affordable prices, the old idea of In-Memory databases [1] found its way into commercial ERP systems. It is suited to substantially increase the performance of extensive data analysis processes. Thus, it allows to integrate OLAP and OLTP functionality in one database management system. This not only means that more up-to-date data can be used in analyses, but also makes the data warehouse system partially obsolete. Microservices represent another trend in enterprise computing. Whilst their name is misleading, they can make an important contribution to scalability and, related to the previous, to deployment of enterprise systems. Microservices may have an impact on the architecture of software systems, especially in cases where the load different parts of a system have to handle, varies. Related, but not confined to microservices is a further trend that concerns the management of enterprise systems. "Software as a Service" does not only relieve the burden on internal IT management, but is often accompanied by special billing models that may reduce overall costs – and promise better scalability. These innovations can be of great benefit and may even be a prerequisite for the realisation of certain business models. Nevertheless, they have no significant impact on the basic architecture and functionality of ERP systems.

Other trends concern development and maintenance of enterprise systems. The idea of model-driven development (MDD) has been around for some time [2]. It is based on the convincing assumption that focusing on conceptual models without the need to bother with peculiarities of implementation languages is suited to contribute to productivity of software development projects and to software quality alike. As we shall see, MDD suffers from certain pitfalls, which may have contributed to the disappointing fact that it seems not to have a substantial impact on the development and maintenance of enterprise systems. Recently, a considerable hype was generated by so called "low-code" platforms. They offer the prospect of enabling employees without specific programming knowledge to develop software. The idea is not to develop large systems, but rather to quickly create smaller systems tailored to specific needs, which are suitable for partially replacing the use of spreadsheet programs, for example. Irrespective of the fact that low-code represents a clearly exaggerated marketing trend, it can hardly be assumed that the design and functionality of business software will be influenced by it. For critical accounts of low-code platforms see [3], [4].

Our brief overview of developments that had or might have an impact on the realization and use of enterprise systems

shows that progress that concerns the principal architecture was rather limited. Even though it is interesting to ask about the reasons for this, I will refrain from analysing them here. Instead, in the following we want to explore the question of how future enterprise systems could be designed in order to offer significant advantages. To that end, I will at first look at widely undisputed objectives that enterprise systems should satisfy and identify the pivotal measure to achive them. Against that background I present a vision of future enterprise systems, which we refer to as *self-referential enterprise system* (SRES), that goes clearly beyond the possibilities of current systems. Regarding its implementation, the vision is confronted with considerable challenges that can hardly be overcome with conventional language architectures. However, as I will show, a multi-level language architecture is suited to build and run SRES.

II. Enterprise Systems: Undisputed Objectives and Challenges

Various approaches to develop an idea of how to improve enterprise systems are conceivable. One could ask experienced users to report on aspects of current systems they are not satisfied with – and to express requirements future enterprise systems should fullfil. Alternatively, it would be an option to study architectures of existing systems in order to identify serious weaknesses that call for better solutions. Both approaches require considerable effort. In addition, they are accompanied by specific methodical challenges that make the success of such studies questionable. We therefore choose a different approach. Apart from specific objectives and corresponding requirements, there are various goals and related issues that should be widely agreed upon. An analysis of these objectives is not only suited to identify shorcomings of existing systems, but also to provide insights into how future systems could be designed to represent significant progress.

A. Reuse

Reuse is of pivotal relevance for the economics of enterprise systems. That does not only concern development costs, but also the effort to adapt a system to changing requirements. Especially in cases where reuse enables significant economies of scale, cost reductions can be tremendous. Reuse of software artefacts among a range of companies implies the identification of common requirements. In other words: reuse depends on *abstraction* – from specific peculiarities of certain systems onto invariant commonalities shared by a range of systems. A closer look at reuse recommends differentiating between range and productivity of reuse, also known as the power-generality trade-off [5]. The more specific a reusable artefact is, the higher is its contribution to development productivity in cases where it fits – the lower are, however, chances that it fits. On the other hand, the more generic a reusable artefact is, the higher is its potential range of reuse, hence, the achievable economies of scale. Hence, there is need to find a proper trade-off between power and generality or, certainly better, to relax this conflict of goals. Even though the idea of reuse is especially related to software artefacts, enterprise systems should also promote the reuse of knowledge among its users. This requires to account for diverse needs and abilities of users (see Subsection II-B).

B. Accounting for Context and Perspectives

An enterprise system is not an end in itself. It is supposed to support the business. That requires accounting for relevant aspects of a company's action system, such as corporate goals, business processes, organizational structure, or decision scenarios. If the relevant context is not represented in the enterprise system, it will usually be documented separately, more or less accurate and reliable. This does not only create issues with accessibility of relevant documents, but also with their consistency. As a consequence, it is demanding to assess how well IT and business are aligned. If a changing environment demands for adapting the business or even the business model, it is required to account for both, the enterprise system and a company's action system. Again, without a representation of relevant aspects of its context, it requires additional effort to provide for conjoint change of business and IT.

Large organizations depend on separation of concerns, which translates to a variety of different professional perspectives that comprise specific goals, interests and technical languages. To provide effective support, an enterprise system should offer appropriate representations for all perspectives relevant for its users. Appropriately designed user interfaces that allow for individual adaptions are very useful in this respect. However, they are hardly sufficient to help users with gaining a deeper understanding of the system they use, the company they work for, and how their work relates to the work of others.

C. Reduction of Complexity and Need for Transparency

Enterprise systems are supposed to reduce an organization's complexity. At the same time, they contribute to a subtle increase of complexity. Often, software systems penetrate companies to a degree that many employees perceive their work through the applications they use. In other words: corporate reality is more and more constructed through enterprise systems. At the same time, to most employees the software they use remains a black box. That is not only in obvious contrast to the idea of enlightenment, which demands for a demystification of the world that surrounds us, it is also a threat to a company's competitiveness, which requires employees that are able to assess limitations and possible modifications of the systems they use. In addition, enterprise systems are part of ever growing IT infrastructures with a huge amount of different elements and dependencies between these. The resulting complexity is a clear threat to IT management and, hence, to the efficient use of IT infrastructures.

D. Integration

Integration is a prerequisite of the efficient use of enterprise systems. It requires accounting for various aspects. First, one needs to distinguish static, functional and dynamic integration.

In all cases, integration requires the affected software systems to communicate, which in turn requires common concepts, materialized, e.g., through datatypes, classes, database schemas, interface types, event types, etc. Second, similar to reuse, there is a conflict between generality and power to be accounted for. The more specific common concepts are, in other words: the more semantics they carry, the more efficient and safe communication can be, hence, the higher is the level of integration. However, the more specific concepts that enable integration are, the more systems will be excluded. This corresponds to the use of technical languages by humans. In addition to common concepts, integration of software systems also recommends the common representation of corresponding instances in order to avoid redundancy, which in turn requires common *namespaces*.

A further aspect of integration concerns organizational integration, which corresponds to IT-business alignment. Integration of this kind, too, requires common concepts shared by the two worlds. If an enterprise system requires users to know technical concepts such as file, record or module, organizational integration will be weaker than it would be with using domain-specific concepts users are familiar with. Like reuse, integration requires abstraction – on common concepts and from specific details that are peculiar to certain systems or users.

