Technische Universität München

sebis

TEAR 2014

Type-Safety in EA Model Analysis

01.09.2014

Thomas Reschenhofer, Ivan Monahov, Florian Matthes

Software Engineering for Business Information Systems (sebis)
Department of Informatics
Technische Universität München, Germany

wwwmatthes.in.tum.de

Overview

1. **Introduction**
   - Model-based EA analysis
   - The untyped core expression language

2. **Contribution**
   - The typed model-based expression language (MxL)
   - Transparency of quantitative EA model
   - In-browser code editor
   - Automated refactoring

3. **Conclusion and outlook**

Motivation

- An EA model covers business as well as IT aspects to provide a holistic view of an organization and supports decision makers with relevant information.

- **Controlling** and **planning** an EA and its evolution requires its analysis
- **Qualitative** EA analysis **not sufficient** because of size and complexity of EAs

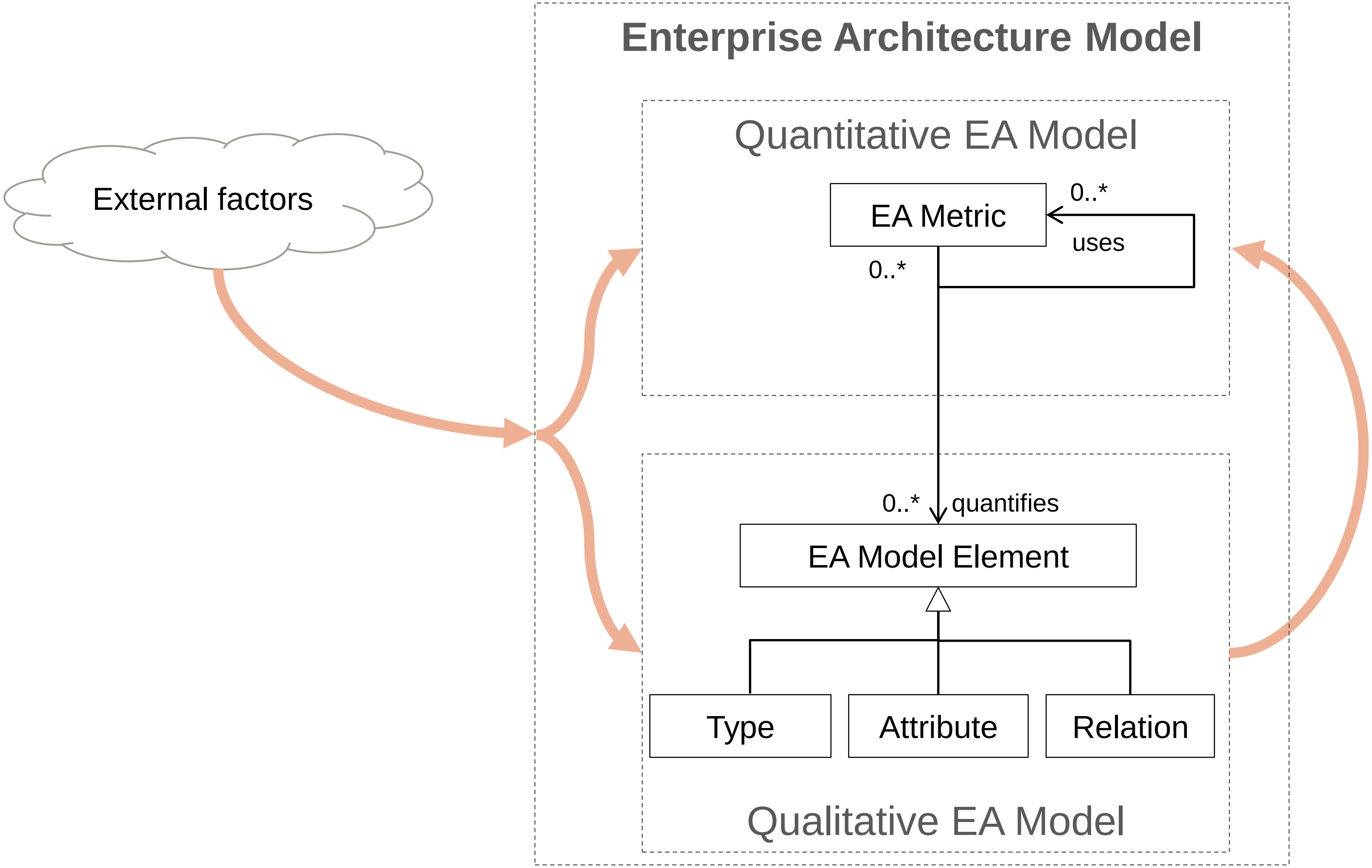
  → **Quantitative** EA analysis with EA metrics

