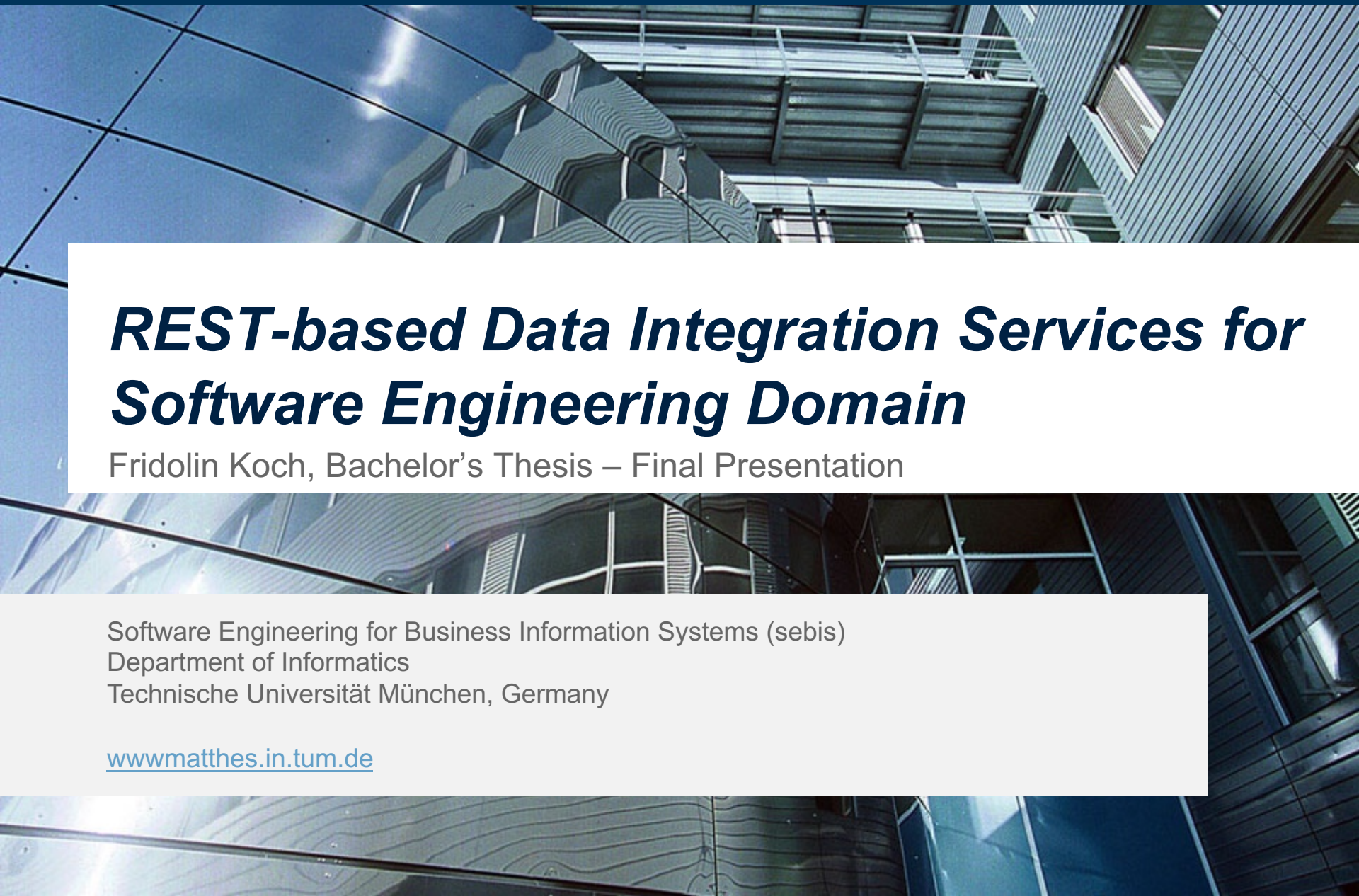


REST-based Data Integration Services for Software Engineering Domain

Fridolin Koch, Bachelor's Thesis – Final Presentation

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

wwwmatthes.in.tum.de

Motivation

- Existing barrier in the adoption of software architecture knowledge management (SAKM) systems
  - Many different software architecture life cycle tools produce data in different formats (Enterprise Architect, Excel, Jira, etc.)
  - Repeatedly integrating this data into such a system can be a challenging and tedious task
- In general the task of data integration is addressed by **E**xtract-**T**ransform-**L**oad-Tool (ETL-Tool)
  - Wide range of commercial and open source ETL-Tool available
  - But: Mostly tailored to generic use cases → Difficult to embedded in existing domain specific tools

Research questions

RQ1
What are the use cases of data integration services?