E. Adaptability

The requirements an enterprise system should satisfy may change over time. In this case, it is of crucial importance that it can be adapted with little effort and risk. At best, possible changes had been accounted for already, when a system was first designed. Ideally, this would be reflected by a software architecture that separates a presumably invariant core from possibly variable parts. If the variable parts represent monotonic extensions of the core, which is the desirable case but not trivial to achieve, changes to variable parts would not have side-effects on the core. The prerequisite of such an architecture is abstraction. Only if one succeeds in abstracting onto invariant properties of a system, these could be bundled in an invariant core. In an ideal case, changes could be performed by competent users without the need to dive into source code.

The quest for adaptability is confronted with a conflict of goals, too. It is reflected by the notions of loose and tight coupling. Loose coupling, which is favored by many as an effective measure to achieve adaptability, aims at reducing dependencies between components – in other words: it builds on generic rather than on specific interfaces – to facilitate their replacement. Abstracting onto commonalities of a range of components creates dependencies: more specific components chiefly depend on more generic ones. As long as these dependencies are invariant, they are of no harm, but of great benefit. All dependent components can be easily changed by changing the common abstraction. Fig. 1 illustrates the advantage of tight coupling in this case – and indicates the problems that arise from abstractions that turn out to be inappropriate. Related to that, there is another conflict to account for. Adaptations of an enterprise system require some kind of language. If this language is generic, as in the case of a general-purpose programming language, a wide range of changes is possible. However, changes of this kind are very time-consuming and risky. On the other hand, a language that clearly restricts changes is likely to reduce effort and risk. Examples include approaches to configuration or domain-specific languages (DSMLs).

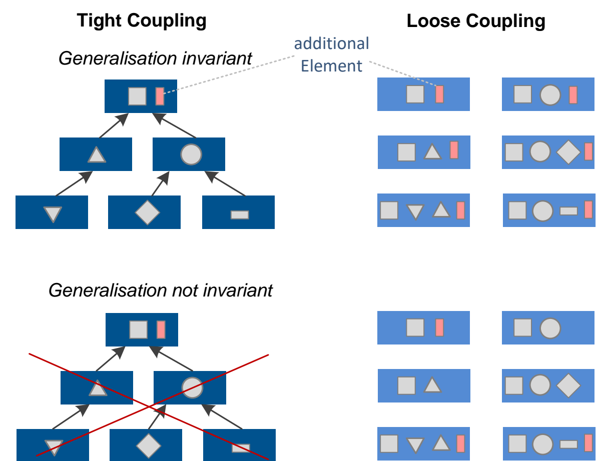


Fig. 1. Comparison of tight and loose coupling

F. Preliminary Conclusions

Our brief overview of objectives that should be widely undisputed reveals the following insights.

- Abstraction is of pivotal relevance. It is the prerequisite of reuse, integration and adaptability.
- Conceptual models are a useful instrument for developing abstractions of high quality. At best, they allow users to participate in their development and evaluation. Conceptual models could also serve as a representation that competent users could change without the need to bother with code.
- Semantics is likely to produce serious goal conflicts that require painful trade-offs. Therefore, approaches that allow mitigating these conflicts promise great benefits. Generalization/specialization is an example of how such a mitigations could work. At a higher level of generalization, a wider range of (re-) use can be expected, whereas more specific levels contribute to higher productivity, while they, at the same time, benefit from greater economies of scale through reusing higher level concepts.
- To take advantage of powerful abstractions, languages are required that provide concepts which allow for expressing these abstractions. The examples in Fig. 5 illustrate the problem. As we shall see, mainstream programming languages – and modeling languages alike – are seriously limited in this respect.

III. Multi-Level Self-Referential Enterprise System

Our first vision of future enterprise systems emerged some time ago. It was mainly inspired by our work on enterprise modeling. It was mainly focused on leveraging the utility of enterprise modelin tools by integrating them with enterprise software. Unfortunately, the vision suffered from serious feasibility problems. Only later, as an outcome of our research on multi-level language architectures, we were able to further refine the vision and to substantiate the design with an architecture that makes it feasible.

A. The Early Vision

The idea of enterprise modeling has been around for some time [6], [7]. It is based on the assumption that an organisation's information system and its action system call for conjoint analysis and design in order to fully exploit the potential of IT. Therefore our work on enterprise modeling was at first focused on supporting early phases of enterprise systems' life-cycle. It resulted in a method for multi-perspective enterprise modeling (MEMO, for an overview see [8]), which includes various domain-specific modeling languages (DSMLs), e.g. for modeling corporate goals [9], [10], IT infrastructures [11], organisation structures [12], business processes [13], and decision processes [14]. These languages are integrated through a common meta-metamodel and common concepts.

Since the proper use of DSMLs as well as the analysis and management of enterprise models demand for supporting tools, we put considerable effort into the development of modeling tools [7], [15], [16]. An enterprise modeling environment such as MEMO4ADO [16] does not only allow to create the various particular models, e.g., business process models, goal models or models of the IT infrastructure. It also integrates them, thus, ensuring referential integrity of modeling elements and allowing for cross-model analysis, e.g., by allowing to navigate from a business process model to all resources that are required for its execution. Fig. 2 shows an overview of diagram types produced with MEMO4ADO and illustrates their integration through common concepts.

These benefits of a traditional environment for enterprise modeling are contrasted with serious limitations. First, models focus on the type level only. This is for a good reason. Usually, we want to intentionally fade out particular instances, since they are changing all the time. However, there are analysis scenarios where instances are important. For example, one may want to know how many instances of a certain business process types were exectuted within a certain month, or when a particular instance started. Other examples include the number of invoices or the invoice with the highest amount etc. To answer questions related to the instance level, one would have to use a corresponding enterprise system. If this system is not integrated with a corresponding enterprise model, it would not be possible to nagivate from one system to the other – an obvious obstacle to decision making.

There is a further reason for integrating an enterprise modeling environment with an enterprise system. The development

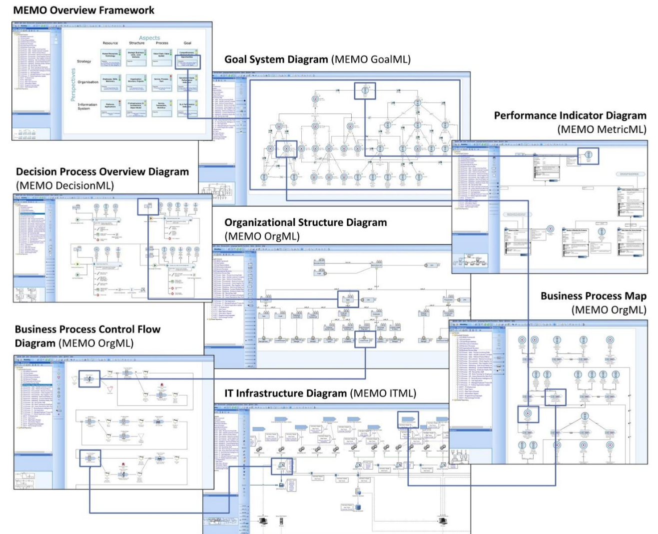


Fig. 2. Elements of MEMO4ADO

of an enterprise model requires considerable effort. Also, parts of an organizational information system are always in the state of change. Even a smoothly working IT infrastructure and well-designed business processes create the need for corresponding models in order to cope with complexity – no matter whether they are in an early or late state of their life-cycle Hence, in order to not waste valuable resources and to enable additional benefits, we came to the conviction that it should be possible to use enterprise models during the entire life-cycle of a company and its information system, both as an instrument to support management and as a means to empower all employees by improving transparency.