- Definition of EA metrics by **domain-specific language (DSL)** based on EA information model

  → Design decisions (functional?, object-oriented?, **statically typed**?, etc.)

- *What are the **disadvantages** of a **dynamically typed DSL** for defining EA metrics in a model-based EA tool?*
- *What are the **implications/benefits** of such a DSL's **static type-safety** (in particular when considering **dynamic** EA models)?*

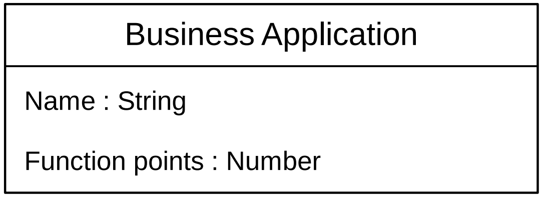
Model-based EA analysis



The untyped core expression language

- Functional and sequence-oriented query/expression language
  - Higher-order functions & lambda expressions
- Inspired by OCL and LINQ
  - Supports Microsoft's *Standard Query Operators*
- Integrated in model-based *Hybrid Wiki* collaboration platform
- **No** static type-safety
  - **No** validation of static semantics at compile-time

Exemplary meta-model



Exemplary Query with untyped core expression language

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

Monahov, I.; Reschenhofer, T.; Matthes, F.: Design and prototypical implementation of a DSL empowering business users to define EAM KPIs

Overview

1. **Introduction**
   - Model-based EA analysis
   - The untyped core expression language

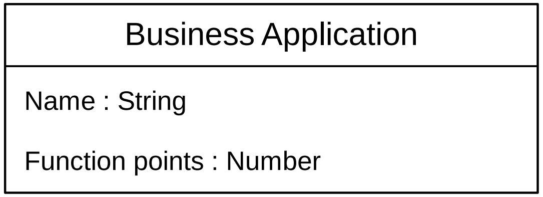
2. **Contribution**
   - The typed model-based expression language (MxL)
   - Transparency of quantitative EA model
   - In-browser code editor
   - Automated refactoring

3. **Conclusion and outlook**

A type system to support type-safe EA analysis

- **Sub typing**
  - Re-use of functionality
- **Polymorphic types**
  - Type parameters in types and functions
  - E.g., signature of select-function: $Sequence\langle T\rangle \times (T \rightarrow U) \rightarrow Sequence\langle U\rangle$
- **Restricted type inference**
  - Omit explicit annotation of types (e.g., types of function parameters)
  - Implicit determination of types

Exemplary meta-model



Business Application

Name : String

Function points : Number

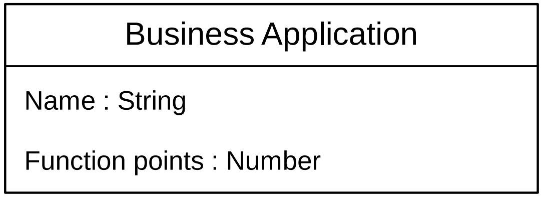
Exemplary Query with untyped core expression language

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

The typed model-based expression language (MxL)

- Implementation of **type-system**
  - Sub typing
  - Polymorphic types
  - Restricted type inference
- **Static type-safety**
  - Validation of an MxL expression‘s static semantics at compile-time

Exemplary meta-model



Exemplary MxL query

```
find ‘Business Application’
  .select((ba:‘Business Application’) =>
        ba.‘Function points’)
  .sum()
```