RQ2
What are the features of the existing data integration service providers?

RQ3
How to design a framework for the data integration services in software engineering domain?

RQ1: Use cases

Two roles where identified

- Developers (DEVs) extend the application with custom services
- Software Architects (SAs) use the system to define, execute and monitor data integration pipelines

DEVs

- Implement services which load or extract data
- Define configuration parameters
- Expose the domain model of the source or target system
- Use the graphical user interface (GUI) to test their implementation

SAs

- Provide configuration parameters using the GUI
- Explore the data model of a service within the GUI
- Create mapping between a source and a target system
- Define pipelines
- Invoke the execution of defined pipelines
- Check system logs to verify the executed pipelines

RQ2: Existing data integration tools

Tools identified using web search

- Apatar
- CloverETL
- IBM InfoSphere DataStage
- Informatica
- Pentaho
- RhinoETL
- Talend Open Studio for Data Integration
- UnifiedViews

For the analysis open-source and partial open-source tools where selected

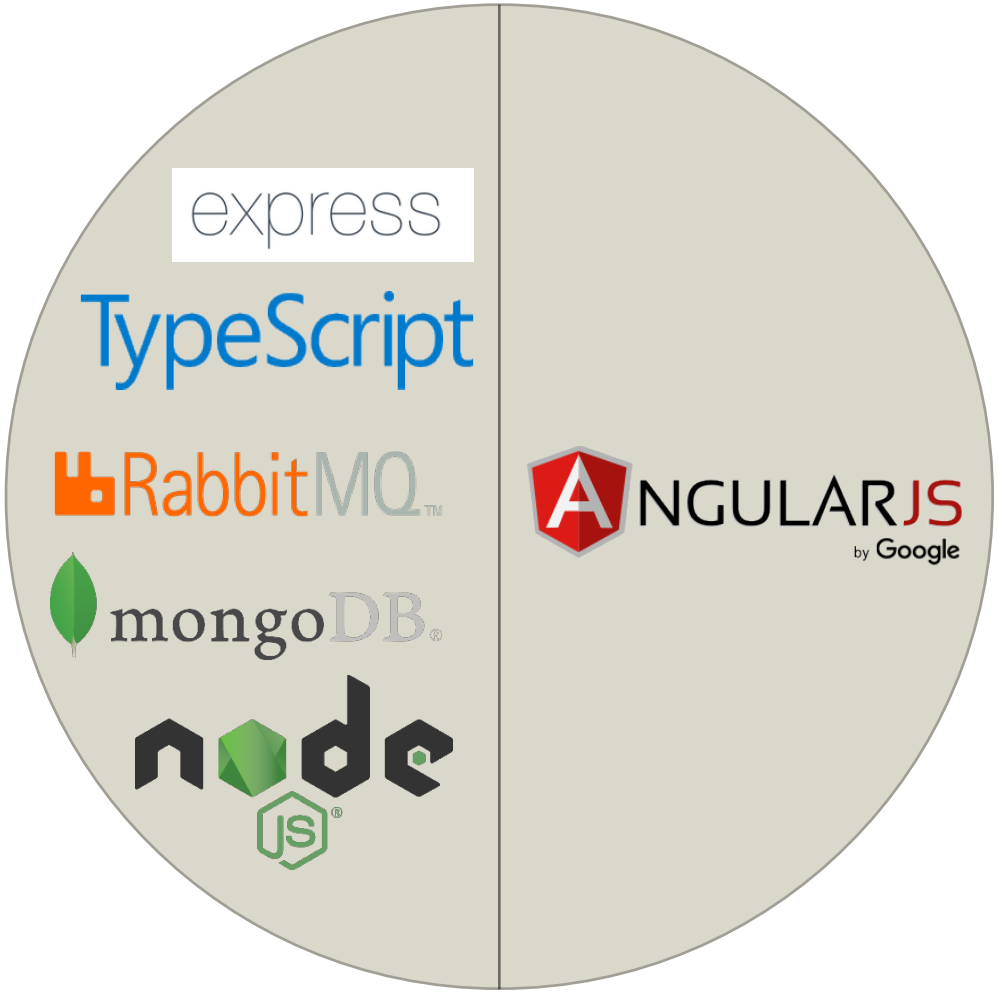
Identified features

- Visual pipeline builder
- Generic use case
- Support many connectors

Identified traits

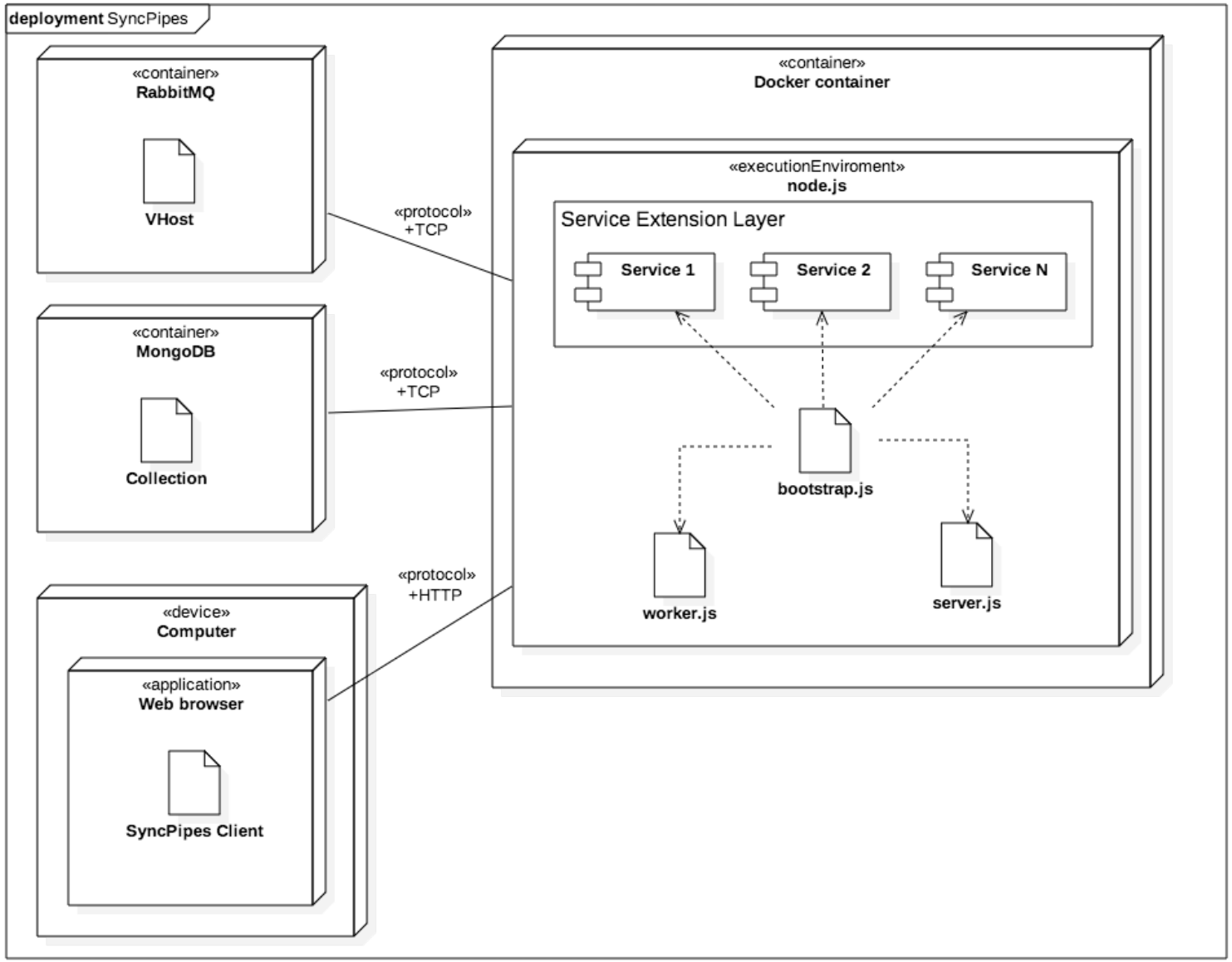
- Focus on data analyst
- Focus is not extensibility!