This idea led quickly to the vision of integrating enterprise models – more precisely: tools for enterprise modeling – with enterprise software. We referred to this vision as "self-referential enterprise system" (SRES) [17]. An SRES results from the integration of an enterprise modeling environment and a corresponding enterprise software system. In an ideal case, developers and competent users could apply changes to parts of an enterprise model which then would become effective in the enterprise system.

To develop a demonstration of an SRES, we aimed at extending an existing enterprise modeling environment accordingly. Unfortunately, we soon had to realize that there were serious problems standing in the way of integrating the two systems. These problems were caused by principal limitations of implementation languages.

B. Challenges

These limitations create serious challenges to the design of SRES. They mainly comprise two interrelated aspects. First, mainstream programming languages do not allow for the straightforward implementation of modeling environments that represent instance level data. Second, related to that, modeling languages that are based on a MOF-like architecture

do not allow for expressing knowledge about instances. The first aspect is illustrated in Fig. 3. It shows a UML class that is conceptually located at M1. However, within a modeling tool, it is implemented as an object at M0. This is for a serious reason. A modeling tool needs to allow defining and changing the properties of a class. However, only objects have state, which can be manipulated. As a consequence, it is not possible to create instances of classes within a model editor. The only option is to generate code – resulting in two separate representations.

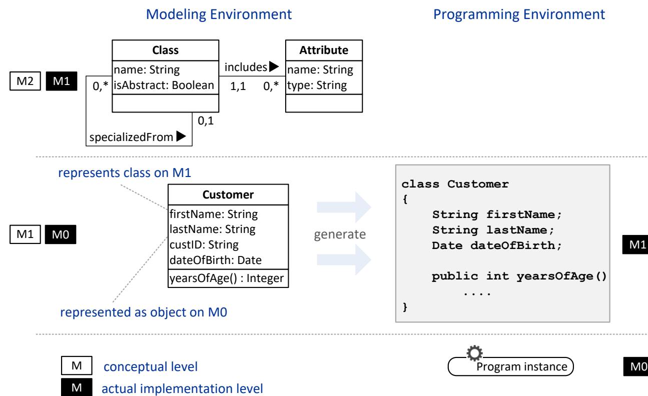


Fig. 3. The need for generating code

As a consequence of these limitations, the architecture we developed with our first conception of SRES was based on a pragmatic notion of integration. At first, concepts defined within an SRES had to be replicated in the corresponding enterprise system. That could, at best, be achieved by generating code from models. Then, both systems had to be integrated through interfaces that allow requests made in one system to be forwarded to the other system. If, e.g., a user who studies the model of an IT infrastructure within the enterprise modeling tool wants know what instances of a certain platform type exist and where they are located, the corresponding interface should allow to send this request transparently to the enterprise system. At the same time, a process manager who is not happy with the performance of a certain business process could navigate to the corresponding process model in the enterprise modeling environment, where he might decide to change the model, which should then lead to the adaptation of the corresponding process schema in the enterprise system. Fig. 4 shows an outline of the corresponding architecture.

It is needless to say that we were not satisfied with this solution. It reflects a poor concept of integration that requires ongoing synchronization. Since new requirements often need to be implemented under time pressure, it is likely that changes are directly applied to code, which over time leads to a depreciation of the corresponding model. In addition, the second problem, namely the lack of expressiveness of modeling languages, could not be overcome with the proposed architecture. An enterprise modeling environment needs to offer DSMLs. Otherwise users would have to model goals, business processes etc. from scratch, which would not only cause inacceptable effort, but would also be a threat to integrity. The concepts provided by a DSML serve the specification of types or classes. Fig. 5 shows fragments of two possible DSMLs and corresponding models. The concept of a printer may be part of a DSML to model IT infrastructures, whereas the concept of an activity may be offered by a DSML for modeling business processes.

Unfortunately, it is not possible to express that a particular printer has a serial number or a certain number of printed pages with the DSML, even though we know that these properties are required. Accordingly, we cannot express the knowledge that a particular business process has a start time and end time, since corresponding attributes would apply to a certain business process type, not to its instances. To express this knowledge, it would have to be added redundantly to every instance of the metaclasses **`Printer`** or **`Activity`**. However, even such a dissatisfactory approach would not allow to subsequently create particular instances within a model editor – for the reasons illustrated in Fig. 3.

A closer look at the model fragments in Fig. 5 reveals a further challenge. The specification of a metaclass like **`Printer`** has to be done from scratch requiring the language designer to know essential properties of a printer. Would it not be more appropriate to use an existing, more generic DSML that already includes a general concept of printer to define printer models? As we shall see such an approach would contribute to the more efficient development of DSMLs and would, at the same time, be suited to improve their quality.

IV. Multi-Level Language Architectures to the Rescue

To cope with the limitations of the MOF architecture, we extended our previous meta modeling language with so called “intrinsic features” that allow to define features such as attributes in a meta class at M2, hence as part of a DSML, which are to be instantiated only with an instance at M0. This extension allowed, e.g., to express the fact that a particular printer has a serial number (see Fig. 5) with a DSML by characterizing the corresponding attribute as intrinsic. Unfortunately, this was little more than a Pyrrhic victory, since intrinsic features could not be expressed by common object-oriented programming languages. Furthermore, the extension was limited to M2 and there were indications already that higher levels of classification might be useful.

The back then young field of *multi-level modeling*, a term introduced more than twenty years ago by Atkinson and Kühne [18], with ancestors that go back even further, cf. [19]–[22], promised to address our needs more convincingly. However, multi-level modeling languages were not accompanied by corresponding programming languages, which we needed for our purpose. Then, about 15 year ago, a discussion at a conference dinner lead to a solution. The *XModeler*, a language engineering environment developed by Clark et al. [23], [24] proved to feature a language architecture that was suitable for a convincing implementation of SRES. Encouraged by these prospects, we started the project “Language Engineering for

