Design and prototypical implementation of a language empowering business users to define Key Performance Indicators for Enterprise Architecture Management

Ivan Monahov, Thomas Reschenhofer, Florian Matthes
Chair for Informatics 19 (sebis)
Technische Universität München (TUM)
Boltzmannstr. 3, 85748 Garching bei München, Germany
Email: {ivan.monahov, reschenh, matthes}@in.tum.de

***Abstract*—To measure the achievement of predefined Enterprise Architecture Management (EAM) goals, it is essential to empower business users to define organization-specific Key Performance Indicators (KPIs). However, to support tool-based calculation of such KPIs, a formal model-based query language is required for their definition and calculation.**
In this paper we first examine existing general-purpose query languages regarding their suitability for the definition of business-user-specific KPIs in a collaborative environment. Thereafter, we justify the demand for a domain-specific query language ensuring a balance between the strengths of existing query languages and the size and purpose of the EAM domain. Based on this, we outline important design details and a prototypical implementation of such a language in a EAM tool. Finally, our language design is being evaluated by the implementation of suggested EAM KPIs from literature on the one hand, and by the development of a prototype for the use in an EU project on the other hand.

***Keywords*-Enterprise architecture management; key performance indicators; domain specific language**

I. Introduction

According to [1], Enterprise Architecture (EA) is the "fundamental organization of a system [enterprise] embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution". Hence, EA is understood as a meaningful picture of the holistic structure of an organization, including all its business, organization, application, information, infrastructure and data aspects. Furthermore, an EA also covers relations between these components and the relation to the given organization-specific context [2].
Enterprise Architecture Management (EAM) is a function for managing the flexibility, efficiency and transparency in an EA by developing, implementing and controlling those components and their relations [3].
Since the complexity of an enterprise's Information Technology (IT) has grown immensely during the last decades, tool support becomes more and more important to support EAM adequately [4]. Prevalent EAM tools [5] provide methods for gathering the EA model's data, modeling techniques for the EA and guidelines for its visualization [6], [7], [8].
However, the mismatch between unstructured information sources in EAM (e.g. spreadsheets, slides, text documents), and the rigid information structures as well as collaboration mechanisms provided by these EAM tools leads to a major problem in EAM. To address this mismatch, our research group introduced the model-based wiki approach *Hybrid Wiki* and provided a reference implementation with our research EAM tool *Tricia* [9]. More precisely, our tool is employed as an emergent and collaborative information management system, allowing an incremental enrichment of wiki pages with structure (e.g. attributes, types, integrity rules).
As stated by [10], the development of an EA becomes more and more difficult by its increasing complexity. Hence, organizations can use Key Performance Indicators (KPI) [11], [12] to quantify certain EA characteristics. These KPIs allow to qualitatively assess the EA and to measure whether or not certain EAM goals are being achieved.
An enterprise architect not only has to deal with the enterprise's architecture, but also with its dynamics, i.e. system's behavior [10]. Hence, the quantification of an EA has to cover the system's structural and behavioral aspects.
In practice, Microsoft's Excel [13] is the *de facto* standard to perform calculations based on structured data by business users [14]. However, in the context of calculations in a collaborative environment, Excel suffers from major drawbacks regarding its scalability, collaboration support, and data modeling capabilities (relations between objects). In the following sections we provide a detailed overview of Excel limitations and presents general purpose query languages important for the design of our solution.
Our aim is the integration of a quantification mechanism into a EAM tool to support the definition and computation of KPIs by business users in the EAM domain. This requires a domain-specific language (DSL) [15] to express computations and thus the formal definition of KPIs [16]. Such a DSL has to be able to define queries on the EA model and aggregations of the obtained data. In addition, it has to provide basic arithmetic and logical operations (e.g. addition, averaging, comparison).
Consequently, our research interest is dedicated to the following questions: *How can business-users define and*

compute KPIs in a collaborative system, *how to design and embed these KPIs in the UI of an EAM tool*, and *how to embed these KPIs in the system architecture of a tool*.
The remainder of this paper is structured as follows: The following Section II motivates the design of a DSL enabling the formal definition of KPIs. Thereafter, Section III describes and justifies the design of such a DSL, whereas Section IV covers its prototypical implementation in an EAM tool. Then, Section V provides an evaluation of our solution. Finally, Section VI concludes the paper and outlines future research topics.

II. Motivation

This section motivates our research aim to rapidly design and implement a new DSL for an EAM tool integrating the benefits of common general purpose query languages (e.g. SQL, OCL, LINQ) on the one hand, and avoiding the major drawbacks of Microsoft's Excel on the other hand.

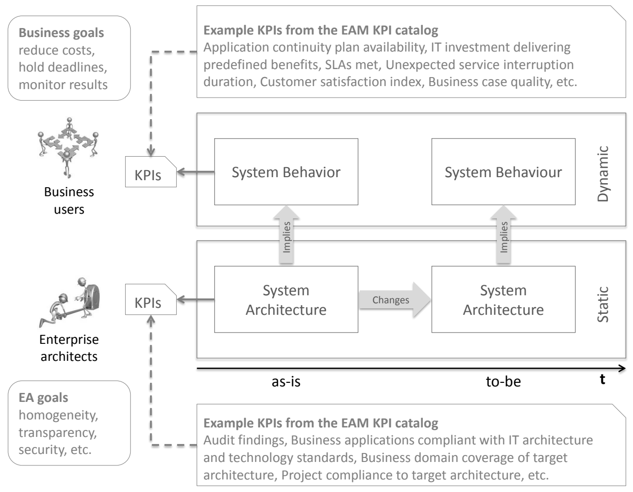

Business goals
reduce costs, hold deadlines, monitor results
Example KPIs from the EAM KPI catalog
Application continuity plan availability, IT investment delivering predefined benefits, SLAs met, Unexpected service interruption duration, Customer satisfaction index, Business case quality, etc.
Business users
KPIs
System Behavior
System Behaviour
Dynamic
implies
implies
Enterprise architects
KPIs
System Architecture
Changes
System Architecture
Static
as-is
to-be
t
EA goals
homogeneity, transparency, security, etc.
Example KPIs from the EAM KPI catalog
Audit findings, Business applications compliant with IT architecture and technology standards, Business domain coverage of target architecture, Project compliance to target architecture, etc.

