On the computation of Enterprise Architecture Managemnet KPIs - essential operators and Ecore supporting languages

Erdisa Subashi, Ivan Monahov, Christopher Schulz, Florian Matthes

Technische Universität München (TUM)
Chair for Informatics 19 (sebis)
Boltzmannstr. 3, 85748 Garching bei München, Germany
{erdisa.subashi,ivan.monahov,christopher.schulz,matthes}@in.tum.de

Abstract. Enterprise Architecture Management (EAM) strives for aligning business and IT, facilitating stakeholder communication, and fostering the continuous transformation of organizations. As in any other management discipline, EAM applies the concept of metrics, which allows for the measurement of the actual goal fulfillment. Furthermore, distinct tools support architects in efficiently gathering, processing, and disseminating information related to the current and future state of the enterprise architecture. However, given that the existing panoply of current EAM tools only offers a limited metric support, we intend to extend our java-based EAM research software with this capability. As a starting point, we first elaborate on a set of operators a language employed for the implementation of EAM metrics has to provide. We afterwards point out which of these operators are inherently included in languages simultaneously supporting Ecore as the meta model of the Eclipse Modeling Framework (EMF). We choose this well-known and open source framework for our studies, since our EAM research software is already able to export Ecore models. Lastly, we demonstrate how one specific EAM metric can be implemented in the examined languages. The gained findings will serve as an input for the future extension of our research tool with KPI functionalities.

Key words: EAM, metrics, Ecore, operators, language

1 Introduction

In the last ten years, Enterprise Architecture (EA) and its management (EAM) received considerable attention from academics, practitioners, consultants, as well as tool vendors. Thereby, EAM targets the enterprise in holistic manner and seeks to evolve the enterprise to facilitate the alignment of business and information technology (IT) as a respond to rapidly changing business environment in modern globalized enterprises. In this sense, an EA can be described as 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" as proposed by the ISO Standard 42010 [24]. According to [12] and [34], the three most important advantages offered by EAM are

– to create a holistic perspective on the enterprise, containing business and IT elements,
– to foster communication by defining a common language for stakeholders with different backgrounds, and
– to gather information from different sources to ensure a consistent decision base.

According to [13], management functions are usually described by planning, leading, organizing, and controlling dimensions. Moreover, according to the same author, the term management generally refers "to the process of assembling and using resources - human, financial, material, and information - in a goal directed manner to accomplish tasks in an organization". Thus, EAM initiatives are driven by specific EAM goals (c.f. [15]).

Key Performance Indicators, also known as KPIs, help an organization to define and measure the progress toward the achievement of these goals. A KPI can be defined as "an item of information collected at regular intervals to track the performance of a system [enterprise]" [18].

Matthes et. al. provide in [28], a collection of 52 EAM KPIs gathered from both literature and industry partners (in the remainder of the paper we refer to the EAM KPI catalog as "catalog"). These KPIs are described by employing the EAM KPI description structure proposed in [29]. Following three elements of this description structure are to be considered when computing these KPIs: *description*, *calculation description* and *information model*. Focusing only on these three elements, we will use the term EAM metric instead of EAM KPI in the reminder of this paper.

According to [27] and [32], there is currently only a limited EAM tool support for metrics and their computation. Our research tool Tricia [30], which can be used to describe and manage EA (c.f. [14]) also lacks support of EAM metric computation. Presently, our Java-based tool enables the description of EAs by employing the concept of Hybrid Wikis. It further provides the functionality to export these Hybrid Wiki EA representations into a Ecore models (the meta-model of the EMF, c.f. [2]). The EMF is a Java framework and code generation facility for building tools and other applications based on a structured model. By using Ecore models, we intend to calculate the catalog's metrics at runtime in a model-driven manner, instead of defining Java implementation of each KPI as a hard-wired method in the code-base. Therefore, our paper addresses the following three research questions:

1. What operators are required for the computation of the metrics presented in the catalog?
2. Which languages that support the EMF inherently provide these operators?
3. How could we implement a concrete metric in these identified languages?