Exemplary MxL query (with implicit parameter type)

```
find ‘Business Application’
  .select(ba => ba.‘Function points’)
  .sum()
```

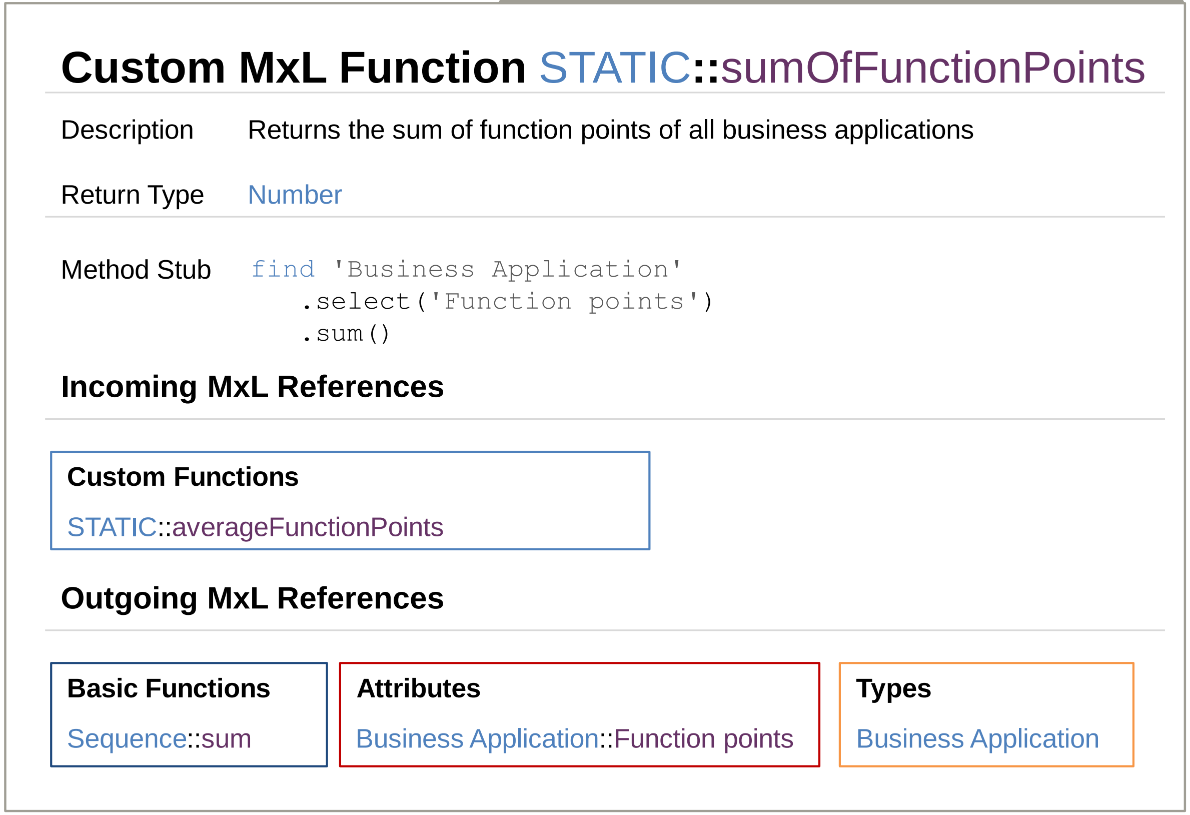
Exemplary MxL query (with implicit lambda)

```
find ‘Business Application’
  .select(‘Function points’)
  .sum()
```

Transparency of quantitative EA model

- **Static type-safety enables validation of static semantics**
  - Resolving identifiers and checking their types
  - Analysis of dependencies between EA metrics and model elements
  - Automated generation of quantitative model's computation graph

