Type-Safety in EA Model Analysis

Thomas Reschenhofer, Ivan Monahov, Florian Matthes
Technische Universität München
Munich, Germany
Email: {reschenh|monahov|matthes}@in.tum.de

***Abstract*—In this paper, we first describe the tension between type-safety and flexibility in EA analysis tools. We then present a web-based system that combines the benefits of static typing with the flexibility of a dynamic and collaborative meta-modeling platform.**

In particular, we describe the underlying meta-model, the syntax and semantics of the expression language, and derive an associated type system, including polymorphic types, subtyping, and limited type inference. We then demonstrate the benefits of static type-safety for enterprise architects, like syntax highlighting, code completion, code navigation, and refactoring, in particular in presence of dynamic meta-model changes. The paper ends with a description of a case study using the tool for the analysis of the application landscape complexity using data from four German banks.

I. Introduction

Enterprise Architecture (EA) is defined as "the fundamental and holistic organization of an enterprise embodied in its components, relations and its environment" [1]. However, due to the increasing complexity of EAs [2], the alignment of an enterprise's information technology (IT) to its business becomes a challenging task. Moreover, this task becomes even harder when also taking into account the increasingly turbulent business environment due to influencing factors like technology innovation, market changes, and legal aspects [3]. To cope with both the EA's complexity and the dynamics of its environment, Enterprise Architecture Management (EAM) involves the holistic planning, development, and controlling of an EA and its evolution [4]. By ensuring the EA's flexibility, efficiency, and transparency, EAM is a method for achieving efficient business IT alignment [5] and hence for improving an enterprise's overall business performance [3].

Various EAM approaches (e.g., TOGAF [6], ArchiMate [7] and the Zachman Framework [8]) understand the EA as a holistic and descriptive model of the enterprise. In the remainder of this paper, we refer to this descriptive model as the (qualitative) EA model. Prevalent EAM tools implementing this model-based approaches support an adaptable Meta-model, which facilitates the definition of organization-specific EA models [9]. Furthermore, an agile and collaborative EAM approach [5] addresses the challenge of a turbulent business environment by enabling an iterative and collaborative design and management of the EA. In this way, multiple EAM stakeholders are contributing to an incremental development of EA models and meta models, facilitating the immediate and stakeholder-driven adaption of the EA on changes of the economic, technical, and regulatory environment [4].

However, due to the size and complexity of EAs, their systematic controlling as part of the EAM discipline is not possible solely with qualitative EA analysis [3], [10], [11], but also with quantitative EA analysis. Therefore, metrics are required to provide a reliable assessment of the achievement of predefined EAM goals, e.g., ensuring compliance, increasing homogeneity, and reducing operating costs [12]. In model-based EAM approaches, the metrics are defined by a proper language capable of defining queries and calculations based on the qualitative EA model [13]. Thereby, the metrics form the quantitative model of the EA. However, the design question whether the language for defining metrics is statically type-safe [14] or not—and thus supposedly more flexible—has a great impact onto the actual behavior of an analysis tool, e.g., regarding its robustness to changes of the underlying meta model. In this context, static type-safety means that an EA metric's static semantics [15] is validated at compile-time. In this paper, we highlight the benefits as well as challenges of static type-safety in the context of EA analysis.

The remainder of this paper is structured as follows: In Section II we describe a dynamic and collaborative approach to meta modeling. Based on that, we discuss the expressiveness required for EA analysis and subsequently derive an associated type system in Section III. We then present the prototypical implementation and benefits of a web-based system combining the benefits of static type-safety with the flexibility of iterative and collaborative meta modeling (c.f. Section IV). In Section V, our approach is evaluated by conducting a case study, whereas in Section VII we summarize the paper and give an outlook to future research topics.

II. The Underlying Meta Model

As motivated in Section I, in model-based EAM the metrics are based on the qualitative EA model. Specifically, metrics are defined as queries and calculations referring to EA model elements, e.g., types, attributes, and relations [13] (c.f. Figure 1). Therefore, in order to compare a statically type-safe approach to EA analysis with a rather flexible and unsafe one, and in particular to outline the benefits of the former one, we present a concrete meta model-based approach to EAM, namely Wiki4EAM [16].

Wiki4EAM—a wiki-based approach to EAM [17]—addresses the mismatch between existing unstructured information in enterprises and rigid information structures of prevalent EAM tools, whereas instances of the EA model are represented as wiki pages. In this approach, an initially unstructured EA model is incrementally and collaboratively enriched with structure (types, attributes, and relations) allowing the emergent development of qualitative EA models. The emergent structuring of wiki pages is supported by the so-called Hybrid Wikis [16]. The Hybrid Wiki model in Figure 2

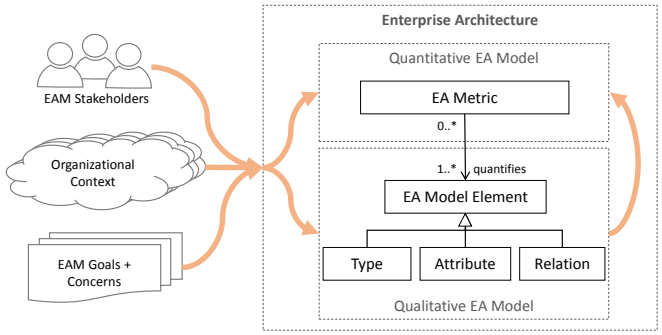

Fig. 1. EAM stakeholders, the organizational context, as well as EAM goals and concerns have an impact onto the EA [4]. Moreover, EA metrics forming the quantitative EA model are based on the the qualitative model consisting of EA model elements, i.e., types, attributes, and relations.