The remainder of this paper unfolds as follows. Section 2 is concerned with the identification of a set of operators required to compute the catalog's metrics. Moreover, we assess the extend to which specific languages support this set of operators. In Section 3 we demonstrate how one specific KPI can be implemented in the examined languages. Finally, Section 4 concludes the paper and outlines future research topics.

2 Contribution

For the computation of metrics a distinct set of operators are needed. To discover these operators, we used the following search engines: Google, Google Scholar, and the library system of our university. Between August, 13^{th} and August, 23^{th} 2012 we searched for terms: "metric", "kpi", "operators", "computation", "query", "language", "model", "Ecore" along with possible combinations between them. This literature review-based research has shown that there is no reference list containing concrete operators for the computation of metrics. For this reasons, we focused on operators provided by query languages and their formal underlying algebra. The identified basic operators in addition to the calculation description as well the information model of each metric (c.f. [28]) allowed us to define a set of required operators.

2.1 Operators for implementing EAM metrics

The ability to query models is needed in order to retrieve required computational data from the underlying models. Undoubtedly, one of the most renowned query languages is the Structured Query Language (SQL) [31] which is partly based on relational algebra [22]. In 1972 Codd coined the term relational algebra as "*An algebra whose primary purpose is to provide a collection of operations on relations of all degrees suitable for selecting data from a relational data base*" [16]. According to the relational algebra, there are following eight operations (in line with the terminology of [16]): *cartesian product*, *union*, *intersection*, *difference*, *restriction* (more commonly known as selection [22]), *join*, *division*, and *projection*. In addition to these relational operations, the relational algebra also includes the following six comparison operators (c.f. [22]): *equal*, *not equal*, *less than*, *greater than*, *less than or equal to*, and *greater than or equal to*. Finally, the algebra also comprises logical operators (c.f. [22]): *and*, *or*, and *not*.

Further, [22] stated that relational algebra does not include arithmetic operators, e.g., addition ($+$), subtraction ($-$), multiplication ($*$), and division ($/$). It also lacks aggregate functions as stated in [26]. Against this background, the same author proposed an extension of the relational algebra with aggregate functions.

We systematically evaluated the information provided in [28] to discover the operators that are required to compute the catalog's set of 52 metrics. The focus was on the description of the metric, more precisely on the computation description and the associated information model. We concluded that in our research we need to consider the operators that are already provided by the relation algebra, as well as those included in its extensions. Considering these two aspects we classified the operators into the following two categories: the first group is used for the computation of metrics (*computation operators*), the second group serves at querying the underlying information models (*querying operators*).

Computation operators Firstly, the computation of each metric requires a set of arithmetic operators applied on numerical values. We investigated all the metrics in the catalog and translated the calculation descriptions into expressions containing corresponding arithmetic operators. The analysis proved, that all of the basic arithmetic operators (c.f. [22]), that is, *addition*, *substraction*, *multiplication*, *division* are needed.

Secondly, specific aggregate functions are needed allowing the computation of collection of numerical values. Klug in [26] defined aggregate functions as follows: "*An aggregate function takes a set of tuples as an argument and produces a single simple value as a result*". Similar to the basic arithmetic operators, our analysis showed, that the following set of aggregate functions (c.f. [26]) are needed to compute the catalog's metrics: *count*, *sum*, *average*, *minimum* and *maximum*.

Thirdly, we need comparison operators, used in logical statements, to determine equality or inequality of variables or values. Our analysis revealed, that following set of comparison operators (c.f. [16]) satisfies the needs of the catalog: *equal*, *not equal*, *grater than*, *less than*, *greater then or equal to*, *less then or equal to*. In addition, we also found, that we need following logical operators: *and*, *or*, as well as *not*.

Querying operators In the second step of our analysis, we examined the catalog to identify operators allowing to construct queries to retrieve data captured by each related information model. In translating each's metric input data into queries we were able to elaborate, that following set of operations (according to [16]) satisfies the needs of the catalog: *restriction* (also known as selection [22]), *join*, and *projection*.