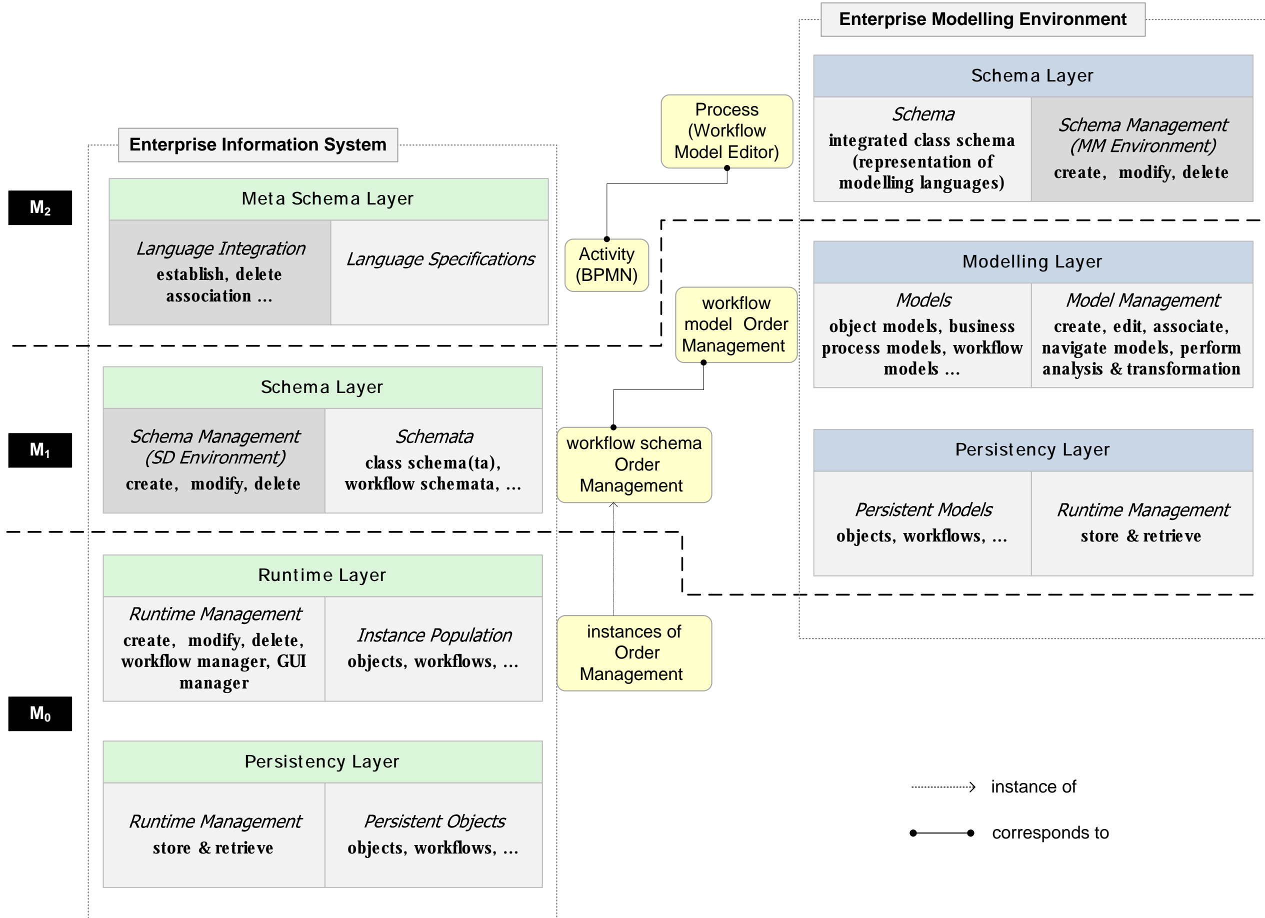


Fig. 4. Outline of early architecture of SRES

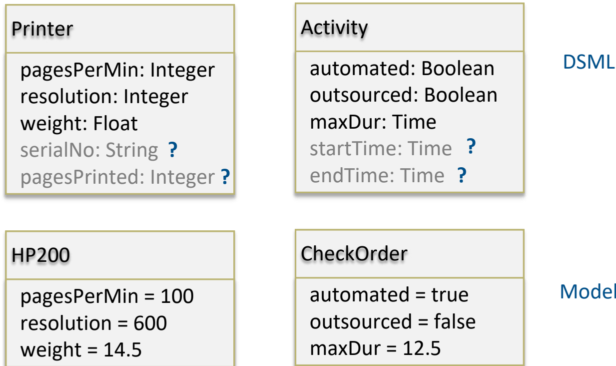


Fig. 5. Limited expressiveness of traditional languages

Multi-Level Modeling" (LE4MM, www.le4mm.org), which is still running today. For a brief history of the project see [25].

A. XModeler^{ML}and FMML^{X}

While the XModeler does not feature a multi-level language, its metamodel, XCore, could be easily extended to enable essential features of a multi-level language: an arbitrary number of classes with an explicit level and deferred instantiation of properties such as attributes or operations. This extension led to the specification and implementation of FMML^{X}, a multi-level modeling language [26]. Different from other multi-level modeling languages such as LML [27] or M-Objects [28], FMML^{X}is executable, that is, it features a common representation of models and corresponding programs. The implementation of FMML^{X}in the XModeler led to the XModeler^{ML}. It is, together with various additional resources such as screencasts and publications, available on the project's webpages at www.le4mm.org.

The language architecture enabled with the FMML^{X} allows to overcome the lack of expressiveness traditional language architectures suffer from. The small FMML^{X} model in Fig. 6 corresponds in part to the example in Fig. 5. It illustrates how knowledge that cannot be expressed with traditional languages can be represented without redundancy.

The class **`Product`** at level 3 serves the definition of properties that apply to all kinds of devices. The class **`Printer`** in part inherits these properties, in part instantiates them. Therefore, a specialization hierarchy would not be sufficient.