Technology stack

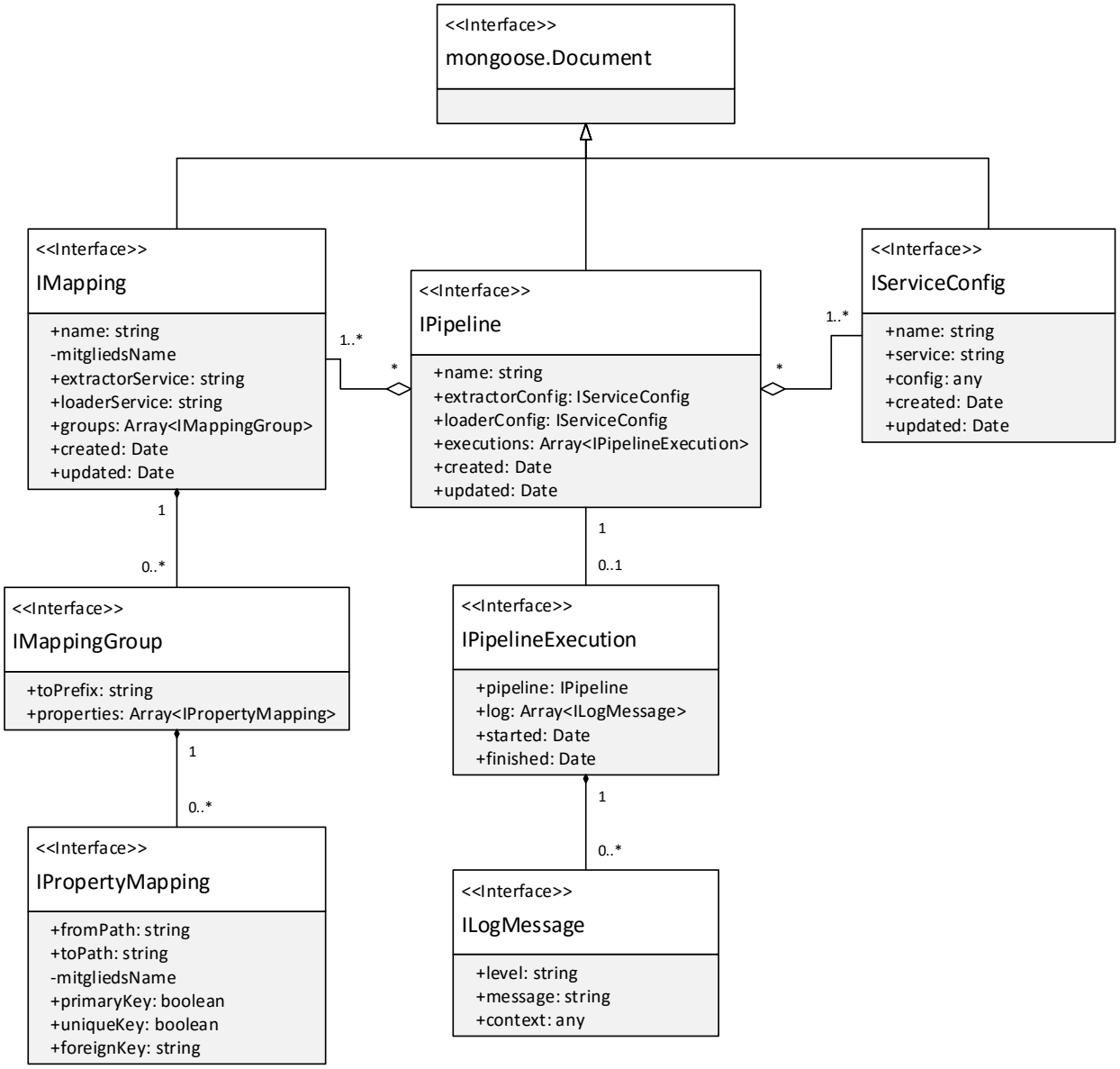


MongoDB **E**xpress.js **A**ngular.js **N**ode.js

Top-level application architecture



Storage layer