Screenshot of an MxL function



Custom MxL Function STATIC::sumOfFunctionPoints

Description Returns the sum of function points of all business applications

Return Type Number

Method Stub

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

Incoming MxL References

Custom Functions

STATIC::averageFunctionPoints

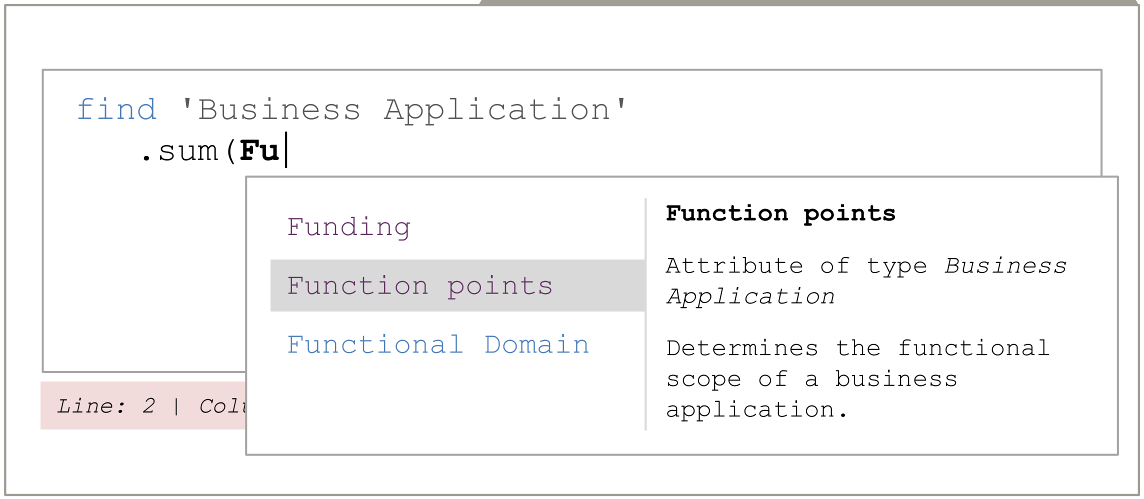
Outgoing MxL References

Basic Functions	**Attributes**	**Types**
Sequence::sum	Business Application::Function points	Business Application

In-browser code editor

- Syntax highlighting
  - Highlighting of keywords, strings, etc.
- Code completion
  - Provision of list of possible identifiers
  - Proposes elements from quantitative and qualitative EA model
- Integrated documentation
- Code navigation
  - Incoming and outgoing references are clickable
- Error localization
  - Highlighting of origin of syntactic and semantic errors

Screenshot of the MxL code editor



```
find 'Business Application'
    .sum(Fu
```

Function points

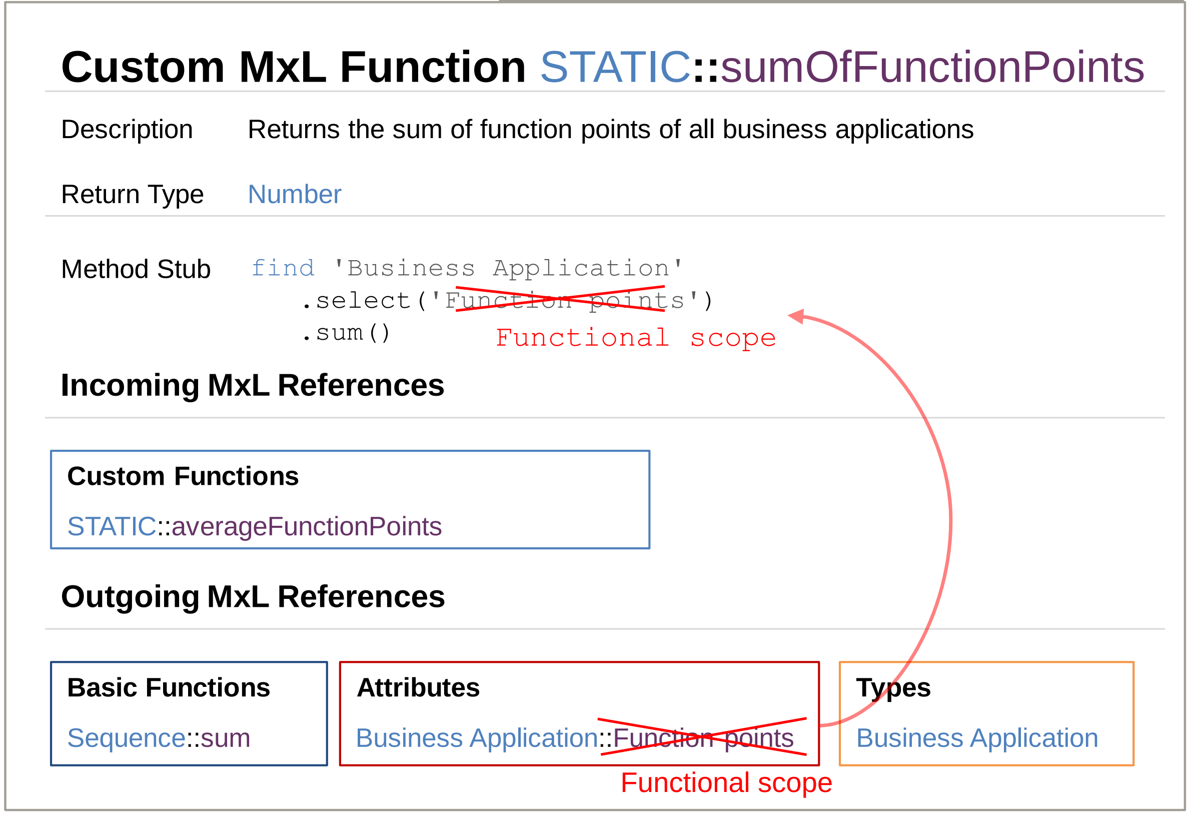
Attribute of type *Business Application*