2.2 Languages

As our research shows, following set of languages supports Ecore: *Java*, *OCL*, *LambdaJ*, *JavaScript* (we use the *Rhino* interpreter) and *LINQ*. EMF supports the direct translation of its models into Java objects. Furthermore, OCL is directly supported by EMF in its role as a navigational language, hence can be used to query these models and their data via a specific console. The three remaining languages LambdaJ, JavaScript, and LINQ have been identified with the help of a literature and web review. For each of these five languages we studied the native operators offered as specified in the according documentation sources [5, 7, 6, 21, 9]. Thereby, the term native refers to the built-in operators of these languages. The results were compared to the essential set of operators discussed above. Table 1 illustrates our findings.

		Java	*JavaScript with jLinq*	*LambdaJ*	*OCL*	*LINQ*
Aggregaton functions	Count	size	count	size	size	Count, Long-Count
	Sum			sum	sum	Sum
	Average			avg		Average
	Minimum	min	min	min	min	Min
	Maximum	max	max	max	max	Max
Relational	Selection		Select	Select	Select	Select, SelectMany
	Grouping			Group		GroupBy
	Join		Join			Join, GroupJoin
Arithmetic	Addition	$+$	$+$	$+$	$+$	$+$
	Substraction	$-$	$-$	$-$	$-$	$-$
	Multiplication	$*$	$*$	$*$	$*$	$*$
	Division	/	/	/	/	/
Comparison	Equal	$=$	equals,$=$	equal, $=$	$=$	$=$
	Not equal	$!=$	$!=$	$!=$	$<>$	$!=$
	Greater than	$>$	greater	greaterThan	$>$	$>$
	Less than	$<$	less,$<$	lessThan,$<$	$<$	$<$
	Greater than or equal to	$\geq$	greaterEquals	$\geq$	$\geq$	$\geq$
	Less than or equal to	$\leq$	lessEquals	$\leq$	$\leq$	$\leq$
Logical	And	&&	and, &&	and	and	&&
	Or	||	or,||	or	or	||
	Not	!	not	not	not	!

Table 1: Support of the identified operators by the examined languages

As illustrated in Table 1, only LINQ inherently supports all of the required operators. Java is missing two aggregate functions (sum and average) and it provides no relational operators. On the other hand, we observe that LambdaJ extends Java with almost all these missing operators, except for join. Javascript, along with the jLinq library does not support the grouping relation and has no native implementation for the sum and average functions. The average function is not specified either in OCL, which lacks also the grouping and join relational operators. Therefore, for the last four languages, special methods supporting the required operators have to be implemented as workarround to support the missing operators.

3 Evaluation

To demonstrate the applicability of metric implementation in the different languages we select one specific metric as a running example. The structure elements of this KPI relevant for its computation are described as follows:

– *Description*: Measurement of the coverage of IT continuity plans in respect to business-critical processes.
– *Calculation description*: Number of business-critical processes relying on business applications, not covered by IT continuity plan, divided by total number of business-critical processes relying on business applications.
– *Information model*: Figure 1 illustrates the information model of this metric.

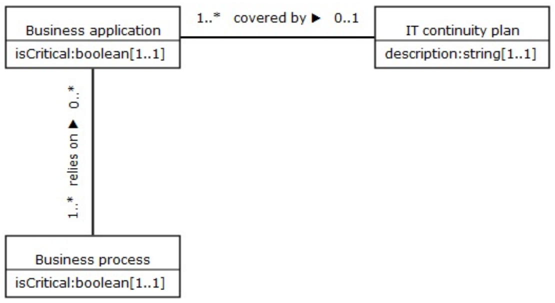

Fig. 1. Information model of the metric "IT continuity plans for business applications supporting critical processes"

In the following, we will focus on the computation of this concrete example described above.