Figure 1. Enterprise architects & business users can use KPIs to quantify behavioral and/or structural aspects of the EA

A. Enterprise as a dynamic system

As stated by Berg-Cross [17], an enterprise is a system, consisting of its structure and its behavior. The system structure is the holistic composition of its elements, whereas the system's behavior refers to the system's variables, their functions or relationships.
According to Forrester [18], a system's behavior arises from its structure. However, understanding the relation between the system's structure and its behavior is very difficult [19]. Furthermore, enterprises are dynamic systems, i.e. they undergo changes over time, which makes it even harder to relate the system's structure to its behavior [17]. As a consequence, to predict an enterprise's behavior over time, the enterprise architect has to understand the dynamics of the system's structure (EA) and behavior to adequately adapt the enterprise's model to ensure alignment, integration and agility of the EA.
To facilitate understanding of EA dynamics, KPIs can be used to quantify the structure and the behavior of the system. Moreover, as shown in Figure 1, these two types of KPIs may be used and managed by a multitude of users having different viewpoints onto the architecture. Thereby, the system's dynamics can be identified by the responsible users, allowing them to respond immediately to changes.

B. Tool-supported KPI computation

In practice, Microsoft's Excel is the *de facto* standard for business users to define and compute KPIs [20]. Important advantages of its spreadsheets are their interactive and easy-to-use interface as well as their flexibility [14]. Business users can start entering data and formulas before thinking about the design of the spreadsheet. However, as stated by Hermans [14], the missing relation between a spreadsheet and its underlying design is one of its major problems. Another problem is its limited modeling capabilities in the sense of insufficient management of relationships and their constraints, which has already been subject of research of many authors [21], [22].
To collaboratively manage a spreadsheet, business users often combine interrelated worksheets in a single spreadsheet. However, according to Hermans [14], these inter-worksheet connections make it hard for business users to understand the data flow and the dynamics in the spreadsheet. Moreover, Excel does support neither versioning nor historicization of its data. Hence, Excel is not suitable for computations in a collaborative environment, since the traceability of changes of the environment is essential.

C. Towards an appropriate DSL

To the understanding of the authors, an authorized user wants to specify a function to define a KPI. Hence, an appropriate domain-specific language (DSL) is required, which should be able to

- access context information (e.g. attributes, relations, current time, current user),
- query the EA model, and
- define operations on the gathered data (e.g. aggregations, arithmetic operations).

The idea of querying models is not new. In recent years many general purpose query languages have been developed and employed by academia and industry. To our understanding, following well-known general-purpose query languages are relevant in the context of business-user-oriented KPIs in a collaborative environment:

- **Structured Query Language (SQL):** The standard language for querying relational databases is SQL. In the last decades, SQL was the subject in research and practice, thus it is well understood and widely used.

Routed back in the relational algebra, SQL supports a sufficient set of operators (selection, projection, aggregation), which is an essential requirement for query languages [23]. However, with the focus on defining business-user-oriented KPIs in a collaborative environment, SQL suffers from some essential drawbacks [24], e.g. its monolithic nature, the lack of support for nested data model, an intransparent and insufficient support for user-defined functionality (UDF), and its static schema.

- **Object Constraint Language (OCL):** OCL [25] is part of the Unified Modeling Language (UML) and able to specify constraints and queries on UML models and is widely accepted and employed in both - academia and industry. However, as stated by Mandel and Cengarle [26], OCL does not form an adequate query language since it is lacking expressiveness to define all operations of a relational algebra. Hence, as concluded by Akehurst [27], the use of OCL as an effective query language requires specific extensions of the language. Moreover, since the original purpose of OCL was the definition of formal constraints in UML models, its expressiveness goes far beyond the definition of queries.
- **Language Integrated Query (LINQ):** LINQ [28] is Microsoft's approach to uniformly access different types of data sources (e.g. relational databases, XML structures), whereas in fact a query expressed in LINQ is translated to the query language of the source (e.g. SQL, xQuery). Furthermore, LINQ defines a set of general purpose *Standard Query Operators* [29], e.g. *where* (selection), *select* (projection), and *orderby*. However, as described in the next section, we require only a limited subset of operators of this general-purpose query language, relevant for the domain of EAM.

Our goal is the definition and evaluation of a minimal domain-specific query language, providing a sufficient expressiveness (minimal set of query, aggregation, and arithmetic operators), appropriate readability for business users, and optimizability. Its design is explained in detail in the next section.

III. Language Design

As motivated above, this section covers the design of a domain-specific query language.

A. Basic language design

The design of a DSL [15] is always implied by the language's domain, which is in this case the definition and calculation of KPIs by business users in a collaborative environment. Thus, the following fundamental language paradigms and properties are implied:

1) Semi-declarative syntax / Functional language: As stated by Sauer and Härder [24], a semi-declarative syntax (as supported by OCL and LINQ) allows more flexible queries than a purely declarative one by allowing the explicit application of certain query operators (e.g. projection, selection, ordering, etc.). Therefore, the order of the application of these query operators is not fixed and may be adapted to the user's needs.

A functional language supports these semi-declarative queries by providing the query operators as higher-order functions [30], i.e. our language handles functions as *first class objects* [31], so that they are usable as parameter values in other functions. This technique is especially useful for query operators, e.g. the selection operator will take a function, which will be applied to each element of a source sequence and determine, if the element should remain in the sequence, or not (see Section III-B3).

2) Object-orientation: Object orientation [30], [32] is implicated by the EAM tool's representation of EA elements as entities with attributes and references to other objects. For example, an instance of type *Project* may have an attribute *Start date* as well as a reference *Members* to instances of type *Employee*. Therefore an object-oriented language (e.g. OCL, LINQ) provides a convenient and intuitive access to EA elements, their attributes, related objects and also their methods.

3) Reflection: Reflection means, that a system is able to inspect its own state at runtime [30], e.g. an object can determine its available attributes or methods. Since Tricia allows schematic changes of the EA model at runtime, reflection allows to examine the EA model's schema at runtime.