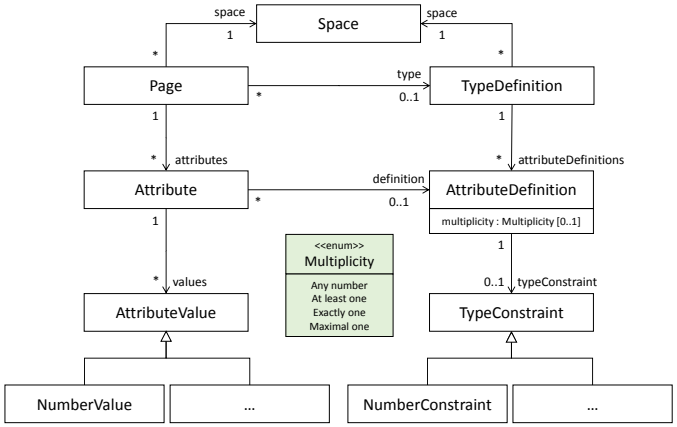

Fig. 2. The Hybrid Wiki data model by Matthes et al. [18].

facilitates the incremental definition of the data model's types (*TypeDefinition*), which in turn may have several attribute definitions (*AttributeDefinition*) with certain multiplicities and attribute types (*TypeConstraint*). The types and hence the imposed schema can be applied to wiki pages (*Page*, representing instances of the data model) at run-time, leading to an emergent structuring of initially unstructured content. Additionally, both the instances and schema elements of the Hybrid Wiki model are logically structured [15] into workspaces (*Space*).

Therefore, the elements of the Hybrid Wiki model serve as a foundation for the definition of metrics for quantitative EA analysis, whereas in particular the aspect of collaborative and iterative meta modeling is a major challenge for the quantitative model's type-safety.

III. The Untyped Core Expression Language

Based on the underlying meta model presented in Section II, we firstly motivate the expressiveness a language has to provide in order to define best-practice metrics [12]. Thereafter, we describe a proper untyped core expression language, and subsequently derive challenges for implementing static type-safety.

As shown by Monahov et al. [19], an expression language for the definition of prevalent EA metrics has to provide a certain set of basic types, e.g., *String*, *Number*, *Boolean*, and *Sequence*. Moreover, since metrics are implemented as queries of the underlying EA model, the language has to support the following classes of operators for manipulating sequences (ordered multi-sets) of EA objects of a given type (e.g., all objects of type *Business Application*):

- **Query operators** include projection (*select*), selection (*where*), grouping (*groupby*), and sorting (*orderby*).
- **Aggregation operators** fold up all elements of a sequence to a single value, e.g., *count* and *sum*.
- **Quantifier operators** return either *true* or *false* based on a given predicate, e.g., *any* (Returns *true* if at least one element fulfills the given predicate) and *all* (Returns *true* if all elements fulfill the given predicate).
- **Set operators** produce a sequence based on the presence or absence of equivalent elements within another sequence, e.g., *except* and *intersect*.
- **Element operators** choose a certain element of a sequence, e.g., *first* and *last*
- **Partitioning operators** divide a sequence into sub sequences, e.g., *take* and *skip*

These sequence operators are based on Microsoft's *Standard Query Operators* [20].

The untyped core expression language as a functional programming language implements the aforementioned sequence operators as higher-order functions [21], i.e., functions whose parameters are also functions (e.g., predicates in case of the filter operator). In the context of higher-order functions, lambda-expressions [14], [22] are used to define anonymous (nameless) functions. The untyped core expression language uses C#'s lambda notation, whereas a list of parameters is followed by an arrow (=>) and the function's implementation, i.e., the parameters are mapped to the given function mapping:

```
(p1,p2, ...) => (<function mapping>)
```

For example, the following lambda expression defines a function with two parameters and maps these parameters to the application of an arithmetic addition:

```
(a,b) => (a.add(b))
```

Therefore, based on the meta model depicted in Figure 4 (type *Business Application* with an attribute *Function points*), a query formulated with the untyped core expression language and summing up the function points of all business applications looks like the following:

```
find "Business Application"
  .select(ba =>
      ba["Function points"].first())
  .sum()
```

In this example, the *find* operator determines all objects of the given type (*Business Application*), whereas the existence of this type cannot be checked at compile-time do to the fact that the language is not statically type-safe. Based on the sequence of objects, the *select* operator takes a lambda as parameter, which maps each business application to its function points.

Again, due to the type-unsafety of the language, the existence and attribute type of the attribute cannot be checked at compile-time. Moreover, since attributes of the underlying Hybrid Wiki model potentially have multiple values, an attribute's value in the untyped core expression language is expected to be a sequence of objects, wherefore the application of the first operator is required to gather the single value of the attribute. Finally, the *sum* operator sums up a sequence of numbers, whereas again the actual type of the source sequence cannot be determined at compile-time. As shown by Monahov et al. [19], the language enables the definition of prevalent EA metrics [12] on different layers of the EA [4] (e.g., business layer and infrastructure layer).