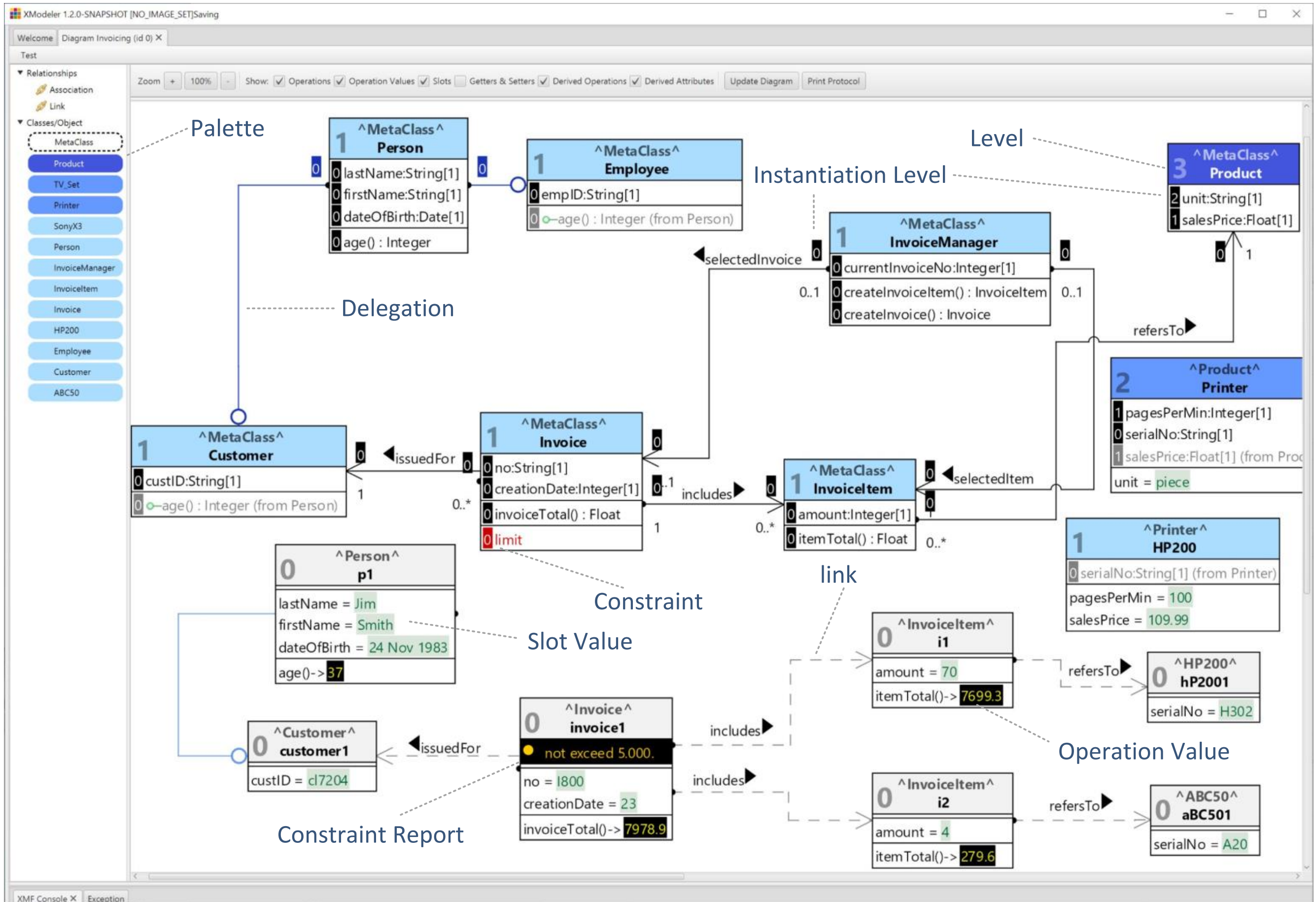


Fig. 6. Example FMML^{X} model created with the XModeler^{ML}

The number printed in white in a black rectangle next to an attribute or operation is to indicate the level where it is supposed to be instantiated or executed. The fact that every class is an object is, on the one hand, illustrated by slot values, such as 100 for **`pagesPerMin`** in the object **`HP200`**, which represents a printer model at level 1. On the other hand, it is shown by the values returned from the operation **`invoiceTotal()`** executed by instances of the classes **`Invoice`**, **`HX500`** or **`Person`**. In other words: an FMML^{X} model is executable. The objects it consists of can be displayed in a diagram editor, either with a standard or a customized notation, or by using an object browser or a customized GUI. To strengthen the integrity of the model – and of the executable program simultaneously – the FMML^{X} allows for adding constraints to classes, which are then immediately evaluated as soon as corresponding instances are created or changed. For exampole, the constraint **`limit`** in **`Invoice`** defines that the total of an invoice must not exceed 5.000, which is violated by the object **`invoice1`**.

The model also shows that multi-level models overcome the traditional distinction of modeling language and model. As soon as a class is created, it extends the palette and can be used to create further instances. Furthermore, there can be links between objects at any level. A language, which is a model at a higher level, can be changed at runtime, which leads to an immediate update of the affected models. Note, however, that changes at higher levels can be challenging. Therefore, it is of crucial importance that concepts represented in a multi-level model are the more invariant the higher they are located in the hierarchy [29].

An arbitrary number of classification levels is enabled by a reflexive and recursive metamodel that specifies and implements the FMML^{X}. Fig. 7 shows the simplified meta model with a few selected constraints. Since **`Class`** inherits from **`Object`**, every class in the system is an object, has state and can be executed. FMML^{X} objects are instantiated from **`MetaClass`**. With their instantiation they are assigned an object of the **`Level`**. It allows to either define a specific level through an integer or to define a range of possible values in case the level of a class should be contingent [30]. Deferred instantiation of properties such as attributes, operations or associations is enabled through the attribute **`instLevel`** that serves the definition of the intended instantiation level. For a more comprehensive description of the metamodel, the related

instantiation mechanism as well as the overall architecture of the XModeler^{ML} see [31], [32].

The XModeler^{ML}, which is provided as open source, and multiple resources including screencasts, publications and example models are available on the LE4MM webpages at www.le4mm.org.

B. A Foundation for Self-Referential Enterprise Systems

A multi-level language such as the FMML^{X} and a corresponding language engineering and execution environment as the XModeler^{ML} provide a powerful foundation for SRES. First, they allow a common representation of models and programs. Hence, there is no need for the synchronization of two different representations. Users of an SRES can navigate from the software they use to corresponding models and meta models. If they are qualified and authorized, they may also change a model with the effect that the software they use is instantly changed, too. A multi-level model of an enterprise may be comprised of multiple DSMLs, which are defined at different levels. Since these DSMLs are all executable, they are domain-specific programming languages at the same time.

Such a language architecture supports reuse and adaptability. The knowledge represented in DSMLs can be reused. If a DSML does not fit specific needs, the language it was specified with can be used to define a new customized DSML. This alllows to benefit from economies of scale supported by high level DSMLs and to benifit from the productivity provided by more specific DSMLs, thus relaxing a crucial design conflict. An SRES would then be based on a multi-level model and a corresponding runtime environment. In addition, there would be a component that serves making object models persistent and supports object retrieval. Further components would enable presentation and interaction. Since the XModeler^{ML} supports the MVC pattern, multiple views could be added to predefined diagram and browser views. Fig. 8 shows a highly simplified representation of the architecture.

C. Illustration

At an operational level, an SRES would provide GUIs similar to those known of today's ERP systems. These allow to access objects at level 0, that is, objects that represent data about particular entities or aggregations of these. In addition, an SRES would allow users to navigate to elements of the integrated enterprise model, which are typically located at level 1. These models can be presented in diagram editors featuring a standard or a customized graphical notation. In addition users could also be offered textual representations of these models. In case, users are overwhelmed by distinguishing different levels of abstraction, they could also be provided with a more traditional GUI that allows for accessing objects at different levels without the need to understand the notion of a classification level. For those users and administrators who want to understand or eventually change the DSMLs used to specify the models, an SRES would allow for accessing the full multi-level model representing an SRES. It could be represented by diagrams, within an object browser or in text editors. The dotted edges between selected elements of the different layers are to indicate that all these representations are integrated, since they are only different views of the same multi-level system. Fig. 9 illustrates how the various levels of abstraction covered by a multi-level SRES can be presented to users.