4) Strict evaluation: Strict evaluation means, that function arguments are evaluated before the function itself [30], or in more general, before a value is bound to a name, it is evaluated. In LINQ, queries are evaluated non-strictly (lazy), i.e. the query is performed when the first element of the result is needed, which enables the possibility of query optimizations until it is performed. However, non-strict evaluation makes it very hard for business users to understand the query process, e.g. when a query is performed. Hence, we decided to design our language to be strict.

5) Basic data types: The language provides a set of basic types, which are listed in Table I.

6) Dynamic type system: The type system of a language may be either static or dynamic (or a special mix of both) [32], [30]. In a statically typed language, the type of an object is known at compile time, whereas in a dynamically typed language, the type of an object is known at runtime, e.g., LINQ.

We design our language to be dynamically typed, because the type of an object, especially the type of EA elements, is non-rigid [8], i.e. the type of an object can change at runtime (EA schema can be changed at runtime).

7) Dynamic binding: If a function is invoked on an object, the dynamic dispatch mechanism, which implements

	Name	Description	Examples
Simple data types	Object	Each element of the EA is of type *Object*.	
	String	Each character sequence encapsulated in quotation marks is a value of type *String*	$"HelloWorld", "1.23"$
	Number	Represents both integers and decimals. Also a string representing a number is of type *Number*	$1.23, -4.56, "1.23"$
	Boolean	*true* and *false*, but also its string representations are values of type *Boolean*	$true, "false", "yes"$
	Date	A string value representing a date is also a value of type *Date*	$"01.01.2000"$
Constructor data types	Sequence	A sequence of values, written as *[element1, element2, ...]*. In contrast to the mathematical term "set" [33], in sequences: • order matters, e.g. $[1, 2]$ is not equal to $[2, 1]$ • duplicates are allowed, e.g. $[1, 2]$ is not equal to $[1, 1, 2]$	$[1.0, "HelloWorld", true]$
	Map	A fixed collection of key-value-pairs. The notation is similar to the JavaScript Object Notation (JSON), i.e. *{key1 : value1, key2 : value2, ...}*	$\{"name" : "Joe", "age" : 40\}$
	Function	Because the language allows higher-order functions (see Section III-A1), there are objects of type *Function*. Anonymous functions can be written as *? (param1, param2, ...) (any expression)*	$?(a, b)\ (a.add(b))$
	Entity	An entity is a complex object, i.e. an object with attributes and/or references to other objects. In the EAM tool, each EAM element is represented as such an entity, i.e. as object of type *Entity*	

Table I
SIMPLE DATA TYPES AND CONSTRUCTOR DATA TYPES IN OUR LANGUAGE

dynamic binding, uses runtime type information to look up the proper function [34]. Since the most EAM tools provide either no inheritance or just limited subtype relation capabilities, dynamic binding enhances the reuse of functionality even in these tools.

B. Basic functions in our language

To give a formal definition of a KPI, our language has to provide a set of basic functions or operators, which can be classified into *arithmetic operators*, *comparison and logical operators*, *query operators* and *aggregation operators*.

1) Arithmetic operators: To support computations, our language provides operators for the addition, subtraction, multiplication and division of numbers, as well as operators for the integer division and modulo.

2) Comparison and logical operators: In addition to arithmetic operators, our language provides common comparison operators, e.g. *isNull*, *equals*, *greaterThan*, and *lessThanOrEqualTo*. All these operators return a boolean value, i.e. either *true* or *false*.
Furthermore, our language provides the logical operators *and*, *or* and *not* to combine and invert the results of the comparison operators.

3) Query operators: A major purpose of our language is to express queries on the EA model. As stated in Section II, there are already many well-known query languages, whereas each of them supports a similar set of query operators. However, since Microsoft's LINQ was designed as an uniform access to different types of data sources, Microsoft already worked out a common set of operators supported by a multitude of query languages, which they named the *Standard Query Operators* [29]. Consequently, the set of our language's query operators (listed in Table II) is based on Microsoft's Standard Query Operators).
Based on a sequence of all elements of a certain type (e.g. all instances of type *Project*), query operators are able to determine a certain subset of these objects (e.g. all Projects started before a certain date). Additionally, these operators empower the language to perform multiple types of joins.

4) Aggregation operators: In contrast to query operators, aggregation operators may not return sequences, i.e. all elements of a sequence are folded up to a single value. The aggregation operators supported by our language are listed in Table III, whereas they are also inspired by Microsoft's Standard Query Operators.

The following Section IV highlights some relevant aspects of our language's integration in an EAM tool.

IV. PROTOTYPICAL IMPLEMENTATION

Based on the design decisions from the previous section, we focus now on a prototypical implementation of our language in an EAM tool. Although there are many prevalent EAM tools [4], we decided to implement it in our research EAM tool *Tricia* [35], because

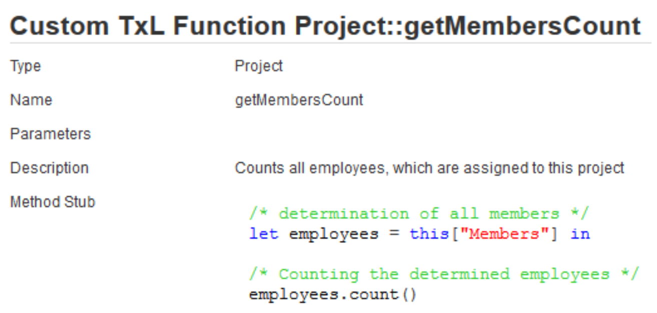
- it supports modeling both an EA's structure and its behavior (as motivated in Section II-A),
- it supports several collaboration mechanisms, e.g. versioning, historicization, collaborative editing, modeling