Determines the functional scope of a business application.

Automated refactoring

- **Automated adaption** of expression **on changes** of the meta model, e.g., when
  - Renaming of elements
  - Changing the type of elements
  - Deleting elements
  - Creating elements
- Keeping semantic consistency

Screenshot of an MxL function



Custom MxL Function STATIC::sumOfFunctionPoints

Description Returns the sum of function points of all business applications

Return Type Number

Method Stub

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

Functional scope

Incoming MxL References

Custom Functions

STATIC::averageFunctionPoints

Outgoing MxL References

Basic Functions

Sequence::sum

Attributes

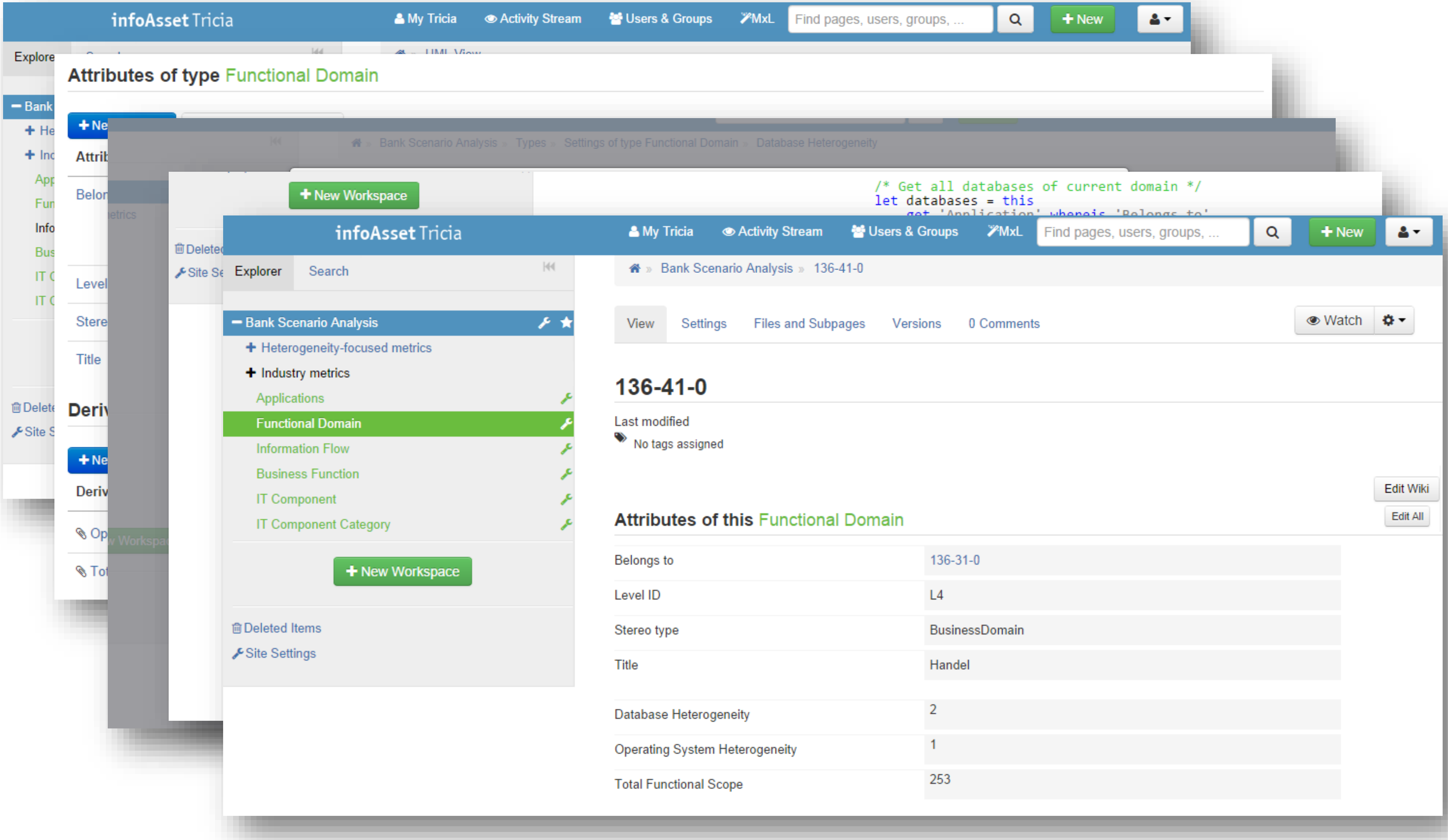
Business Application::~~Function points~~

Functional scope

Types

Business Application

Evaluation: Defining metrics for application landscapes complexity of four German banks



Attributes of type Functional Domain

Bank Scenario Analysis » Types » Settings of type Functional Domain » Database Heterogeneity

```
/* Get all databases of current domain */
let databases = this
```

Bank Scenario Analysis » 136-41-0

- Bank Scenario Analysis
  - + Heterogeneity-focused metrics
  - + Industry metrics
  - Applications
  - Functional Domain
  - Information Flow
  - Business Function
  - IT Component
  - IT Component Category

136-41-0

Last modified

No tags assigned

Attributes of this Functional Domain