3.1 Java

The **Java** programming language is a general-purpose concurrent class-based object-oriented programming language specifically designed to have as few implementation dependencies as possible [20]. In this paper we use EMF to translate the metrics information models into efficient and easily customizable Java code. As we can see from the snippet the computation of the metric is possible using the existing constructs of Java. The issue with this implementation is the lack of the relational operations e.g. selection or join which can make relatively easier the process of retrieving data from the collections (c.f. Table 1).

Listing **??** illustrates the computation of our example metric using Java. To keep the code readable, we do not show in the snippet the part of the code that checks for NullPointerException when there are business processes that do not rely on any business applications.

backgroundcolor= numbers=left, numberstyle= keywordstyle=

```
[language=Java, basicstyle=\footnotesize, caption=Java implementation, label=javaImplementat
int criticalBP = 0;
int criticalBPNotCovered = 0;
for (BusinessProcess process : allBusinessProcesses){
if(process.isCritical() && process.getReliesOn() != null)
{
criticalBP += 1;
if (process.getReliesOn().getCoveredBy() == null)
criticalBPNotCovered += 1;
}
}

double result = criticalBPNotCovered/criticalBP * 100;
```

3.2 OCL

The **Object Constraint Language (OCL)** is a specification language for describing constraints on object-oriented models [25]. Despite the fact that OCL was originally used to add constraints to models, now OCL is being widely used in different fields [33]. In this paper we describe OCL as a language employed to query collections of data.

The OCL implementation in EMF is supported through Eclipse OCL. The Eclipse OCL implementation is part of the Model Development Tools (MDT) project. It allows the evaluation of OCL queries on Ecore models using a console for the interactive evaluation of OCL expressions on models [3]. Among the many operations that OCL defines on the collection types, the operation used to specify a selection from a specific collection is the *select* command [17]. Using this command we can specify which elements of a collection we want to retrieve to compute a concrete metric as illustrated in Listing **??**. In this example we take the sizes (number of items) of the collections that match a specific criterion and use the values to compute the metric.

An advantage of using this language is the fact that the evaluation of OCL expressions does not change the state of the system, because they return just a value [21].

[language=OCL, basicstyle=, caption=OCL implementation, label=oclImplementation,frame=single,columns=fu
(BusinessProcess.allInstances()-¿ select(process:BusinessProcess—process.isCritical = true and process.reliesOn-¿size()¡¿0)-¿size())/ (BusinessProcess.allInstances() -¿ select(process: BusinessProcess— process.isCritical = true and process.reliesOn.coveredBy-¿size()=0)-¿size())*100

3.3 LambdaJ

Extending Java by the **LambdaJ** library allows the manipulation of collections in a pseudo-functional and statically typed way [8]. Using LambdaJ the iterations and loops on collections of data can be avoided through a set of operations on collections that this library provides.

Listing **??** illustrates the computation of our example metric using LambdaJ. In using the *select* operator we retrieve from a collection those items that match the criterion specified by the *having* operator. The resulting value is computed using the sizes of the two collections that match our query.

Among the disadvantages of LambdaJ we can mention the fact that it is less performant than Java [8], and as we have experienced, sometimes there is a lack of proper documentation.

[language=Java, basicstyle=, caption=LambdaJ implementation, label=lambdajImplementation,frame=single,col
int criticalBP = select(businessProcesses, having(on(BusinessProcess.class).isCritical()). and(having(on(BusinessProcess.class).getReliesOn(), not(equalTo(null))))).size();

int criticalBPNotCovered = select(businessProcesses, having(on(BusinessProcess.class).isCritical()). and(having(on(BusinessProcess.class).getReliesOn().getCoveredBy(), equalTo(null)))).size();

double result = criticalBPNotCovered/criticalBP * 100;

3.4 JavaScript

JavaScript is defined as a high-level, dynamic and un-typed interpreted programming language that is well-suited to object-oriented and functional programming styles [19]. To integrate scripting with JavaScript in the EMF environment we use **Rhino** [11]. Rhino is a JavaScript interpreter which is entirely written in Java. Its main purpose is to facilitate writing programs that use the Java platform APIs in JavaScript [19]. With Rhino we can query the Java classes generated by EMF using JavaScript code, as we have illustrated in Listing **??**. In this example we used the **jlinq** library to make the code more readable. **Jlinq** is a JavaScript library that allows us to perform queries on arrays of data [6]. Instead of using loops, we can write queries to retrieve the data we need. As illustrated in the example, the business processes are checked to match a specified criterion using the keywords *is* and *isNot*. Therefore we retrieve the sizes of the two collections used to compute the value of the metric.