As mentioned throughout the last paragraph, the untyped core expression language does not allow the validation of an expression's static semantics [15]. Based on the aforementioned example, this means that a tool implementing this language is not able to check for the existence of *Business Application* or the existence and type of its attribute *Function points* at compile-time. As a consequence, the tool is not able to statically analyze an expression's semantic dependencies to elements of the quantitative and qualitative EA model, e.g., the aforementioned metric as element of the quantitative model is referring to the attribute *Function points* as element of the qualitative model. Therefore, if the attribute *Function points* would be renamed, the metric would become inconsistent, since the tool cannot apply an automated refactoring of the metric's definition due to the missing knowledge of the semantic dependencies between elements of the quantitative and qualitative EA model.

In order to facilitate the validation of a metric's static semantics, analysis of an expression's semantic dependencies, and automated refactoring on changes of model elements, we extend the untyped core expression language by static type-safety. Thereby, we derive a type system facilitating the implementation of a statically type-safe language, which includes the following properties:

- **Sub typing:** Inheritance [14], [21] is a common approach for facilitating the re-use of functionality. For example, in the aforementioned example the *select* operator is defined for sequences in general, but can be applied to sequences of arbitrary types of objects, i.e., the *select* operator is reusable for all types of sequences (e.g., a sequence of business processes, functional domains, or infrastructure elements).
- **Polymorphic types:** In the aforementioned example, the return type of the *select* operator depends on the type of the source sequence on the one hand, and on the return type of the lambda on the other hand. Specifically, the select operator's signature can be described by $Sequence < T > \times (T \rightarrow U) \rightarrow Sequence < U >$. However, in order to define sequence operators with type parameters (T and U), static type-safety in the context of EA analysis requires polymorphic types [14].
- **Restricted type inference:** In order to avoid the explicit annotation of types to identifiers and thus to keep expressions readable and simple, in particular in the definition of lambda parameters, restricted type inference [14] allows the implicit determination of the parameter types (e.g., in the aforementioned expression the type of the lambda's parameter can be inferred automatically by observing the source sequence's type).

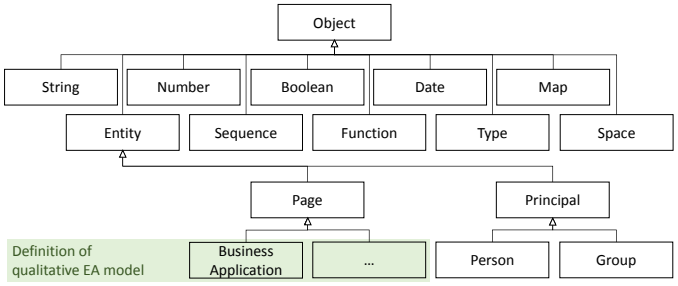

Fig. 3. The type hierarchy of the typed model-based expression language.

IV. The Typed Model-Based Expression Language

Based on the type system derived in Section III as well as the meta model described in Section II, we designed and prototypically implemented a statically type-safe language, namely the typed model-based expression language (MxL). The following section outlines the highlights of this language's type system as well as benefits of the static type-safety.

A. Type System of MxL

As described in Section III, MxL's type system has to support sub typing. Therefore, MxL organizes its types in a type hierarchy, which is depicted in Figure 3. The type hierarchy also contains the type *Function*, which is induced by MxL's nature of being a functional language.

Furthermore, a function's signature can be specified by type parametrization [14] (e.g., the type *Function<Number,Boolean>*defines a function with a parameter of type *Number* returning an object of type *Boolean*). Similarly, the type *Sequence* can be parametrized to specify the type of a sequence's elements (e.g., the type *Sequence<Number>*defines a sequence of numbers). As shown in Figure 3, all types of the qualitative EA model are specified as sub types of the type *Page* (representing instances of the EA model, e.g., *Business Application*, *Process*, or *Infrastructure Element*). Since these EA types are implemented by types of the Hybrid Wiki model (c.f. Figure 2), this part of the type hierarchy is flexible and changeable at run-time and thus facilitates the incremental development of the EA model.

Furthermore, static type-safety enables the resolution of identifiers occurring in MxL expressions and the validation of an expression's static semantics. The following MxL expression is semantically equivalent to the previously shown untyped expression, i.e., it determines all business applications (by the *find*-operation), and subsequently applies the sequence operation *sum* with a lambda as parameter, mapping each business application to its function points and summing them up:

```
find 'Business Application'
  .sum((ba:'Business Application') =>
                    ba.'Function points')
```

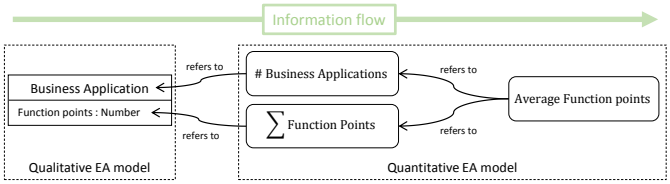

Fig. 4. This example shows the computation of the average function points of business applications. By showing the metrics used by (outgoing references) or using (incoming references) a certain metric, the information flow through the quantitative model can be identified as shown in this figure.