Name	**Parameters**	**Returns**	**Description**
where	$source : T[\,]$ $pred : T \rightarrow bool$	$T[\,]$	The *where* operator filters the source list on base of the given predicate, which is applied on each element. If the predicate evaluates to $true$, the element remains in the resulting sequence, otherwise it will be removed.
select	$source : T[\,]$ $map : T \rightarrow U$	$U[\,]$	The *select* operator applies the given *map* function to each element of the source sequence and returns a sequence containing the results of each individual map application.
selectMany	$source : T[\,]$ $map : T \rightarrow U[\,]$	$U[\,]$	The *selectMany* operator is similar to the *Select* operator, but the *map* function, which will be applied on each element of the source sequence, returns a sequence of elements. The concatenation of all sequences forms the result of this operator.
take	$source : T[\,]$ $n : int$	$T[\,]$	The *take* operator returns a sequence with the *n* first elements of the source sequence.
skip	$source : T[\,]$ $n : int$	$T[\,]$	The *skip* operator returns a sequence with all elements of the source sequence, except the first *n* ones.
concat	$first : T[\,]$ $second : T[\,]$	$T[\,]$	The *concat* operator concatenates the first sequence with the second one, i.e. the resulting sequence contains all elements of the first sequence, followed by all elements of the second one.
orderby	$source : T[\,]$ $keySel : T \rightarrow K$ $descending : bool$	$T[\,]$	The *orderby* operator sorts the source sequence by the *keySelector*, whereas a natural order will be applied. The *descending* parameter determines, if the elements should be ordered ascending ("lowest first") or descending ("biggest first").
groupby	$source : T[\,]$ $keySel : T \rightarrow K$ $map : T[\,] \rightarrow U$	$\{K, U\}[\,]$	The *groupby* operator determines a key for each element of the source sequence by applying the *keySelector*. All elements with the same key form a new sequence, on which the *map* function will be applied. The result of the *GroupBy* operator is a sequence of objects, whereas each object contains a key and the related result of the optional *map* function.
distinct	$source : T[\,]$	$T[\,]$	The *distinct* operator removes all duplicates of the source sequence. The equality of elements will be tested by there *equals* operator.
intersect	$first : T[\,]$ $second : T[\,]$	$T[\,]$	The *intersect* operator calculates the intersection of the first and the second sequence, i.e. this operator returns a distinct sequence with all elements, which are containing in the first and the second sequence.
except	$first : T[\,]$ $second : T[\,]$	$T[\,]$	The *except* operator returns a distinct sequence with all elements, which are containing in the first, but not in the second sequence.

Table II
OUR LANGUAGE'S QUERY OPERATORS BASED ON *Microsoft's Standard Query Operators* [29]

Name	**Parameters**	**Returns**	**Description**
count	$source : T[\,]$	$Number$	The *count* operator returns the number of elements in the source sequence.
sum	$source : T[\,]$	$Number$	The *sum* operator returns the sum of all numerical values in the source sequence.
min	$source : T[\,]$	$Number$	The *min* operator returns the minimum of all numerical values in the source sequence.
max	$source : T[\,]$	$Number$	The *max* operator returns the maximum of all numerical values in the source sequence.
average	$source : T[\,]$	$Number$	The *average* operator calculates the average of all numerical values in the source sequence.
first	$source : T[\,]$	T	The *first* operator returns the first element of the source sequence, if the sequence has at least 1 element, otherwise the operator throws an exception.
firstOrNull	$source : T[\,]$	T	The *firstOrNull* operator returns the first element of the source sequence, if the sequence has at least 1 element, otherwise the operator returns $null$.
single	$source : T[\,]$	T	The *single* operator returns the first element of the source sequence, if the sequence has exactly 1 element, otherwise the operator throws an exception.
aggregate	$source : T[\,]$ $f : (T, U) \rightarrow U$ $seed : U$	U	The *aggregate* operator provides a mechanism to define a custom aggregation. The function *func* will be applied on each element, whereas the second parameter is the result of the application on the previous element, or in the case of the first element, the *seed* value. The result of the final application will be returned as result of the aggregate operator.

Table III
THE AGGREGATION OPERATORS OF OUR LANGUAGE BASED ON *Microsoft's Standard Query Operators* [29]



Custom TxL Function Project::getMembersCount

Type	Project
Name	getMembersCount
Parameters	
Description	Counts all employees, which are assigned to this project
Method Stub	

```
/* determination of all members */
let employees = this["Members"] in

/* Counting the determined employees */
employees.count()
```

Figure 2. A TxL custom function, which determines the number of a project's members. Moreover, this function is using the name binding construct *let* to bind the list of members to the name *employees*.

on runtime as well as the management of relations between model elements, and

- we can access and modify its source code.

We named our concrete implementation *Tricia Expression Language* (TxL), and outline the most important implementation details of our prototype in the next paragraphs.
Tricia is a plugin-based software. The core functionality of Tricia is contained in the plugin *platform*. The integration of TxL in Tricia was done by adding a new plugin named *script*, which contains the three logical components

- *Script core*,
- *Embedded script*, and
- *Derived attributes*.

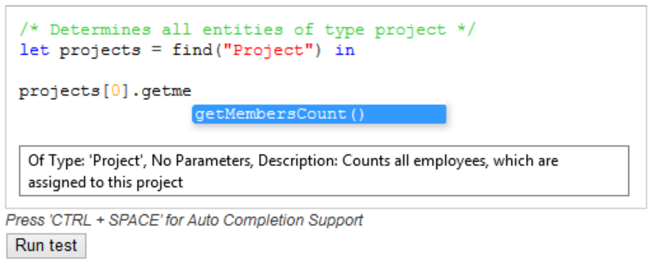
A. Script core

The component *script core* covers the functionality to scan, parse and evaluate a TxL expression, i.e. it contains the TxL interpreter as well as the TxL evaluation environment. Furthermore, the *script core* component includes all operators covered by Section III-B.

1) Extensibility: One of the major drawbacks of SQL mentioned in Section II is its intransparent support for user-defined functionality. Addressing this drawback, TxL allows the definition of *Basic functions* (implemented in the tool's implementation language *Java*) as well as the definition of *Custom functions* (implemented in TxL).
While *Basic functions* provide access to tool related and contextual information, *Custom functions* can be defined at runtime, allowing the user to encapsulate reusable functionality. Figure 2 depicts an exemplary custom function, which could be invoked as follows:

```
anyProject.getMembersCount()
```

2) User interface enhancements: The *script core* component supports in-browser code editing and provides syntax highlighting and auto completion (c.f. Figure 3). Our code editor uses the JavaScript component CodeMirror [36].