[language=ocl, basicstyle=, caption=Rhino implementation, label=rhinoImplementation,frame=single,columns=f
var criticalBP = jLinq.from(businesProcesses). is('isCritical').is('reliesOn').select().length;

var criticalBPNotCovered = jLinq.from(businesProcesses). is('isCritical').isNot('reliesOn.coveredBy').select().length

var result = criticalBPNotCovered/criticalBP * 100;

3.5 LINQ

Language Integrated Query (LINQ) is a language designed to simplify data queries in .NET [27]. The advantage of LINQ is its ability to query collections of objects by using language keywords and familiar operators [10]. LINQ is introduced as part of the .NET Framework, and it can construct a query in C# [23] or Visual Basic [35]. During our related work research on languages supporting Ecore, we found out that there is a project called EMF4NET [4]. However, when we tried to use this project we discovered, that it was still in proposal phase (the status was last checked on August, 31^{th} 2012) and there was no workaround available. Thus, we tried to find alternative paths to compute metrics using LINQ on Ecore models. Thereby, we found and tried to evaluate the tool Acceleo [1].

Acceleo is a code generator used to transform models into code [1]. One of the possible transformations is to generate C# code from UML 2.0 models. This is evaluated not to be a favorable solution in our case, because firstly we would need to translate Ecore models into UML 2.0 models, and secondly we could not directly use the instance data.

4 Conclusion and outlook

Depending on the fact, that there is a lack of tool support for the computation of EAM metrics (c.f. [27], [32]), we intend to extend our java-based research software to provide this capability. Therefore, we first identified operators that a language employed for the computation of EAM metrics in the context of Ecore has to provide. Then we identified languages supporting Ecore and assessed the degree to which these languages inherently support the required operators. Finally, we implemented all of the 52 EAM metrics from the catalog in each of these languages to prove, that the computations can be performed. After finishing the implementation following new questions emerged, which are to be considered in future research:

– *Bigger set of examined EAM metrics* - we focused in our studies only on the 52 EAM metrics provided by the catalog. Further research should consider other sources of EAM metrics, which could lead to an extension of the identified operator set.
– *Integration* - currently we use our research tool to export Ecore models and the metrics are computed out of the tool. In future, one of the described solutions is to integrated in our tool to extend it by the capability to compute metrics on run-time.
– *Domain-specific language (DSL)* - adopting the solutions presented above, we are forced to create redundant data (Hybrid models are exported in Ecore models). Therefore, future research should also focus on the development of a new DSL, providing all identified operators and supporting our Hybrid models.

References

1. Acceleo official Web site. `http://www.acceleo.org/pages/home/en`, 2012. [Online; accessed May-2012].
2. Eclipse Modeling Framework Project Web site. `http://www.eclipse.org/modeling/emf/`, 2012. [Online; accessed May-2012].
3. Eclipse project : Model Development Tools (MDT). `http://www.eclipse.org/modeling/mdt/?project=ocl`, 2012. [Online; accessed May-2012].
4. EMF4Net proposal project Web site. `http://wiki.eclipse.org/EMF4Net_Proposal`, 2012. [Online; accessed May-2012].
5. Java dcumentation. `http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html`, 2012. [Online; accessed July-2012].