In this example, the type checker verifies the existence of a type *Business Application*. In this example, the apostrophes are required since the type name contains a special characters (space). Since the type of the *find*-operator's result is *Sequence<Business Application>*, the type checker ensures that the lambda's explicit type matches the expected parameter type of the sum operator (c.f., polymorphic types [14] as described in Section III), which is determined by the source sequence's type. Since this is the case, the type checker knows that the identifier *Function points* in the lambda's implementation refers to an attribute of a business application and can determine its attribute type. In the *sum* operation, the result type of the lambda has to be *Number*, which is the case in this example, wherefore the expression is semantically consistent.

By the restricted type inference of MxL, this exemplary expression can be shortened to the following one:

```
find 'Business Application'
  .sum(ba => ba.'Function points')
```

In this example, the type checker implicitly determines the type of the lambda's parameter. Furthermore, by an MxL feature called *implicit lambdas*, this aforementioned expression can be even more shortened to the following one:

```
find 'Business Application'
  .sum('Function points')
```

In this case, the type checker determines that the type of the parameter of the *sum* operation is not a function. Hence, it interprets the parameter as the implementation of a lambda, introduces an implicit and anonymous parameter, and interprets the identifier *Function points* as member of this implicit parameter. Hence, the expression is semantically equal to the previous one.

Therefore, static type-safety not only enables the validation of an expression's static semantics at compile-time, but also allows to shorten and simplify MxL expressions by using certain implicit notations.

B. Transparency of Quantitative EA Models

In order to assess the impact of changes of certain metrics, the user changing the metric has to know which other metrics are referring to the given metric. Hence, the quantitative model has to be transparent in the sense that dependencies between metrics have to be made explicit to the users of the tool.

The static type-safety of MxL allows the validation of an expression's static semantics by resolving identifiers and checking their types. Based on this, the language obtains all elements the expression refers to (e.g., functions, types, attributes, etc.), analyzes them, and uses these dependencies to maintain the quantitative model's computation graph (c.f. Figure 4). Figure 5 shows the definition of an MxL function *sumOfFunctionPoints*, which has outgoing MxL references to the sequence operation *sum*, to the type *Business Application* (through the *find*-operation), and to the attribute *Function points*. Assuming there is another function *averageFunctionPoints* using *sumOfFunctionPoints* (as depicted in Figure 4), there is also an incoming MxL reference.

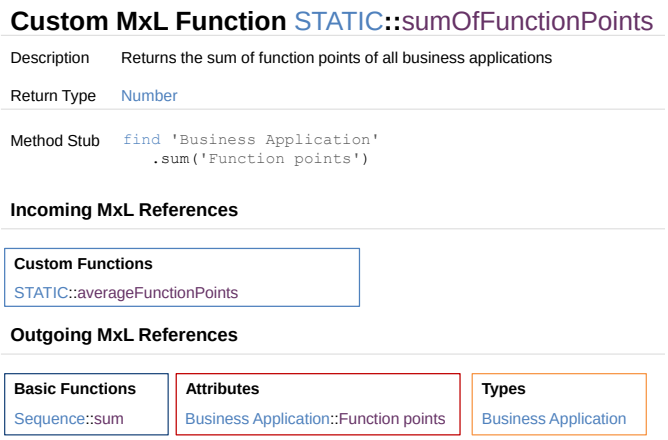

Fig. 5. The definition of the custom MxL function *sumOfFunctionPoints* (c.f. Figure 4). The language automatically determines the relationships of this function (incoming and outgoing MxL references) fostering the transparency of the quantitative model.

Therefore, when intending to change a certain metric, the tool implementing MxL shows all other metrics directly affected by this change (incoming MxL references). This supports the user in assessing the impact of a change. Moreover, since maintaining all dependencies between metrics, the tool is also able to automatically generate the visualization of the quantitative model as a computation graph similar to that in Figure 4.

C. A In-Browser Code Editor for MxL

For defining an EA metric by MxL at run time, we improved the in-browser code editor of the untyped core expression language by Monahov et al. [19]. Thereby, we also use MxL's static type-safety to enhance the code editor's features as described in the following:

- **Syntax highlighting:** The MxL in-browser code editor supports syntactic highlighting by coloring keywords, strings, etc. For example, the *find* keyword as described in Subsection IV-A is colored blue in Figure 6. Moreover, the semantic analysis by the MxL type checker also allows semantic highlighting by coloring types, attributes, or relations.
- **Code completion:** The MxL 2.0 in-browser code editor provides a list of possible identifiers based on a prefix as defined by the user. The list of proposals contains elements from both the quantitative (e.g., existing metrics) and qualitative (e.g., existing

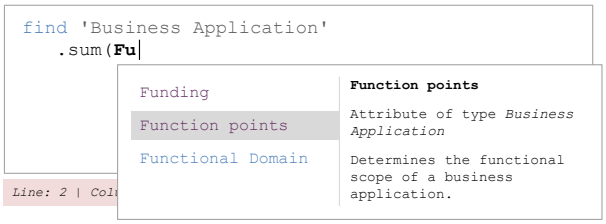

Fig. 6. The MxL in-browser code editor provides integrated development environment (IDE) services like syntax highlighting and code completion [15] with an integrated EA model documentation.

types and attributes) model. In the example shown in Figure 6, the underlying qualitative model defines a type *Functional Domain* as well as two attributes *Function points* and *Funding*, whereas all of them match the given prefix *Fun*. Furthermore, static type-safety allows further refinement of the proposals list based on the validation of the (partial) expression's static semantics.