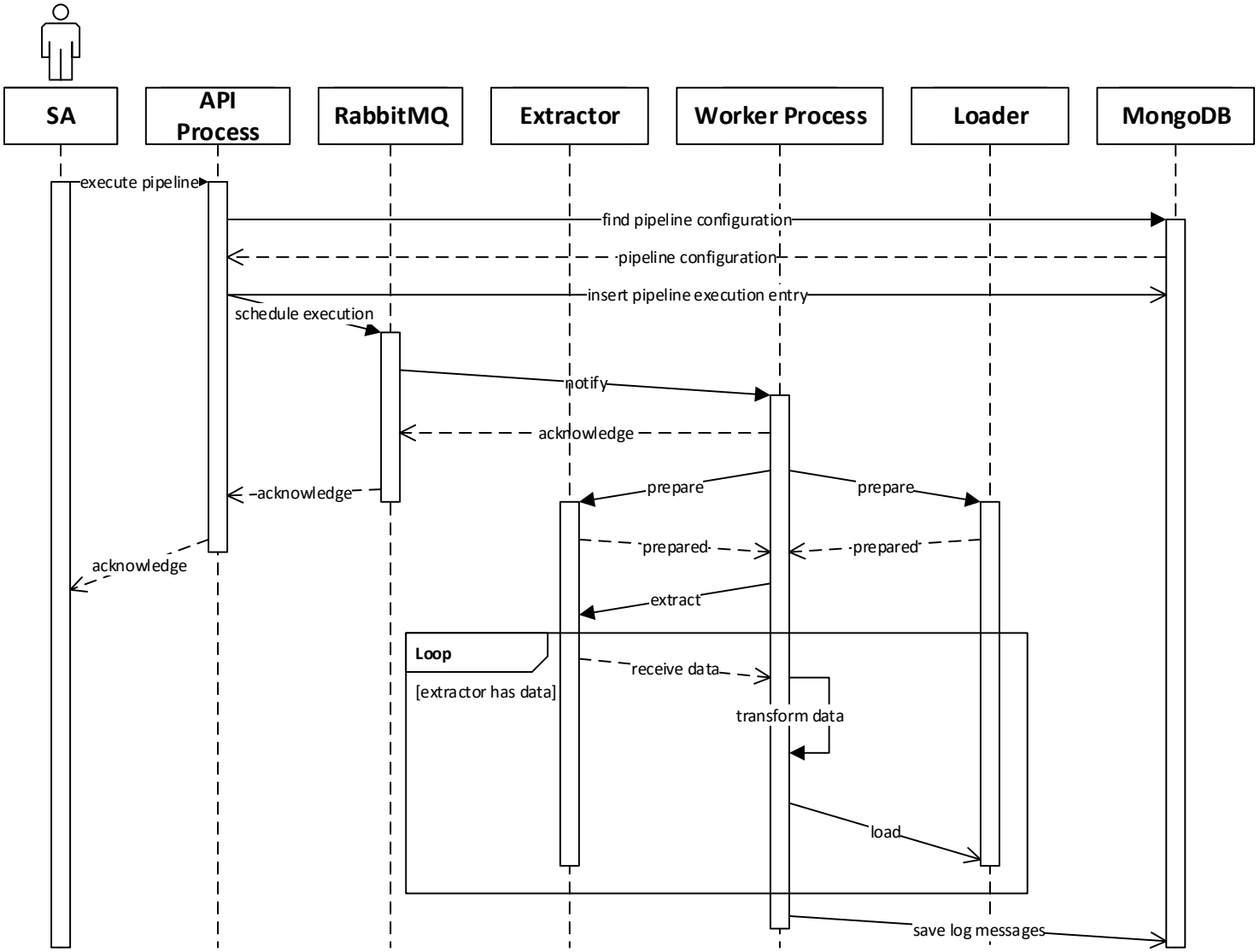


Software architect workflow

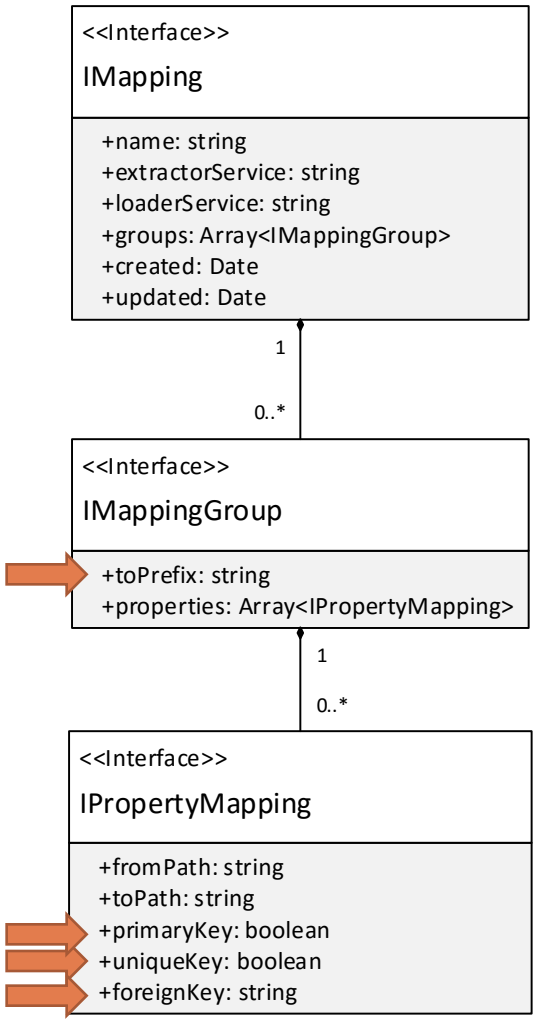
The software architect (SA) uses the application to transform data