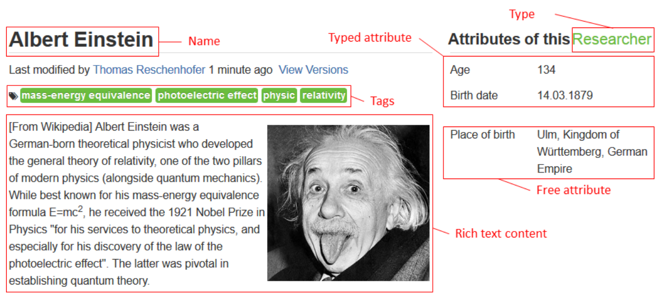
```
/* Determines all entities of type project */
let projects = find("Project") in

projects[0].getme
                getMembersCount()
```

Of Type: 'Project', No Parameters, Description: Counts all employees, which are assigned to this project

Press 'CTRL + SPACE' for Auto Completion Support

Figure 3. TxL code editor in action.



Albert Einstein

Last modified by Thomas Reschenhofer 1 minute ago View Versions

mass-energy equivalence photoelectric effect physic relativity

[From Wikipedia] Albert Einstein was a German-born theoretical physicist who developed the general theory of relativity, one of the two pillars of modern physics (alongside quantum mechanics). While best known for his mass-energy equivalence formula E=mc^2, he received the 1921 Nobel Prize in Physics "for his services to theoretical physics, and especially for his discovery of the law of the photoelectric effect". The latter was pivotal in establishing quantum theory.

Attributes of this Researcher

Age	134
Birth date	14.03.1879
Place of birth	Ulm, Kingdom of Württemberg, German Empire

Figure 4. A Tricia page and its basic parts

B. Embedded script

A page in Tricia is a hybrid entity [35], i.e. it contains structured as well as unstructured information. The unstructured part is a regular web page (HTML markup), whereas the structured part is a set of attribute-value pairs as well as type annotations. Figure 4 depicts a typical Tricia page and explains its parts. If a TxL expression is evaluated in the context of a page, this page can be accessed in the expression by the *this* keyword. The *embedded script* component allows the embedding of single TxL expressions into the rich text content. We defined following syntax to embed TxL expressions in HTML markup:

```
$[eval()$ a TxL expression $eval]$
```

For example, the content

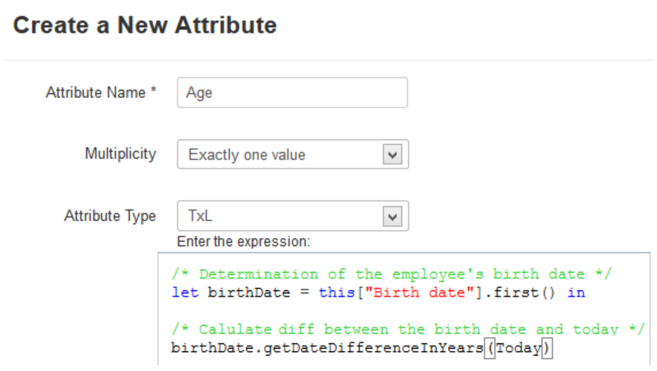
```
1 + 2 = $[eval()$ 1.0.add(2.0) $eval]$
```

would result in the following output (browser representation of the above code):

```
1 + 2 = 3
```

Moreover, these embedded TxL expressions are able to render HTML markup, allowing the user to define custom visualizations based on the evaluated value of TxL expressions. E.g., a user may embed the following TxL expression to show different images depending on the outcome of a TxL function (by using a conditional construct of the form *<boolean condition> ? <evaluate if true> : <evaluate if false>*):

```
isRainy()
```



Create a New Attribute

Attribute Name *	Age
Multiplicity	Exactly one value
Attribute Type	TxL

Enter the expression:

```
/* Determination of the employee's birth date */
let birthDate = this["Birth date"].first() in

/* Calulate diff between the birth date and today */
birthDate.getDateDifferenceInYears(Today)
```

Figure 5. The definition of the *Age* attribute as an example for a derived attribute

```
? "<img src='rain.jpg'/>"
: "<img src='sun.jpg'/>"
```

C. Derived attributes

In contrast to the *embedded script* component, the *derived attributes* component extends the structured part of a page. As Figure 4 depicts, a Tricia page can have two types of attributes:

- **Typed attributes:** An attached type can define attributes, which will be suggested for each instance the type is applied to (e.g. in Figure 4, the attribute *Birth date* is specified in type *Employee*).
- **Free attributes:** Each page may have an arbitrary number of free attributes, which are not specified by any attached type (e.g. in Figure 4, *Place of birth* is a free attribute).

Regardless of whether an attribute is typed or free, it may be of type *TxL*, which allows the definition of an attribute whose value is not static, but derived from contextual information (e.g. other attributes of the current element) and/or queryable EA data from other attribute values. For example, a type *Employee* may have the attribute *Birth date* from type *Date*, but also an attribute *Age* from type *TxL* defining an expression to calculate the difference between the Birth date and the current date. Figure 5 depicts the suitable attribute definition, whereas Figure 4 shows its application on an instance of type *Employee*.

V. Evaluation of TxL

To provide an evaluation of our language design, we first ensured all suggested EAM KPIs from literature [37], [11] can be implemented by our language.
Then, we employed our prototypical implementation TxL in an EU project called *SmartNet Navigator* [38]. In this project, over 30 companies from the German textile industry participate in a collaborative innovation management process.
Although this project does not belong to the classical EAM