D. Brief Evaluation

Instead of a comprehensive evaluation of multi-level architectures which can, e.g., be found in [31], I will focus on a few essential aspects only, referring to the objectives described in Section II. The integration of enterprise software with a corresponding enterprise model is obviously suited to empower users, since they have a much better chance to understand and eventually change the software they deal with. Since an SRES provides an integrated representation of company's action system and its information system, users are also supported with aligning business and IT. The complexity inherent especially to larger organizations is reduced by models that were created with DSMLs. The common representation of models and programs allows for doing without two separate representations. This does not only foster referential integrity, but also supports protection of investments into models, which otherwise are likely to be devaluated over time. Adapting an SRES to changing requirements is, at best, facilitated by applying changes at a higher level in the hierarchy only once instead of repeatedly at lower levels. In addition, a multi-level architecture also fosters reuse and, hence, economics of acquiring and managing enterprise systems. Furthermore, it also promotes cross-organizational integration of enterprise systems. Integration depends on common concepts. If, e.g., company A sends a message to company B referring to the particular printer model "HP200", communication would fail, if the software company B is using does not know a corresponding class. Within a traditional scenario, there would be no way to a apply a usefull interpretation of an unknown class. It would just be some class. In case of a multi-level architecture, there would be the chance to identify it as some kind of printer, if the corresonding class was known by B.

In light of these attractive prospects, it does not seem too daring to claim that multi-level architectures are suited to make enterprise software clearly more powerful. This claim leads to the obvious question why multi-level architectures have not taken off yet. There are various reasons for this unpleasant situation. First, the benefits of multi-level architectures are not easy to understand. Second, for legitimation reasons decision maker tend to opt for mature mainstream solutions. Multi-level systems are definitely not mainstream. Existing implementations of development and execution environments are restricted to academic prototypes. Third, there may be principal objections against multi-level modeling, since it may seem strange to those who are used to languages that are rectricted to one classification level only. Multi-level models provide indeed features unknown of in traditional modeling and programming languages. Not only that they allow for an arbitrary number of classification levels and regard all

Fig. 7. Meta model of FMML^X

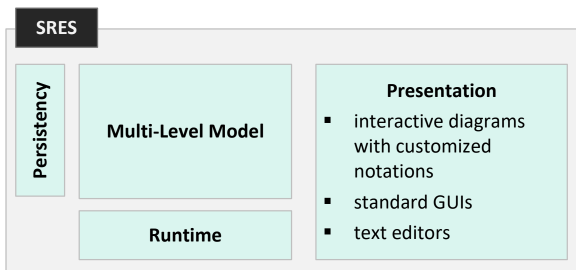
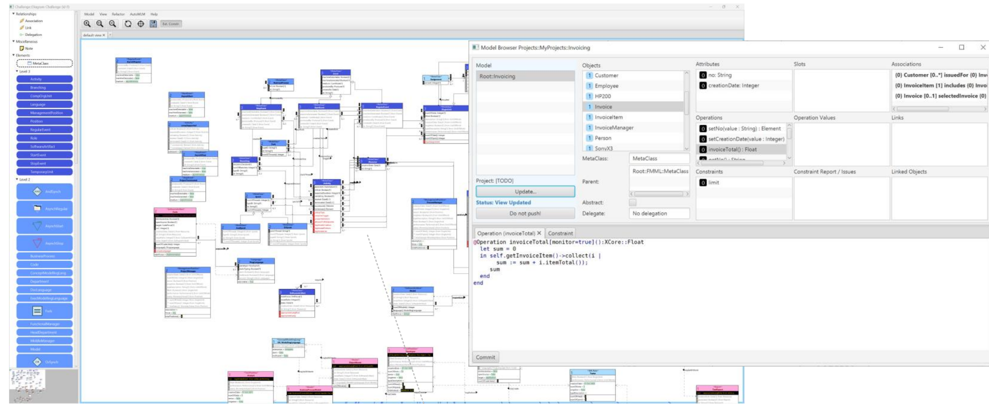
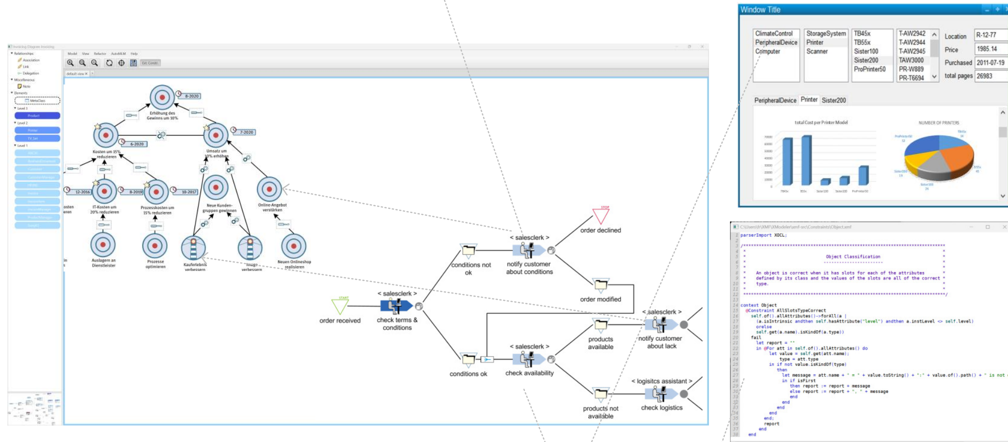


Fig. 8. Sketch of multi-level architecture of SRES

classes as objects, they also blur the boundary between a language and its application. Therefore, it seems appropriate to speak of a paradigm shift. However, as the example in Fig. 10 shows, natural languages and especially technical languages in advanced societies evolve in a hierarchical order. The introduction of a new technical language is usually based on a refinement of existing technical languages.

A further objection against multi-level architectures relates to their complexity. It is indeed clearly more demanding to develop a multi-level model than a traditional model. However, that is the case, too, for developing a compiler-compiler compared to a developing a compiler, or for developing a





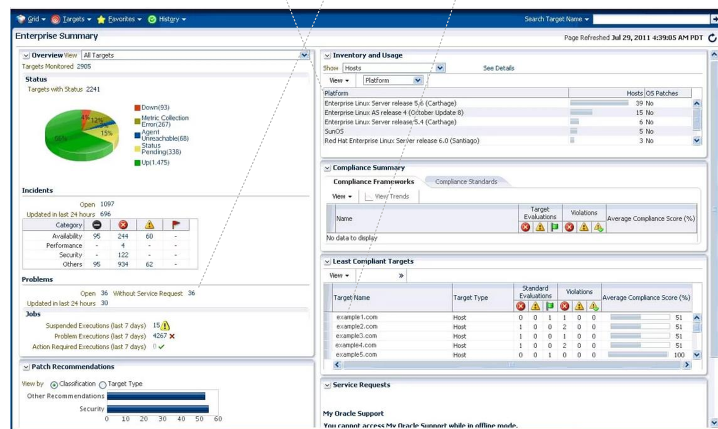


Fig. 9. SRES from user perspectives

GUI builder compared to designing a particular GUI. Reducing complexity implies increasing it first. Those who design multi-level models are confronted with remarkable complexity, especially in cases where requirements vary to a large extent. Those, however, who use an existing multi-level model and fit it to more specific needs benefit from a level of complexity that is certainly lower than that of creating a UML class diagram from scratch or dealing with representations that are used for the configuration of ERP systems.