1. The SA selects the extractor and loader service
2. For both services SA may create a new configuration or use an existing configuration
3. The SA creates a mapping between the two services/system
4. A pipeline is composed by the SA, by selecting a loader & extractor configuration and a mapping
5. The SA executes the composed pipeline
6. The SA verifies the correctness of the execution by reviewing the logs

Data transformation I



Data transformation II



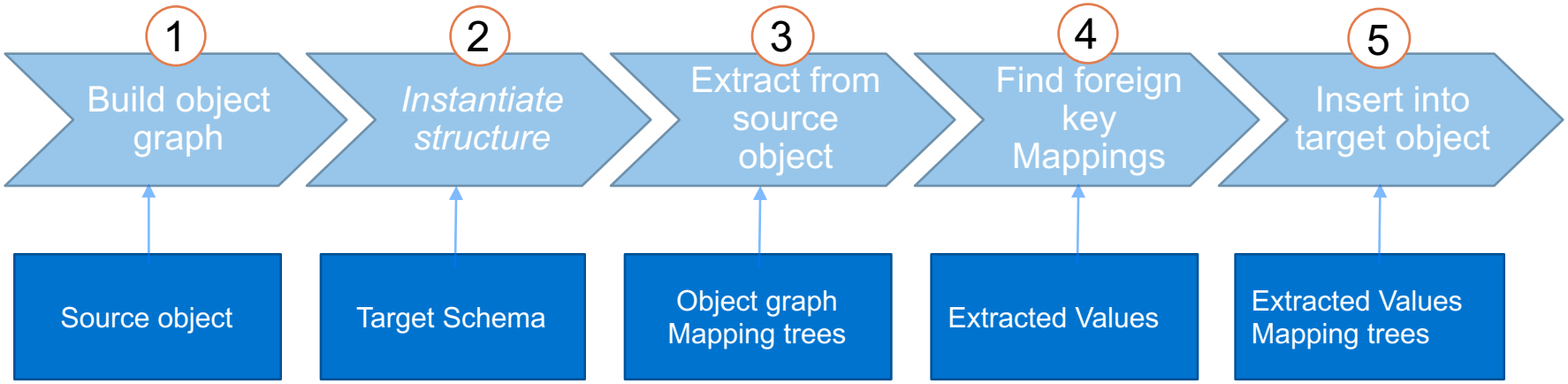
- Mapping is based on paths, which are similar to XPath but simpler.
  Example: */projects/issues/author*
  - Refers to all authors of all issues of all projects
  - Automatic differentiation between objects and arrays
- **toPrefix** is the placement path in the target object
- **uniqueKey** indicates if the transformer should be aware of duplicates (only works with arrays)
- **foreignKey** is a placement information for the parent mapping group, comparable to a simple SQL WHERE statement
- **primaryKey** is a flag indicating if the transformer will merge the target object, if mapped multiple times, to an object.