Belongs to	136-31-0
Level ID	L4
Stereo type	BusinessDomain
Title	Handel
Database Heterogeneity	2
Operating System Heterogeneity	1
Total Functional Scope	253

Overview

sebis

1. **Introduction**
   - Model-based EA analysis
   - The untyped core expression language

2. **Contribution**
   - The typed model-based expression language (MxL)
   - Transparency of quantitative EA model
   - In-browser code editor
   - Automated refactoring

3. **Conclusion and outlook**

Conclusion & outlook

- Untyped core expression language
  - Enables the user-oriented definition of EA metrics
  - **Problem**: Lack of validation of static semantics on compile-time

- Typed model-based expression language (MxL)
  - Validation of **static semantics** through type checking
  - **Transparency** of quantitative EA model
  - Enables **automated refactoring** on changes of meta model

- Prototype evaluated in research environment
  - Measuring complexity of application landscapes
  - Data from four German banks

- Outlook / Open issues
  - Performance issues on execution of MxL queries
  - Temporal EA analysis
  - Evaluation strategies of MxL expressions

Questions?

Thomas Reschenhofer
M.Sc.

Technische Universität München
Department of Informatics
Chair of Software Engineering for
Business Information Systems

Boltzmannstraße 3
85748 Garching bei München

Tel +49.89.289.17100
Fax +49.89.289.17136

thomas.reschenhofer@tum.de
wwwmatthes.in.tum.de