6. jlinq Project Web site. `http://hugoware.net/projects/jlinq`, 2012. [Online; accessed May-2012].
7. LambdaJ dcumentation. `http://lambdaj.googlecode.com/svn/trunk/html/apidocs/ch/lambdaj/Lambda.html`, 2012. [Online; accessed July-2012].
8. LambdaJ official Web site. `http://code.google.com/p/lambdaj/`, 2012. [Online; accessed May-2012].
9. LINQ dcumentation. `http://msdn.microsoft.com/en-us/library/bb882641`, 2012. [Online; accessed July-2012].
10. LINQ Project Web site. `http://msdn.microsoft.com/en-us/library/bb397926.aspx`, 2012. [Online; accessed May-2012].
11. Rhino official Web site. `http://www.mozilla.org/rhino/`, 2012. [Online; accessed May-2012].
12. S. Aier, C. Riege, and R. Winter. Classification of enterprise architecture scenarios – an exploratory analysis. *Enterprise Modelling and Information Systems Architectures*, 3:14–23, 2008.
13. J. Black and L. Porter. *Management: meeting new challenges*. Prentice Hall, 2000.
14. T. Büchner, F. Matthes, and C. Neubert. Data model driven implementation of web cooperation systems with tricia. In *Proceedings of the Third international conference on Objects and databases*. Springer-Verlag, 2010.
15. S. Buckl, T. Dierl, F. Matthes, and C. M. Schweda. Building blocks for enterprise architecture management solutions. In F. e. a. Harmsen, editor, *Practice-Driven Research on Enterprise Transformation, second working conference, PRET 2010, Delft*, pages 17–46, Berlin, Heidelberg, Germany, 2010. Springer.
16. E. F. Codd. Relational completeness of data base sublanguages. In *Database Systems*, pages 65–98. Prentice-Hall, 1972.
17. R. S. Corporation. *Object Constraint Language Specification: Version 1.1*. Rational Software, 1997.
18. C. Fitz-Gibbon. *Performance Indicators*. Bera Dialogues. Multilingual Matters, 1990.
19. D. Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media, 2011.
20. J. Gosling, B. Joy, G. Steele, and G. Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.
21. O. M. Group. Ocl 2.0 specification, June 2005.
22. T. Halpin and T. Morgan. *Information Modeling and Relational Databases, Second Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2 edition, Mar. 2008.
23. A. Hejlsberg, S. Wiltamuth, and P. Golde. *C# Language Specification*. Addison-Wesley Longman Publishing Co., Inc., 2003.
24. International Organization for Standardization. ISO/IEC 42010:2007 Systems and software engineering – Recommended practice for architectural description of software-intensive systems, 2007.
25. A. Kleppe, J. Warmer, and S. Cook. Informal formality? the object constraint language and its application in the uml metamodel. In *Selected papers from the First International Workshop on The Unified Modeling Language: Beyond the Notation*, pages 148–161. Springer-Verlag, 1999.
26. A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, July 1982.
27. F. Marguerie, S. Eichert, and J. Wooley. *Linq in action*. Manning Publications Co., 2008.
28. F. Matthes, I. Monahov, A. Schneider, and S. Christopher. Eam kpi catalog v1.0. Technical report, Technische Universität München, München, Germany, 2012.

29. F. Matthes, I. Monahov, A. Schneider, and S. Christopher. Towards a unified and configurable structure for ea management kpis. In 7th *Workshop on Trends in Enterprise Architecture Research (TEAR 2012)*, Barcelona, Spain, 2012.
30. F. Matthes and C. Neubert. Using hybrid wikis for enterprise architecture management. In 7th *International Symposium on Wikis and Open Collaboration (WikiSym)*, Mountain View, California, USA, 2011.
31. J. Melton. Sql language summary. *ACM Comput. Surv.*, Mar. 1996.
32. J. Short and C. Wilson. Gartner assessment of enterprise architecture tool capabilities. 2011.
33. H. Song, Y. Sun, L. Zhou, and G. Huang. Towards instant automatic model refinement based on ocl. pages 167–174, New York, NY, USA, 2007. APSEC.
34. The Open Group. TOGAF "Enterprise Edition" Version 9. `http://www.togaf.org` (cited 2011-06-08), 2009.
35. P. Vick. The microsoft visual basic language specification, 2008.