Data transformation III

primaryKey example

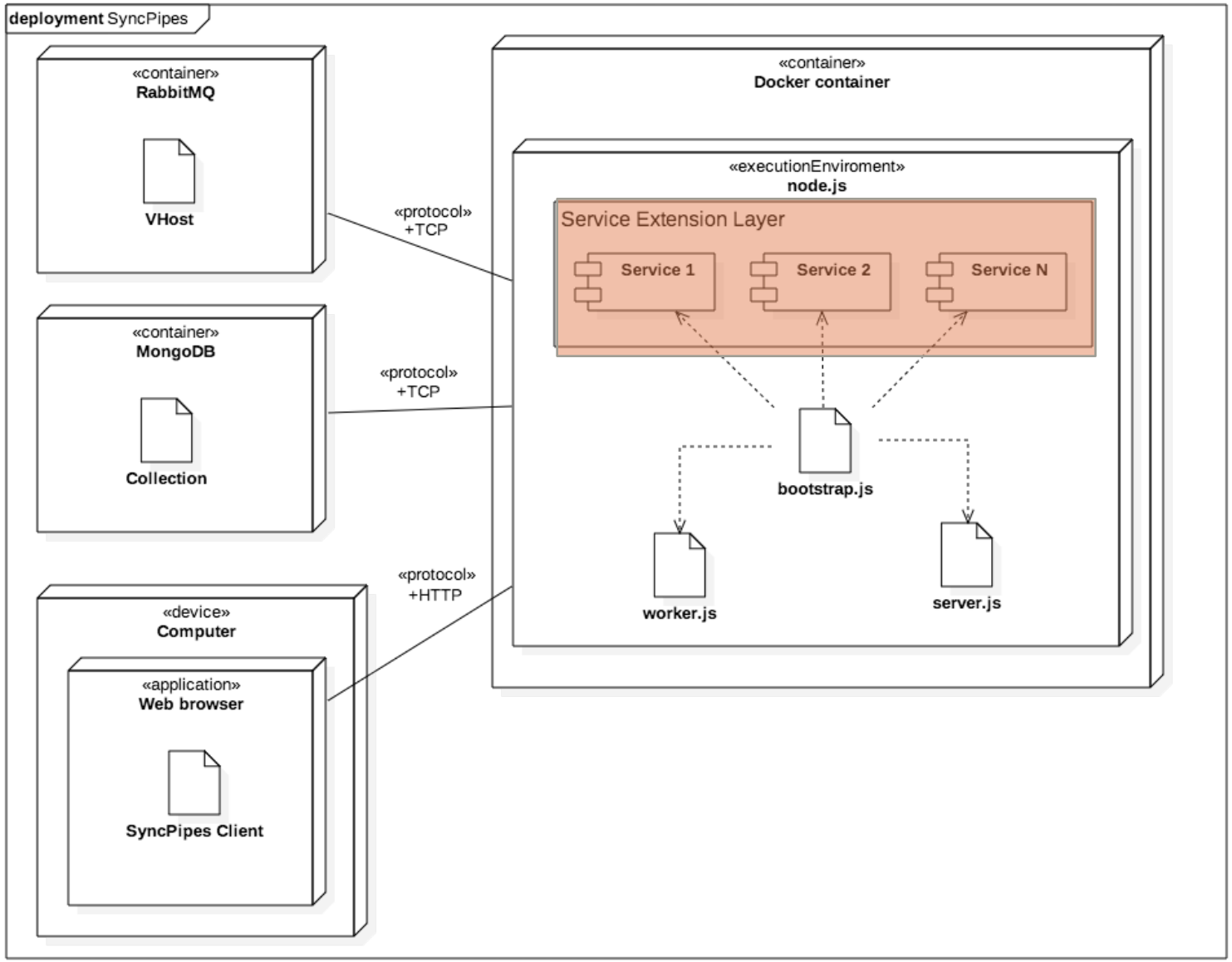
```
// Source object
{
  "users": [
    {"id":1, "name": "John Doe"},
    {"id":1, "location": "Munich"},
  ]
}
// Mapping
[{"fromPath": "users/id", "toPath": "user/id", "primaryKey": true},
{"fromPath": "users/name", "toPath": "user/name"},
{"fromPath": "users/location", "toPath": "user/location"}]
// Object after 1. iteration
{
  "id": 1,
  "name": "John Doe"
}
// Object after 2. iteration with primaryKey = true
{
  "id": 1,
  "name": "John Doe"
  "location": "Munich"
}
// Object after 2. iteration with primaryKey = false
{
  "id": 1,
  "location": "Munich"
}
```

Data transformation IV



1. Build a tree structure from the extracted data
2. Make an object instance using the target systems JSON-Schema
3. Extract data from from the source object using the Mapping
4. Find *foreignKey* Mappings
5. Depending if *foreignKey* Mappings were found, the algorithm decides how to insert the extracted data into the target object.