- **Integrated documentation:** The MxL 2.0 in-browser code editor displays the documentation of selected proposals in the code completion list. For example, in Figure 6 the user selected the proposal *Function points*, wherefore the MxL code editor loads the documentation of this model element in order to provide additional information to the user.
- **Code navigation:** The MxL code-editor supports the navigation through incoming and outgoing MxL references. While the code editor does not contain click-able elements, the incoming and outgoing MxL references (c.f. Subsection IV-B) allow the navigation to MxL expressions using or used by a given one (c.f. Figure 5).
- **Error localization**: The MxL code editor highlights the origin of syntactic and semantic errors. When determining syntactic (e.g., missing closing brackets) or semantic (e.g., unknown type) errors, the MxL code editor outputs the position (line and column) of the error's origin. For example, if the qualitative model would not define a type *Business Application*, the MxL code editor in Figure 6 would display a proper error message referring to line 1 and column 5.

D. Refactoring of MxL Expressions

Since EA metrics are based on the qualitative model, the metrics have to be adapted on changes of the qualitative model in order to keep consistency, i.e., on changes of elements of the qualitative model, all metrics referring to the changing elements have to be updated.

As already discussed in Subsection IV-B, the validation of an MxL expression's static semantics by the MxL type checker allows the determination of incoming and outgoing MxL references. Therefore, for each element of both the qualitative and quantitative model, MxL knows which MxL expressions are referring to this element. As a consequence, on changes of a certain element, all expressions referring to it can be determined and thus adapted according to the change.

For example, the MxLfunction *sumOfFunctionPoints* in Figure 5 refers to an attribute *Function points*. Therefore, if implemented in the untyped core expression language (c.f. Section III), changes of this attribute would lead to inconsistencies of the function. However, due to MxL's static type-safety and the MxL dependency management as described in Subsection IV-B, changes of the attribute *Function points* implies an adaption of the MxL function *sumOfFunctionPoints*, whereas the adaption depends on the kind of the change:

- When **renaming elements**, all occurrences of the element's old name are replaced by its new name in all expressions referring to the renamed element.
- When **changing the type of an element**, the validation of the static semantics of all expressions referring to the changed element is redone. If the validation fails, the user is informed about inconsistent expressions.
- When **deleting an element**, the user can choose to either trigger a cascaded deletion of all expressions referring to this element, or to keep these expressions, which, however, would become inconsistent regarding their static semantics (if there is not a more general version of the deleted object, e.g., an equally-named attribute in a super type).
- When **creating an element**, it is checked if the new element is a specialized version of an existing one (e.g., the creation of an attribute might overwrite an equally named attribute of a super type). If there is a general version of the new element, the applicability of the specialized version in all occurrences of the general version will be checked.

Therefore, when renaming the attribute *Function points* to *Functional scope*, the implementation of the function *sumOfFunctionPoints* in Figure 5 would be automatically changed to

```
find 'Business Application'
  .sum('Functional scope')
```

V. Evaluation

Based on the prototypical implementation of MxL as described in Section IV, we conducted an analytical and observational evaluation [23] of our approach, whereas the observational evaluation took place mainly in our own research environment. The observational evaluation of the tool is discussed in the following section.

A. Evaluation Setting

In a related research activity, our research group identified metrics from literature and industry for measuring the complexity of application landscapes (AL), e.g., topology-based metrics [24], heterogeneity-focused metrics [25]. Based on these identified metrics, an integrated qualitative core model covering the concepts required for the calculation of all metrics was developed. An excerpt of this core model is depicted in Figure 7, whereas the full core model also covers types of the infrastructure layer [4]. Subsequently, four industry

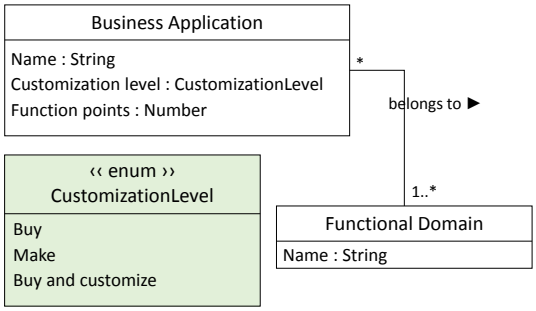

Fig. 7. An excerpt of the qualitative model of the application landscape as part of the EA forming the base for the definition of the quantitative model.

partners provided data corresponding to the developed core model, so that we were able to calculate the metrics based on industrial data. However, due to the industry partner's different approaches to EAM, there was a strong variation regarding the existence or type of certain concepts in the actual provided data sets.

Furthermore, during the research process, we continuously consolidated and adapted the qualitative and quantitative models of the application landscapes according to the regular input of our industry partners. Therefore, we applied MxL's prototype in the context of complexity metrics for application landscapes in order to use and evaluate the benefits of static type-safety.

B. Highlights of the Evaluation

For the evaluation of MxL's prototype, we firstly imported the varying data sets of our industry partners into the prototype, and subsequently implemented the quantitative model, i.e., up to 29 different metrics for each of the data sets (the actual number of implementable metrics depended on the provided data). Thereby we observed that MxL's type system as discussed in Subsection IV-A seems to be adequate for the definition of prevalent model-based EA metrics. By reimplementing all metrics of the EAM KPI Catalog [12] we can even support this claim.