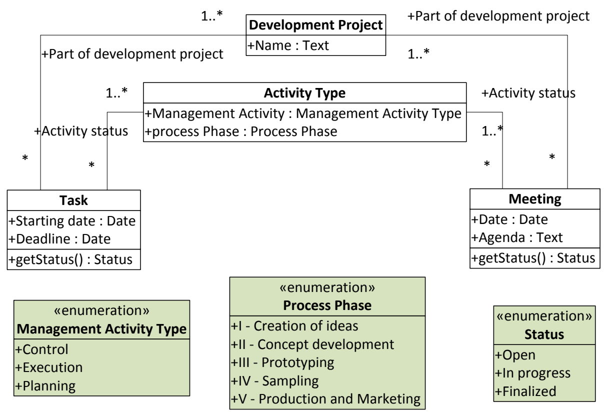


Figure 6. An excerpt of the SmartNet information model

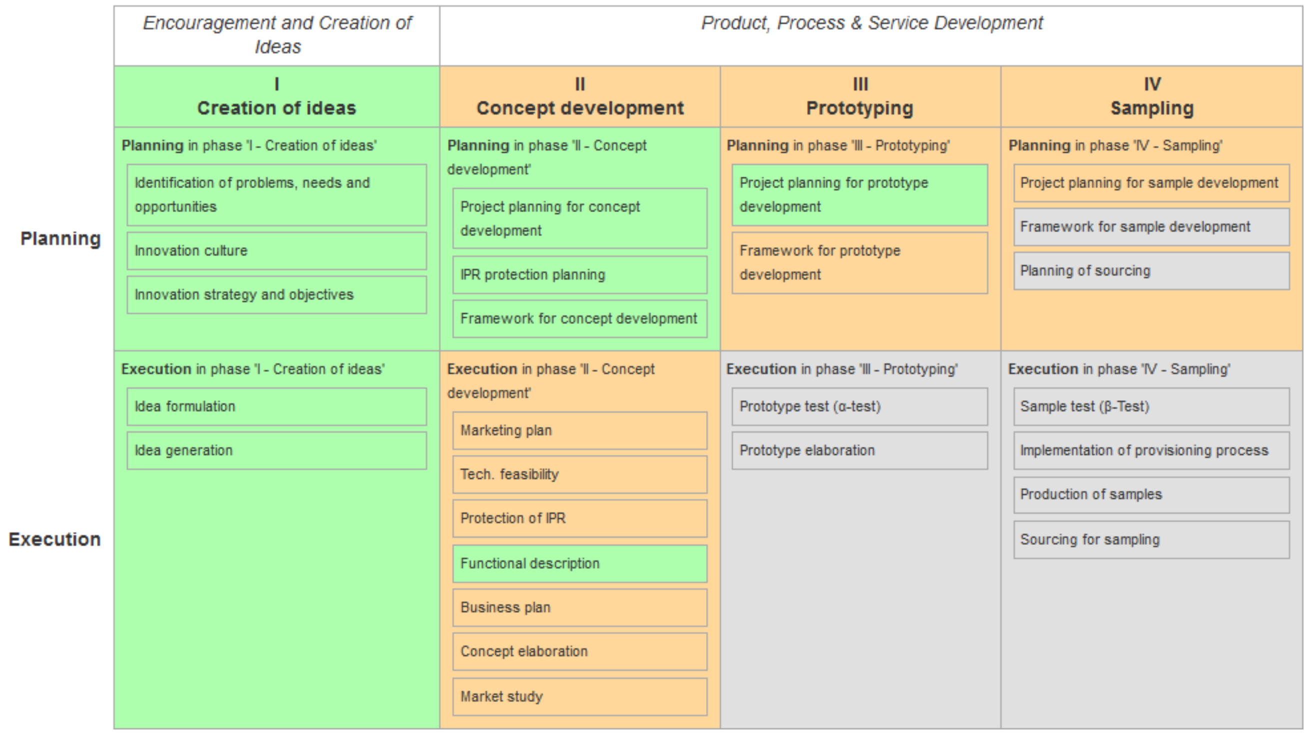
domain, it contains all relevant challenges of a typical EAM project - plenty of different stakeholders, a lot of collaborative tasks and decisions, and last but not least, information from different sources. In this context, a business-user-specific KPI expressed as visualization had been developed with our prototype in order to support decision making.

A. Description of the SmartNet Navigator

The SmartNet Navigator is the automated generation of the visualization of a project's progress. Basically, in this context, a project consists of tasks and meetings, which are assigned to several activity types (see Figure 6). Each activity type is in turn associated with a process phase as well as a management activity type.
Based on a project's tasks and meetings, the SmartNet Navigator visualizes the progress of a project, whereas the status of tasks and meetings gets stepwise aggregated to an activity type status, to a module (tuple of process phase and management activity) status, to a process phase status, and finally to the project status. Figure 7 depicts an excerpt of the SmartNet Navigator of an exemplary project. The status of an element is indicated by its color as listed in Table IV.

Color	Status	Aggregation
Grey	*Open*	If the status of each sub-element of an element is *Open*, the elements status is *Open* as well
Green	*Finalized*	If the status of each sub-element of an element is *Finalized*, the elements status is *Finalized* as well
Orange	*In progress*	If the status of an element is neither *Open* nor *Finalized*, it's *In progress*

Table IV
COLOR ENCODING AND AGGREGATIONS OF THE STATUS OF THE PROJECT, A PROCESS PHASE, A MODULE, AND AN ACTIVITY TYPE



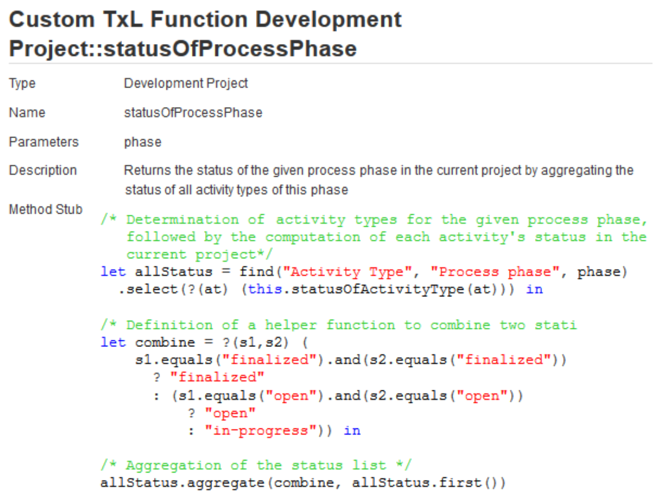
	Encouragement and Creation of Ideas	*Product, Process & Service Development*		
	I Creation of ideas	**II Concept development**	**III Prototyping**	**IV Sampling**
Planning	**Planning** in phase 'I - Creation of ideas' Identification of problems, needs and opportunities Innovation culture Innovation strategy and objectives	**Planning** in phase 'II - Concept development' Project planning for concept development IPR protection planning Framework for concept development	**Planning** in phase 'III - Prototyping' Project planning for prototype development Framework for prototype development	**Planning** in phase 'IV - Sampling' Project planning for sample development Framework for sample development Planning of sourcing
Execution	**Execution** in phase 'I - Creation of ideas' Idea formulation Idea generation	**Execution** in phase 'II - Concept development' Marketing plan Tech. feasibility Protection of IPR Functional description Business plan Concept elaboration Market study	**Execution** in phase 'III - Prototyping' Prototype test (α-test) Prototype elaboration	**Execution** in phase 'IV - Sampling' Sample test (β-Test) Implementation of provisioning process Production of samples Sourcing for sampling

Figure 7. An excerpt of the SmartNet Navigator of an exemplary project. The rows are the management activity types (e.g. "Planning"), the columns are the process phases (e.g. "Creation of Ideas"), the cells are the modules (e.g. "Planning in phase 'I - Creation of Ideas'), and the items in the cells are the activity types (e.g. "Identification of problems, needs and opportunities").