Top-level application architecture



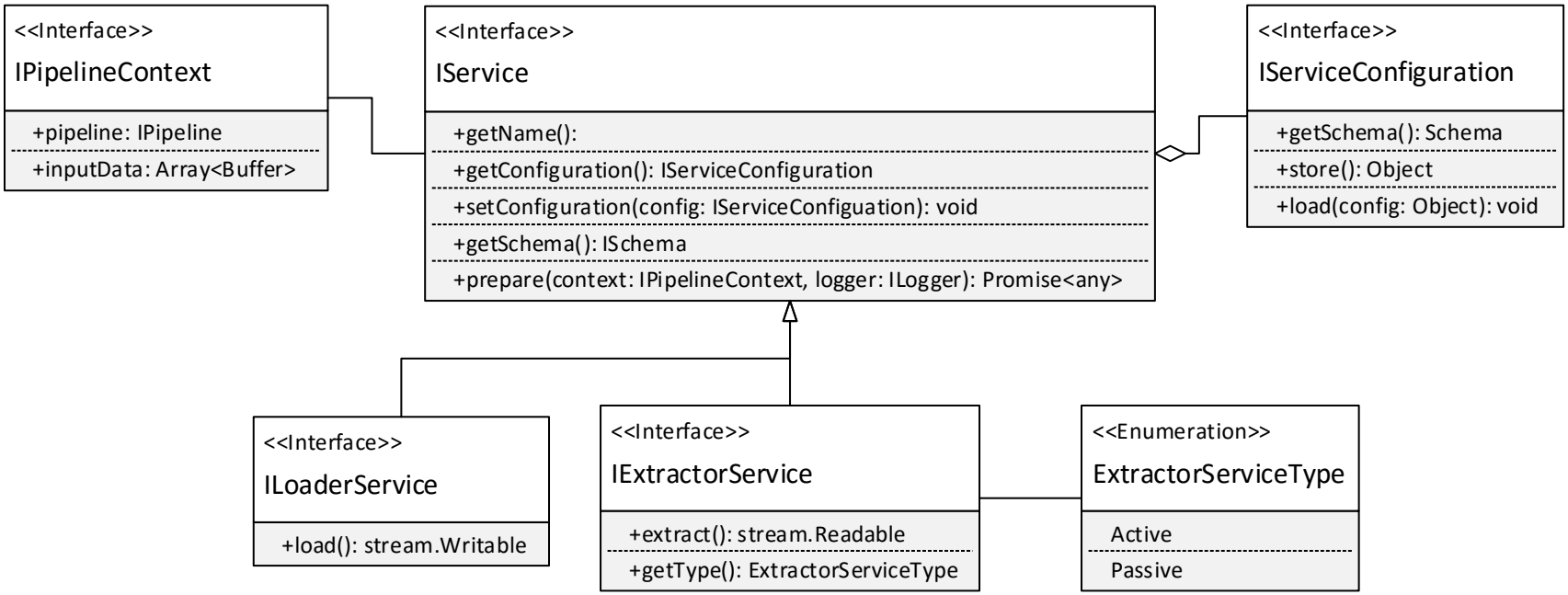
Developer workflow

The developer conducts the following steps create a new service:

1. Decide if the services should **extract** data from a system or if the service should **load** data into a system
2. Describe the provided/expected data using JSON-Schema
3. Define & describe configuration parameters using JSON-Schema
4. Implement the logic for extracting / loading the data
5. Test the implemented service using unit tests or the client application

Service extension layer

- The server application provides a predefined set of interfaces which the developer has to implement
- *node.js* stream API is used for the data flow between the services and the framework
- The application defines two different types of extractor services
  - **Active** extractors fetch all data self-sufficient (e.g. from a REST-API / RDMS)
  - **Passive** extractors need additional data at runtime to extract data



Evaluation
Server application

Set up

- 2 research assistants (RA) from SEBIS
- API documentation and “getting started” guide provided upfront
- Task: implement an extractor and a loader service using provided service extension layer.

Need for

- Better documentation about mapping format **(-)**
- Testing capabilities for service extension layer (Unit-Test, Mock-Objects) **(-)**
- Dynamic service configuration capabilities instead of static JSON-Schema **(x)**
  - Choose configuration value from a list of values
  - Depending values e.g. 1) Select Database 2) Select Table
- JSON-Schema dependent on dynamic configuration values **(+)**
  - Each table has a different schema

Evaluation
Client application

Set up

- 2 research assistants (RA) from SEBIS
- Open interviews
- Focus on usability, not implementation

Need for

- Improved navigation structure matching process of creating new pipelines **(+)**
- Improved mapping view to support dynamic schemata **(+)**
  - Select box with service's configurations
- Extended the configuration form to support **(x)**:
  - Custom JSON-Schema types
  - Dynamic (AJAX) loading of available configuration values (Select from values)
  - Linked input fields (Schema ↔ Tables)
- Improved usability of the mapping form by **(x)**:
  - Enabling semantic validation
  - Prefill the mapping form with required properties of the JSON-Schema
  - Visually connect properties
  - Indicate which properties are already mapped in the schema visualizer

Conclusion

- Roles and corresponding UCs were elicited
- Existing data integration tools were analysed
- REST-based server application with service extension layer were implemented
- Generic client application supporting configuration of pipelines were created
- Evaluation of both server and client application were performed

Future work

- Updating data, creating associations and orphan removal has to be handled by each service

  → Implement generic logic and provide it through the application's core
- Advanced mapping operations like aggregation or partitioning

  → Extend with DSL or JavaScript functions that can be applied to property mappings

Thank you for your attention.

Fridolin Koch

sebis

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

Boltzmannstraße 3
85748 Garching bei München

frido.koch@tum.de
wwwmatthes.in.tum.de