As shown in Figure 7, the concept *Business Application* has an attribute *Customization level* classifying an application into *Buy*, *Make*, or *Buy and customize*. However, in order to aggregate the *Customization level*s of multiple applications, the nominal scale has to be transformed to a numerical one (e.g., one industry partners defined a transformation from *Buy* to 1, from *Make* to 3, and from *Buy and customize* to 5), which allows the calculation of an average *Customization level* of all applications belonging to a given *Functional Domain*. Since each industry partner aims for a different approach for assessing an application's *Customization level*, the question arose which metrics are actually affected by the definition of the *Customization level*. By the semantic analysis of the quantitative EA model (c.f. Subsection IV-B) the tool was able to answer this question automatically and to provide a full set of metrics (transitively) depending on the *Customization level*. Moreover, the EA tool's dependency management also facilitated the automated determination of each metric's individual information model [26] (i.e., which data is actually required for calculating a given metric), which was in particular helpful for industry partners which were evaluating which metrics are applicable in their organization-specific environments. Therefore, the transparency of the quantitative EA model was a valuable feature of the EA tool's prototype implementing MxL.

Furthermore, one of the industry partners agreed to prototypically use the EA tool for the implementation of further organization-specific EA metrics. Thereby, just based on a short and personal introduction to the tool, the involved enterprise architects immediately were able to implement custom metrics by their own, whereas they emphasized in particular the usefulness of the MxL code editor (c.f. Subsection IV-C). According to these enterprise architects, especially the tool's feature of supporting the ad-hoc definition and debugging of queries of the EA model was helpful for accomplishing this task. However, they also mentioned some weaknesses of the current state of the documentation of MxL, which forced them to go for a trial-and-error approach in some occasions.

Since we have done the observation and implementation of best-practice metrics for measuring the complexity of application landscapes and the consolidation of the aforementioned core model in parallel, the qualitative EA model was changed continuously while already having defined metrics referring to elements of the changing EA model. However, due to the automated refactoring of MxL expressions as discussed in Subsection IV-D, the semantic consistency of existing metrics was preserved.

VI. Related Work

While Sections III and IV described our own contribution to the field of quantitative EA analysis, the following section discusses related work in this field.

In general, Business Intelligence (BI) solutions [27] enable the development of extensive and scalable quantitative models by using Data Warehousing (DWH) and Online Analytical Processing (OLAP). However, today's BI tools lack either support for collaboration or proper end-user empowerment, or both [28], which prevents an agile approach to EAM as motivated in Section I of this paper.

A very common and successful class of tools—in particular in business domains—are spreadsheets (e.g., Microsoft Excel [29], Tableau [30], QlikView [31]), which facilitate the development of quantitative models (e.g., for financial reporting [32]) by end-users [33]. However, most of these spreadsheet tools lack proper support for collaboration, e.g., access control or definition of ownerships and responsibilities for certain data elements. Furthermore, as discussed in Section III, spreadsheets are lacking transparency of their design [34], wherefore spreadsheets are not suited for the iterative development of quantitative models.

Since in recent years EAM has become a commonly accepted function for achieving alignment of an enterprise's IT to its business, various tools have emerged in practice supporting EAM [35]. However, these prevalent tools are focusing the development and management of qualitative EA models [9]. Although most of them are providing certain reporting mechanisms for calculating predefined metrics, and several EAM tools even allow the definition of customized metrics by domain-specific languages (DSL), none of them

supports the iterative and or collaborative development of quantitative models [36].

Buschle et al. [10] developed an EAM tool named *Enterprise Architecture Analysis Tool* (EAAT), which is based on the *Predictive, Probabilistic Architecture Modeling Framework* [37]. EAAT is able to cope with uncertainties in the EA model due to outdated, incomplete, incorrect, or missing information. Hence, EAAT's quantitative model is based on probability distributions instead of actual values. In contrast, this paper focuses on a completely different aspect of a language for the definition of EA metrics, namely type-safety.

Moreover, there are various approaches targeting the quantitative analysis of specific aspects of an EA. For example, Iacob et al. [38] are targeting the quantitative analysis of an EA's performance. Therefore, they are introducing an approach based on the ArchiMate enterprise modeling language [7] for quantifying the performance of individual model entities to performance measures and aggregating them to an overall architecture performance. However, the language for defining metrics presented in this approach is statically unsafe with regards to the underlying quantitative EA model.

Furthermore, Clark's XMF [39] is also related to this paper. However, the language presented in this paper—MxL—is tailored to the domain of EA analysis, i.a., by implementing dependency managing, providing in-browser code editor features, and enabling automated refactoring on changes of the underlying qualitative EA model.

VII. Conclusion

The following section summarizes and concludes this paper's contribution including a critical reflection of our work. Moreover, we outline possible future research activities based on the approach presented in this paper.

As motivated in Section I, the increasing complexity of EAs requires quantitative models in order to measure and control the evolution of EAs. However, turbulent business environments induce the need for a continuous adaption of the EA and its qualitative and quantitative models. This issue is addressed by agile approaches to EAM. However, the iterative and collaborative design of the EA meta model is a major challenge for the static typing of a language for defining EA metrics.