Apart from these obstacles there are a few specific peculiarities and restrictions that prevent the outlined multi-level architecture of SRES from being a silver bullet. First, the key features of a language engineering, modeling and execution environment like the XModeler^{ML}, such as an arbitrary number of classifiation levels and executable objects at any level are possible only through dynamic typing. Despite these obvious advantages, dynamic typing is sometimes met with reservations. Type checking happens at runtime only and, compared to languages that feature static typing, the code carries less information. It is, for example, not possible to determine the

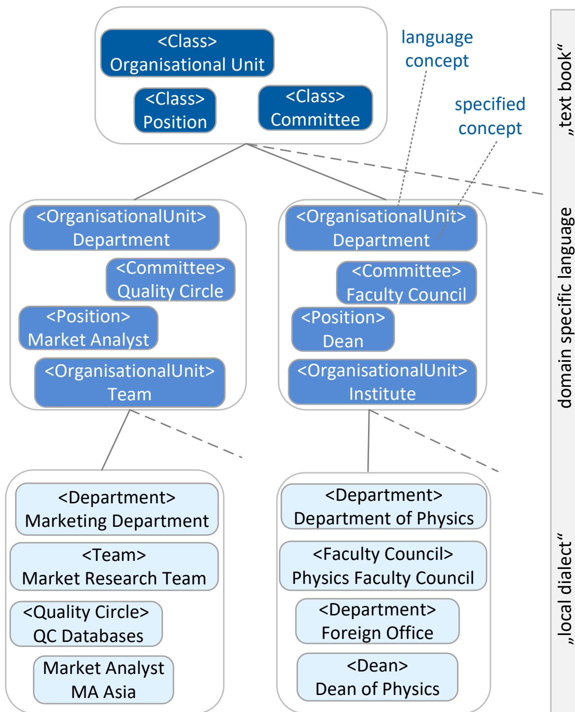


Fig. 10. Multiple levels of concepts in natural language

class of an object a message is sent to in a straightforward way. We believe that this potential advantage of statically typed languages is more than offset by the specific benefits of languages that feature *strong* dynamic typing. Languages like Smalltalk have demonstrated these benefits long ago. Looking back, it is regrettable that Smalltalk was sidelined primarily because the hardware available at the time was unable to compensate for the disadvantages. For a revealing discussion of the specific advantages of dynamic typing and its historical obstacles see this interview with Alan Kay [33]. It is needless to say that performance is not an issue anymore with today's hardware. Second, the design of a multi-level model requires to carefully decide for a trade-off between flexibility and integrity, which is not trivial [34] (see also Subsection II-E).

Nevertheless, the realization and maintenance of multi-level language architectures is challenging. A multi-level hierarchy is extremely useful for maintaining a system as long as the dependencies reflected by the hierarchy – lower level objects depend existentially on higher level classes – are invariant over time. Therefore, the design of multi-level models requires a high level of expertise and great care. Otherwise, the advantage turns into a serious problem. A further aspect is of utmost relevance with respect to the power of multi-level language architectures. So far, multi-level models are widely restricted to static and, to a lesser degree, functional abstractions. It is much harder to define multi-level semantics for dynamic abstractions, e.g. process models. There are important reasons for this, such as the fact that specialization of process types cannot be defined as monotonic extension of a super process type. As a consequence, the substitutability constraint cannot be satisfied – with serious implications for the maintenance of larger process landscapes. There are a few contributions to multi-level process modeling, e.g., [35], [36], [37], [38], but their main focus is on abstracting on static or functional aspects of processes. While the missing support for dynamic abstractions does not invalidate the benefits gained from multi-level static abstractions, it clearly emphasized the need for corresponding research.

V. Conclusions and Future Research

While ERP systems are of pivotal relevance for many companys' competitiveness, only little research on future enterprise software systems happens in academia. At the same time, progress of commercial systems remains modest, at least with respect to principal functionality. Enterprise modeling, on the other hand, has seen more than two decades of research in academia, but only little adoption in business. Nevertheless, the potential benefits of enterprise models are widely undisputed. The presented architecture of SRES is suited to promote the utilility of enterprise models and, at the same time, improve the power of enterprise software. While the implementation of SRES is widely impossible with prevalent language technologies, multi-level languages and corresponding development and execution environments provide a solid foundation for that purpose. In addition to enabling SRES, multi-level language architectures also allow for enriching other types of software with additional abstraction.

Our future research is primarily characterized by two directions. On the one hand, we will continue to work on concepts that allow for multi-level dynamic abstractions. In doing so, we are thinking about developing a relaxed concept of specialization. On the other hand, our work aims to simplify the transition to multi-level modeling by supporting the step-by-step enrichment of a UML editor with further concepts up to the XModeler^{ML}. A first prototype of this UML editor, called "UML-MX" is available on the project's webpage at https://www.wi-inf.uni-due.de/LE4MM/uml-pp/. Among other things, it allows the instantiation and execution of objects from a UML class diagram within the model editor.

References

[1] M. H. Eich, "Mars: The Design of a Main Memory Database Machine," in *Database Machines and Knowledge Base Machines*, ser. The Kluwer International Series in Engineering and Computer Science, Parallel Processing and Fifth Generation Computing, M. Kitsuregawa and H. Tanaka, Eds. Boston, MA: Springer, 1988, vol. 43, pp. 325–338.

[2] R. B. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," in *Workshop on the Future of Software Engineering (FOSE '07)*, L. C. Briand and A. L. Wolf, Eds. IEEE CS Press, 2007, pp. 37–54.

[3] A. C. Bock and U. Frank, "Low-Code Platform," *Business & Information Systems Engineering*, vol. 63, no. 6, pp. 733–740, 2021.

[4] J. Cabot, "Positioning of the Low-Code Movement within the Field of Model-Driven Engineering," in *Proceedings of the 23rd ACM/IEEE International Conference on Modell Driven Engineering Languages and Systems*. IEEE, 2020, pp. 535–538.

[5] A. C. Bock, "The Power/Generality Trade-Off in Decision and Problem Modeling: Theoretical Background and Multi-level Modeling as a Resolution," in *Enterprise, Business-Process and Information Systems Modeling*, ser. Lecture Notes in Business Information Processing, J. Gulden, I. Reinhartz-Berger, R. Schmidt, S. Guerreiro, W. Guédria, and P. Bera, Eds. Cham: Springer International Publishing, 2018, vol. 318, pp. 213–228.

[6] J. A. Zachman, "A Framework for Information Systems Architecture," *IBM Systems Journal*, vol. 26, no. 3, pp. 276–292, 1987.

[7] U. Frank, *Multiperspektivische Unternehmensmodellierung: Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung*. München: Oldenbourg, 1994.

[8] ——, "Multi-Perspective Enterprise Modeling: Foundational Concepts, Prospects and Future Research Challenges," *Software and Systems Modeling*, vol. 13, no. 3, pp. 941–962, 2014.

[9] S. Overbeek, U. Frank, and C. A. Köhling, "A Language for Multi-Perspective Goal Modelling: Challenges, Requirements and Solutions," *Computer Standards & Interfaces*, vol. 38, pp. 1–16, 2015.