For more detailed description of the implementation and evaluation of the SmartNet Navigator in the context of this EU project we refer to the publication of Hauder et al. [39].

B. Implementation of the SmartNet Navigator

Based on the model in Figure 6 and an appropriate test data set, we defined custom TxL functions for the stepwise visualization and aggregation of a project's status. For example, the computation of a process phase's status is depicted in Figure 8, which in turn is used for the definition of the status table's header (e.g. by specifying the background color of the column). Therefore, by executing the function *statusTable* on an element of type *Development Project*, this function generates the HTML markup defining the SmartNet Navigator (see Figure 7), which can be embedded in any Tricia page. The execution of the status aggregation and visualization functions is depicted in Figure 9. Since the definition of the data model as well as the definition of custom TxL functions is done at runtime, TxL allows the definition of complex visualizations and computation at runtime, as shown by the implementation of the SmartNet use case.
To sum up, this experiment was so successful and useful for the involved business users, that a German industry company (infoAsset AG [40]) is implementing a spin-off from this prototype for industrial use.



Custom TxL Function Development Project::statusOfProcessPhase

Type	Development Project
Name	statusOfProcessPhase
Parameters	phase
Description	Returns the status of the given process phase in the current project by aggregating the status of all activity types of this phase
Method Stub	

```
/* Determination of activity types for the given process phase,
   followed by the computation of each activity's status in the
   current project*/
let allStatus = find("Activity Type", "Process phase", phase)
  .select(?(at) (this.statusOfActivityType(at))) in

/* Definition of a helper function to combine two stati
let combine = ?(s1,s2) (
    s1.equals("finalized").and(s2.equals("finalized"))
      ? "finalized"
      : (s1.equals("open").and(s2.equals("open"))
          ? "open"
          : "in-progress")) in

/* Aggregation of the status list */
allStatus.aggregate(combine, allStatus.first())
```

Figure 8. Implementation of the function for the computation of a process phase's status. A description for the *aggregation* operator can be found in Table III

VI. Summary, Conclusion and Outlook

As motivated in Section II, a collaborative environment requires the support for the definition and computation of KPIs by business users. Since Microsoft's Excel suffers from some major drawbacks in this context (e.g. missing collaboration support), we decided to integrate a quantification mechanism

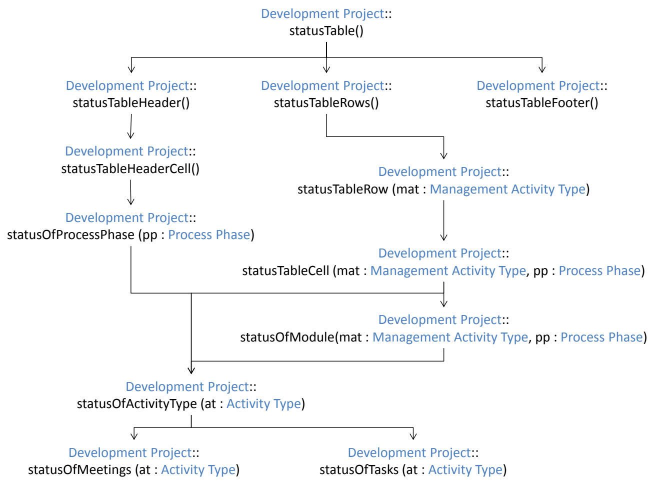


Figure 9. Depiction of the execution of the status aggregation and visualization functions of the SmartNet Navigator

into a collaborative EAM tool, which, however, requires an appropriate domain-specific query language to define and compute KPIs. Because of our aim to implement a domain-specific query language, which provides a minimal, but sufficient expressiveness, appropriate readability for business users as well as optimizability, we designed a new language instead of integrating an existing one.

The language (Section III covers its basic design as well as its supported operators) is inspired by the context of defining and computing business-user-oriented KPIs in a collaborative environment on the one hand, and by existing general-purpose languages (e.g. SQL, OCL, LINQ) on the other hand. Its prototypical implementation in our research EAM tool, covered in Section IV, enables derived attributes of entities as well as embedded expressions. Moreover, since integrated in a web-based tool, our language supports the definition of user-specific, rudimentary visualizations by the generation of HTML markup.

To evaluate the language (see Section V), we implemented the KPIs of the EAM KPI Catalog [37] as well as the SmartNet Navigator, computing and visualizing a project's status based on related tasks and meetings.

As stated in Section II, a non-collaborative tool (e.g. Excel) is not suitable for the business-user-oriented definition and computation of KPIs in a collaborative environment. However, the integration of an appropriate query mechanism in a collaborative EAM tool enables the definition and computation of KPIs. The integration of our language in an EAM tool (see Section IV) allows embedding of expressions in the rich text content of a page as well as the definition of derived attributes for types on schema level. This enables user-defined visualizations on the one hand, and extends the modeling capabilities of the EAM tool by enabling derived attributes on the other hand.

Although the evaluation of our language highlights the capability of defining, computing and visualizing KPIs in a collaborative environment, it is not yet clear, if the language is practicable for business users, which have at least some experience with Microsoft's Excel. Therefore, further evaluation has to be done in order to improve the language and its syntax, especially regarding its suitability for business users. Apart from this, further new questions emerged during our research, which should be tackled in future research:

- **Authorization concept:** An authorization concept for the DSL has to be implemented to control the use of TxL, e.g. studying the interplay between KPI computation and role-based access-protections of the underlying data entities.
- **Query analysis and optimization:** Since we think, that TxL can be employed in any Big Data scenarios, performance issues have to be considered, i.e. optimization of operator application, materialization of evaluated values, etc.
- **Time series:** To support time series over the value of a KPI, a version history of the evaluated value of a TxL expression is required.
- **Visualizations:** To visualize the results of a TxL expression representing a KPI in an appropriate way, the language has to be extended by certain visualization constructs to depict the evaluated results in user-friendly way, e.g. as a diagram or as traffic lights.
- **Further evaluation:** Both, the language and its implementation have to be evaluated in practice. For this purpose, we plan an operational use in our teaching activities and in our Wiki4EAM community [9]

References

[1] The Open Group. TOGAF Version 9.1: Section 2.2. http://pubs.opengroup.org/architecture/togaf9-doc/arch/chap02.html, October 2012.