To address this issue, we firstly introduced the underlying meta model in Section II, and subsequently described an untyped core expression language for defining EA metrics. Based on that, we derived a type system for the typed model-based expression language (c.f. Section III. Subsequently, we outlined the prototypical implementation of a web-based platform implementing MxL and discussed benefits of the language's static type-safety in Section IV, i.e., an improved web-based code editor, dependency management, as well as automatic refactoring on changes of the underlying meta-model. The prototype was evaluated by a case study in a parallel research activity for analyzing application landscape complexity using data from four German banks (c.f. Section V).

Although successfully applied in our own research group, the approach has to be studied in depth in a business environment in order to comply to the observational evaluation as defined by Hevner et al. [23]. And as already mentioned in Section V, currently the prototype is employed in a German bank, where it is used by enterprise architects for the holistic lifecycle management of organization-specific metrics. Thereby, we also want to evaluate also non-functional aspects, e.g., usability of the prototype and ease-of-learn of MxL in general.

Moreover, the evaluation of the prototype for defining and computing complexity metrics for application landscapes revealed performance issues, in particular when performing complex graph algorithms, e.g., $O(n^3)$ algorithms on big application landscapes (with applications as nodes and dependencies between them as edges). This is mainly due to MxL's simple evaluation strategy, i.e., MxL executes queries solely in main memory instead of parsing it to a query language which is executed more efficiently (e.g., SQL).

As stated in Section III, a language for the definition of best-practice metrics [12] has to support a certain minimal expressiveness. While MxL fulfills this requirement, this is also true for other language, e.g., OCL [40] and LINQ [41]—the foundations of MxL. However, since aiming a language with minimal but sufficient expressiveness as well as with regard to future research concerning the further development of the language (e.g., temporal analysis by accessing the information model's history via the language, as discussed in more detail below), we decided to design and redevelop MxL tailored to EA analysis.

Moreover, we want to tackle the following questions which emerged during our research:

- **Temporal EA analysis:** In order to understand and control an EA's evolution, the enterprise architect has to analyze the evolution of the EA metrics. Therefore, in future research, we will focus on the temporal analysis of EAs, which includes accessing the EA model's history as well as versioning of EA metrics and their results.

- **Evaluation strategies of MxL expressions:** While the current prototype executes a metric on demand (i.e., when requested by a user), most EA metrics have to be evaluated regularly according to a certain schedule (e.g., at the beginning of each month) [12]. Moreover, the validation of the expression's static semantics of a metric and the resulting analysis of dependencies between metrics and elements of the EA model would allow the recalculation of a metric on change of the underlying information model. Both evaluation strategies (*by schedule* and *on change*) will be considered in future research.

- **EA metric visualizations:** The visualization of the result of metrics reinforces the human's cognition and supports the user's understanding and interpretation of the metric's result [42]. Therefore, the employment of our prototype with proper visualization components based on the user-defined metrics (e.g., a time-series graph for visualizing an EA metric's evolution) is another focus of our future research activities.

VIII. Acknowledgement

This research has been sponsored in part by the German Federal Ministry of Education and Research (BMBF) with grant number TUM: 01IS12057.

References

[1] International Organization for Standardization, "ISO/IEC 42010:2007 Systems and software engineering – Recommended practice for architectural description of software-intensive systems," Geneva and Switzerland, 2007.

[2] C. Lucke, S. Krell, and U. Lechner, "Critical Issues in Enterprise Architecting - A Literature Review," *Americas Conference on Information Systems*, 2010.

[3] F. Ahlemann, E. Stettiner, M. Messerschmidt, and C. Legner, *Strategic Enterprise Architecture Management*. Springer-Verlag, 2012.

[4] S. Buckl, T. Dierl, F. Matthes, and C. M. Schweda, "Building Blocks for Enterprise Architecture Management Solutions," *The 2nd Practice-driven Research on Enterprise Transformation*, 2010.

[5] S. Buckl, F. Matthes, I. Monahov, S. Roth, C. Schulz, and C. M. Schweda, "Towards an Agile Design of the Enterprise Architecture Management Function," *Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, 2011.

[6] The Open Group, "TOGAF Version 9.1," 2011. [Online]. Available: http://pubs.opengroup.org/architecture/togaf9-doc/arch/

[7] M. Lankhorst, *Enterprise Architecture at Work: Modelling, Communication and Analysis*, 2nd ed. Berlin: Springer and Springer-Verlag, 2009.

[8] J. A. Zachman, "A Framework for Information Systems Architecture," *IBM Syst. J.*, 1987.

[9] F. Matthes, S. Buckl, J. Leitel, and C. M. Schweda, "Enterprise Architecture Management Tool Survey 2008," *ISIS Business Integration Special, Nomina Informations- und Marketing Services*, 2008.

[10] M. Buschle, P. Johnson, and K. Shahzad, "The Enterprise Architecture Analysis Tool–Support for the Predictive, Probabilistic Architecture Modeling Framework," *Americas Conference on Information Systems*, 2013.

[11] J. K. Lankes, "Metrics for Appilcation Landscapes," Ph.D. dissertation, Fakultät für Informatik, Technische Universität München, Germany, München, 2008.