[10] A. Bock and U. Frank, "MEMO GoalML: A Context-Enriched Modeling Language to Support Reflective Organizational Goal Planning and Decision Processes," in *Conceptual Modeling: 35th International Conference, ER 2016*, I. Comyn-Wattiau, K. Tanaka, I.-Y. Song, S. Yamamoto, and M. Saeki, Eds. Cham: Springer, 2016, pp. 515–529.

[11] U. Frank, M. Kaczmarek-Heß, and S. D. Kinderen, "IT Infrastructure Modeling Language (ITML): A DSML for Supporting IT Management. ICB Report No. 71, University of Duisburg-Essen."

[12] U. Frank, "MEMO Organisation Modelling Language (1): Focus on Organisational Structure."

[13] ——, "MEMO Organisation Modelling Language (2): Focus on Business Processes. ICB Research Report No. 49., University of Duisburg-Essen," 2011.

[14] Alexander Bock, "Beyond Narrow Decision Models: Toward Integrative Models of Organizational Decision Processes," in *Proceedings of the 17th IEEE Conference on Business Informatics (CBI 2015)*, D. Aveiro, U. Frank, K. J. Lin, and J. Tribolet, Eds., Lisbon, 2015.

[15] J. Gulden and U. Frank, "MEMOCenterNG – A Full-Featured Modeling Environment for Organisation Modeling and Model-Driven Software Development," in *Proceedings of the 2nd International Workshop on Future Trends of Model-Driven Development (FTMDD 2010)*, 2010.

[16] A. Bock, U. Frank, and M. Kaczmarek-Heß, "MEMO4ADO: A Comprehensive Environment for Multi-Perspective Enterprise Modeling," in *Proceedings of the Modellierung 2022 Satellite Events*, J. Michael, J. Pfeiffer, and A. Wortmann, Eds. Bonn: GI, 2022, pp. 245–255.

[17] U. Frank and S. Strecker, "Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems: Requirements, Conceptual Foundation and Design Options. ICB Research Report No. 31. University of Duisburg-Essen," Essen.

[18] C. Atkinson and T. Kühne, "The Essence of Multilevel Metamodeling," in *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, ser. Lecture Notes in Computer Science, M. Gorgolla and C. Kobryn, Eds. Berlin and London, New York: Springer, 2001, pp. 19–33.

[19] J. J. Odell, "Power Types," *Journal of Object-Oriented Programming*, vol. 7, no. 2, pp. 8–12, 1994.

[20] R. C. Goldstein and V. C. Storey, "Materialization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 5, pp. 835–842, 1994.

[21] A. Pirotte, E. Zimányi, D. Massart, and T. Yakusheva, "Materialization: A Powerful and Ubiquitous Abstraction Pattern," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 1994, pp. 630–641.

[22] M. Jarke, S. Eherer, R. Gallersdörfer, M. Jeusfeld, and M. Staudt, "ConceptBase – A Deductive Object Base for Meta Data Management," *Journal of Intelligent Information Systems*, vol. 4, no. 2, pp. 167–192, 1995.

[23] T. Clark, P. Sammut, and J. S. Willans, "Super-Languages: Developing Languages and Applications with XMF (2nd ed.)," *CoRR*, 2015. [Online]. Available: http://arxiv.org/abs/1506.03363

[24] T. Clark, P. Sammut, and J. Willans, *Applied Metamodelling: A Foundation for Language Driven Development*, 2nd ed. Ceteva, 2008.

[25] U. Frank and T. Clark, "Language Engineering for Multi-Level Modeling (LE4MM): A Long-Term Project to Promote the Integrated Development of Languages, Models and Code," in *Proceedings of the Research Projects Exhibition at the 35th International Conference on Advanced Information Systems Engineering (CAiSE 2023)*, ser. CEUR, J. Font, L. Arcega, J.-F. Reyes-Román, and G. Giachetti, Eds., 2023, pp. 97–104.

[26] U. Frank, "The Flexible Multi-Level Modelling and Execution Language FMML^{X}. ICB Research Report No. 66. University of Duisburg-Essen," Essen.

[27] C. Atkinson and R. Gerbig, "Flexible deep modeling with melanee," in *Modellierung 2016, 2.-4. März 2016, Karlsruhe - Workshopband*, ser. Modellierung 2016, S. B. U. Reimer, Ed., vol. 255. Bonn: Gesellschaft für Informatik, 2016, pp. 117–122. [Online]. Available: http://subs.emis.de/LNI/Proceedings/Proceedings255/117.pdf

[28] B. Neumayr, K. Grün, and M. Schrefl, "Multi-level domain modeling with m-objects and m-relationships," in *Proceedings of the 6th Asia-Pacific Conference on Conceptual Modeling (APCCM)*, S. Link and M. Kirchberg, Eds. Wellington: Australian Computer Society, 2009, pp. 107–116.

[29] U. Frank, "Prolegomena of a Multi-Level Modeling Method Illustrated with the FMML^{X}," in *Proceedings of the 24th ACM/IEEE International Conference on Modell Driven Engineering Languages and Systems: Companion Proceedings*. IEEE, 2021.

[30] U. Frank and D. Töpel, "Contingent Level Classes: Motivation, Conceptualization, Modeling Guidelines, and Implications for Model Management," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, E. Guerra and L. Iovino, Eds. New York, NY, USA: ACM, 2020, pp. 622–631.

[31] U. Frank, "Multi-level Modeling: Cornerstones of a Rationale," *Software and Systems Modeling*, vol. 21, no. 1, pp. 451—-480, 2022.

[32] T. Clark and J. Willans, "Software Language Engineering with XMF and XModeler," in *Computational linguistics*, I. R. Management Association, Ed. Hershey, Pennsylvania (701 E. Chocolate Avenue, Hershey, Pa., 17033, USA): IGI Global, 2014, pp. 866–896.

[33] S. Feldman, "A Conversation with Alan Kay," *Queue*, vol. 2, no. 9, pp. 20–30, 2004.

[34] U. Frank and T. Clark, "Peculiarities of Language Engineering in Multi-Level Environments or: Design by Elimination," in *Kühne (Ed.) 2022 – Proceedings of the 25th International*, pp. 424–433.

[35] B. Neumayr, C. G. Schuetz, and M. Schrefl, "Dual deep modeling of business processes: 7:1-31 pages / enterprise modelling and information systems architectures (emisaj), vol. 17 (2022)," 2022.

[36] A. Lange and C. Atkinson, "Multi-level Modeling with LML. A Contribution to the MULTI Process Challenge," *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, vol. 17, pp. 1–36, 2022.

[37] M. A. Jeusfeld, "Evaluating DeepTelos for ConceptBase: A Contribution to the MULTI Process Challenge," *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, vol. 17, 2022.

[38] U. Frank and T. Clark, "Multi-Level Design of Process-Oriented Enterprise Information Systems," *Enterprise Modeling and Information Systems Engineering (EMISAJ)*, vol. 10, pp. 1–50, 2022.