[2] Sabine Buckl, Thomas Dierl, Florian Matthes, and Christian M. Schweda. Building Blocks for Enterprise Architecture Management Solutions. 2010.

[3] Frederic Ahlemann, Eric Stettiner, Marcus Messerschmidt, and Christine Legner. *Strategic Enterprise Architecture Management*. Springer-Verlag, 2012.

[4] Florian Matthes, Sabine Buckl, Jana Leitel, and Christian Schweda. Enterprise Architecture Management Tool Survey 2008. 2008.

[5] Robert A. Handler and Chris Wilson. Magic Quadrant for Enterprise Architecture Tools. http://imagesrv.gartner.com/media-products/pdf/reprints/ibm/external/volume4/article28.pdf, 2011.

[6] Sabine Buckl, Alexander M. Ernst, Josef Lankes, and Florian Matthes. Enterprise Architecture Management Pattern Catalog (Version 1.0, February 2008). Technical report, Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany, 2008.

[7] Sabine M. Buckl. *Developing Organization-Specific Enterprise Architecture Management Functions Using a Method Base*. Dissertation, Technische Universität München, München, 2011.

[8] Christian M. Schweda. *Development of Organization-Specific Enterprise Architecture Modeling Languages Using Building Blocks*. Dissertation, Technische Universität München, München, 2011.

[9] Florian Matthes and Christian Neubert. Wiki4eam - using hybrid wikis for enterprise architecture management. 2011.

[10] Frederik Ahlemann, Eric Stettiner, Marcus Messerschmidt, and Christine Legner. *Strategic Enterprise Architecture Management*. Springer-Verlag, 2012.

[11] Matthias Stutz. *Kennzahlen für Unternehmensarchitekturen: Entwicklung einer Methode zum Aufbau eines Kennzahlensystems für die wertorientierte Steuerung der Veränderung von Unternehmensarchitekturen*. Verlag Dr. Kovac, 2009.

[12] Josef K. Lankes. *Metrics for Appilcation Landscapes*. Dissertation, Technische Universität München, München, 2008.

[13] Microsoft Excel - Office.com. http://office.microsoft.com/en-us/excel, March 2013.

[14] Felienne Frederieke Johanna Hermans. *Analyzing and Visualizing Spreadsheets*. PhD thesis, Software Engineering Research Group, Delft University of Technology, Netherlands, 2012.

[15] Martin Fowler. *Domain Specific Languages*. Addison-Wesley Longman, 2010.

[16] Viara Popova and Alexei Sharpanskykh. Modeling organizational performance indicators. 2010.

[17] Gary Berg-Cross. Improving representation and conceptualization for enterprise architectures. *International Conference on Enterprise System Theory*, 2007.

[18] Jay W. Forrester. *Principles of Systems*. Wright-Allen Press, 1968.

[19] Alan Karl Graham. *Principles on the relationship between structure and behavior of dynamic systems*. Dissertation, Massachusetts Institute of Technology, Cambridge, 1973.

[20] Ray Panko. Facing the problem of spreadsheet errors. *Decision Line*, 37, 2006.

[21] Jacome Cunha, Joao Saraiva, and Joost Visser. From Spreadsheets to Relational Databases and Back. 2008.

[22] Gregor Engels and Martin Erwig. ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications. 2005.

[23] C. J. Date. *An Introduction to Datebase Systems*. Addison-Wesley, 8 edition, 2003.

[24] Caetano Sauer and Theo Haerder. Compilation of Query Languages into MapReduce. *Datenbank Spektrum*, 13:5–15, 2013.

[25] OMG Object Constraint Language (OCL). http://www.omg.org/spec/OCL/2.3.1/, 2012.

[26] Luis Mandel and Maria Vistoria Cengarle. On the Expressive Power of the Object Constraint Language. *FM '99 Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems*, pages 854–874, 1999.

[27] David H. Akehurst and Behzad Bordbar. On Querying UML Data Models with OCL. 2001.

[28] Don Box and Anders Heijlsberg. LINQ: .NET Language-Integrated Query. http://msdn.microsoft.com/en-us/library/bb308959.aspx, 2007.

[29] Anders Heijlsberg and Mads Torgersen. The .NET Standard Query Operators. http://msdn.microsoft.com/en-us/library/bb394939.aspx, 2007.

[30] Peter Varn Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004.

[31] Michael Scott. *Concepts of Programming Languages*. Morgan Kaufmann Publishers, 2001.

[32] Robert W. Sebesta. *Programming Language Pragmatics*. Addison Wesley, 2006.

[33] Robert R. Stoll. *Set Theory and Logic*. Dover Publications, 1979.

[34] Scott Milton and Heinz W. Schmidt. Dynamic Dispatch in Object-Oriented Languages. 1994.

[35] Florian Matthes, Christian Neubert, and Alexander Steinhoff. Hybrid wikis: Empowering users to collaboratively structure information. 2011.

[36] CodeMirror. http://codemirror.net/, October 2012.

[37] Florian Matthes, Ivan Monahov, Alexander Schneider, and Christopher Schulz. "eam kpi catalog v1.0". 2011.

[38] Heiko Matheis. SmartNet Navigator and application guidelines. *Sehenth Framework Programme*, 2013. SmartNets - The Transformation from Collaborative Knowledge Exploration Networks into Cross Sectoral and Service Oriented Integrated Value Systems.

[39] Matheus Hauder, Sascha Roth, Florian Matthes, Armin Lau, and Heiko Matheis. Supporting collaborative product development through automated interpretation of artifacts. 2013.

[40] infoAsset AG. www.infoasset.de, October 2012.