[12] F. Matthes, I. Monahov, A. Schneider, and C. Schulz, "EAM KPI Catalog v1.0." [Online]. Available: http://wwwmatthes.in.tum.de/pages/19kw70p0u5vwv/EAM-KPI-Catalog

[13] G. Caldiera, V. R. Basili, and H. D. Rombach, "The Goal Question Metric Approach," *Encyclopedia of software engineering*, 1994.

[14] B. C. Pierce, *Types and Programming Languages*. Cambridge and Mass: MIT Press, 2002.

[15] M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. L. Kats, E. Visser, and G. Wachsmuth, *DSL Engineering-Designing, Implementing and Using Domain-Specific Languages*. dslbook. org, 2013.

[16] F. Matthes and C. Neubert, "Wiki4EAM - Using Hybrid Wikis for Enterprise Architecture Management," *International Symposium on Wikis and Open Collaboration*, 2011.

[17] S. Buckl, F. Matthes, C. Neubert, and C. M. Schweda, "A Wiki-based Approach to Enterprise Architecture Documentation and Analysis," *European Conference on Information Systems*, 2009.

[18] F. Matthes, C. Neubert, and A. Steinhoff, "Hybrid Wikis: Empowering Users to Collaboratively Structure Information," *International Conference on Software and Data Technologies*, 2011.

[19] I. Monahov, T. Reschenhofer, and F. Matthes, "Design and Prototypical Implementation of a Language Empowering Business Users to Define Key Performance Indicators for Enterprise Architecture Management," *Trends in Enterprise Architecture Research Workshop*, 2013.

[20] A. Heijlsberg and M. Torgersen, "Standard Query Operators Overview," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb397896.aspx

[21] P. van Roy and S. Haridi, *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004.

[22] A. Church, *The calculi of lambda-conversion*, ser. Annals of mathematics studies. Princeton and London: Princeton University Press and H. Milford, Oxford university press, 1941, vol. 6.

[23] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[24] R. Lagerström, C. Y. Baldwin, A. D. Maccormack, and S. Aier, "Visualizing and Measuring Enterprise Application Architecture: An Exploratory Telecom Case," *Harvard Business School Working Paper*, no. 13-103, 2013.

[25] A. Schuetz, T. Widjaja, and J. Kaiser, "Complexity in Enterprise Architectures - Conceptualization And Introduction of a Measure from a System Theoretic Perspective," *European Conference on Information Systems*, 2013.

[26] F. Matthes, I. Monahov, A. W. Schneider, and C. Schulz, "Towards a Unified and Confiugrable Structure for EA Management KPIs," *Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation*, 2012.

[27] R. L. Sallam, J. Tapadinhas, D. Yuen, and B. Hostmann, "Magic Quadrant for Business Intelligence and Analytics Platforms 2014," 2014.

[28] H. Berthold, P. Rösch, S. Zöller, F. Wortmann, A. Carenini, S. Campbell, P. Bisson, and F. Strohmaier, "An Architecture for Ad-hoc and Collaborative Business Intelligence," *EDBT/ICDT Workshops*, 2010.

[29] "Microsoft Excel - Spreadsheet Software." [Online]. Available: http://office.microsoft.com/en-us/excel/

[30] "Tableau Software: Business Intelligence and Analytics." [Online]. Available: http://www.tableausoftware.com/

[31] "QlikView - Business Intelligence for Everyone." [Online]. Available: http://www.qlik.com/

[32] R. Panko, "Facing the Problem of Spreadsheet Errors," *Decision Line*, no. 5, 2006.

[33] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, Eds., *End User Development*, 9th ed., ser. Human-Computer Interaction Series. Springer, 2006.

[34] F. Hermans, "Analyzing and Visualizing Spreadsheets," Ph.D. dissertation, Technische Universiteit Delft, 2012.

[35] R. A. Handler and C. Wilson, "Magic Quadrant for Enterprise Architecture Tools," 2012. [Online]. Available: https://www.gartner.com/doc/2219916/magic-quadrant-enterprise-architecture-tools

[36] M. Hauder, S. Roth, C. Schulz, and F. Matthes, "Current Tool Support for Metrics in Enterprise Architecture Management," *DASMA Software Metrik Kongress (Metrikon 2013)*, 2013.

[37] P. Johnson, J. Ullberg, M. Buschle, U. Franke, and K. Shahzad, "P2AMF: Predictive, Probabilistic Architecture Modeling Framework," *Enterprise Interoperability*, pp. 104–117, 2013.

[38] M.-E. Iacob and H. Jonkers, "Quantitative Analysis of Enterprise Architectures," in *Interoperability of Enterprise Software and Applications*. Springer, 2006, pp. 239–252.

[39] T. Clark, P. Sammut, and J. Willans, *Superlanguages: Developing Languages and Applications with XMF*. Ceteva, 2008.

[40] Object Management Group, "OMG Object Constraint Language (OCL)," 2012. [Online]. Available: http://www.omg.org/spec/UML/2.4.1

[41] P. Pialorsi and M. Russo, *Programming Microsoft LINQ in Microsoft .NET Framework 4*. Microsoft Press, 2010.

[42] N. Gershon and W. Page, "What Storytelling Can Do for Information Visualization," *Communications of the ACM*, vol. 44, no. 8, pp. 31–37, 2001.