Informatics

# Entwicklung von Geschäftsapplikationen mit Low-Code-Ansatz - ER2CDS

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Software Engineering und Internet Computing**

eingereicht von

**Gallus Huber, BSc**
Matrikelnummer 51905700

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Wirtsch.Inf.Univ. Dr.rer.pol. Dominik Bork

Wien, 13. September 2024

Gallus Huber

Dominik Bork

Informatics

# Towards Low Code Business App Development - ER2CDS

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Software Engineering and Internet Computing**

by

**Gallus Huber, BSc**
Registration Number 51905700

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Wirtsch.Inf.Univ. Dr.rer.pol. Dominik Bork

Vienna, 13th September, 2024

Gallus Huber

Dominik Bork

# Erklärung zur Verfassung der Arbeit

Gallus Huber, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 13. September 2024

Gallus Huber

# Kurzfassung

Diese Arbeit präsentiert ER2CDS, einen neuartigen Ansatz für modellgetriebene Entwicklung von SAP Core Data Services (CDS). ER2CDS adressiert die Notwendigkeit einer höheren Abstraktionsebene in der CDS-Entwicklung und ermöglicht, mit Hilfe einer domänenspezifische Sprache, textuelles, sowie grafisches Modellieren von CDS-Views. Basierend auf Webtechnologien und dem Language Server Protocol (LSP) unterstützt ER2CDS das hybride Modellieren von CDS, sowie den Import bestehender SAP CDS-View-Entities. Dieser modellgetriebene Ansatz zielt darauf ab, Domänenexperten einzubeziehen und diesen zu ermöglichen CDS-View-Entities selbstständig zu entwickeln. Diese Arbeit beschreibt die Entwicklung der domänenspezifischen Sprache (DSL) von ER2CDS, sowie die Implementierung von ER2CDS. ER2CDS wird in Form eines Experiments und einer Fallstudie mit Domänenexperten und CDS-Entwicklern evaluiert. Die Ergebnisse zeigen eine hohe Benutzerfreundlichkeit des Tools, sowie die Bereitschaft von Domänenexperten und CDS-Entwicklern dieses produktiv zu nutzen.

# Abstract

This thesis introduces ER2CDS, a novel approach and tool support for the model-driven engineering of SAP Core Data Services (CDS). ER2CDS addresses the need for a higher abstraction level in CDS development, enabling blended, i.e., textual and graphical modeling of CDS Views through a domain-specific language. Based on web technologies and the Language Server Protocol (LSP) modeling tool support for ER2CDS is realized. The tool supports the hybrid modeling of CDS and the import of existing SAP CDS view entities for analysis and development support. This model-driven approach aims to broaden the target audience of CDS development by including and enabling domain experts to develop CDS views, mitigating the need for extensive programming skills. We report on the development of the ER2CDS domain-specific language (DSL) and the implementation of the corresponding ER2CDS modeling tool. Finally, ER2CDS is evaluated in the form of a controlled experiment and a case study with domain experts and CDS developers. The results show a high usability score for our tool and a willingness by domain experts and CDS developers to use it.

# Contents

CHAPTER 1

# Introduction

This chapter addresses the thesis's motivation and problem. It also outlines the work's aim and concrete research questions. Then, the methodological approach is described, and an overview of the structure is given.

## 1.1 Motivation and Problem Statement

Compared to code, models allow different stakeholders to get a technical understanding of an application without prior programming knowledge. Furthermore, models are a critical tool in managing the complexity of applications [MBWM23]. Combined with model-to-text transformations, a model-driven engineering process allows the creation of all different kinds of applications [HRW11].

Especially in the context of business informatics, business experts drive the development process. Allowing them to create applications independently by applying a model-driven engineering process would provide a significant business impact. On the one hand, business experts can quickly create domain-specific applications without extensive preparation, and on the other hand, a reduced complexity for developers. Furthermore, maintaining applications with a graphical user interface leads to a better understanding and fewer errors [CTVW19, DLP+23].

One of the most used enterprise solutions is the Enterprise Resource Planning (ERP) solution by SAP SE (SAP) [Sta]. It allows developers to create database views with enhanced access functions, so-called Core Data Services (CDS) [SEa]. Furthermore, it already allows the generation of a front-end application for an existing CDS [SEo]. However, CDS are currently developed in a textual editor, requiring skills in SQL and additional CDS-specific syntax [SEa]. It is, therefore, impossible for users with no programming background to create CDS themselves. Currently, the development process of CDS is twofold. First, the business expert states the needs and the data that should be retrieved. Then, a developer has to implement the CDS accordingly. One of the significant problems with this approach is to get a shared understanding of the problem domain. This leads to unsatisfactory results, as well as long development times.

## 1.2 Aim of Work

This work aims to create a new modeling tool based on the Language Server Protocol (LSP) [BL23, Micf] for Entity-Relationship (ER) diagrams [Che76a], which allow users to transform ER models into CDS. Furthermore, the goal is to assess the business value of such a low-code framework on a real-world application. Our evaluation is twofold and composes a controlled experiment—focusing on the import/export features and the SAP integration—and a case study with practitioners—focusing on the usability of the ER2CDS tool. ER2CDS should contribute to embracing model-driven engineering processes in the SAP ecosystem. The main goal is to provide a tool that allows business experts and developers to create CDS more efficiently.

First, we will develop a hybrid tool to model ER diagrams in a textual and graphical manner similar to the one proposed in [GB21]. The main focus is extending the ER modeling language [Che76a] to cover all aspects of CDS and the integration into the SAP ecosystem. In more detail, CDS supports specific features, e.g., *associations*, join relationships executed on demand, which should be represented in the modeling tool, as this information is necessary for the transformation step. Additionally, the modeling extension should represent the data model of the SAP ERP solution and allow users to select existing entities to build their models. In particular, the SAP system's database schema, consisting of tables and the corresponding columns, should be represented in the tool to ensure ease of use and the correctness of the models.

Furthermore, the resulting ER model will be transformed. The focus is hereby on a text output, following best practices of CDS development. The correctness and quality of code are some of the most important factors in ensuring user acceptance.

Concretely, the following research questions will drive the research and development efforts of the new ER2CDS tool and need to be answered.

- **RQ1: To what extent does a model-driven engineering process improve the development of Core Data Services?**
  - ER2CDS should provide an alternative to the text-based development process of Core Data Services and simplify the process by using models as an abstraction mechanism.
- **RQ2: What is an appropriate means to model Core Data Services?**
  - To support the modeling of Core Data Services, a novel modeling language has to be developed.

- **RQ3: Which extensions to the Language Server Protocol are necessary to support hybrid textual and graphical modeling for Core Data Services?**
  - ER2CDS should provide a textual as well as a graphical editor. The Language Server Protocol was originally designed for textual editors [NN22] and must therefore be extended.
- **RQ4: What are the advantages and disadvantages of the integration of external data sources to a web-based modeling tool?**
  - To improve the modeling process and extend the validation methods, an existing data model should be integrated into ER2CDS.
- **RQ5: To what extent can a Core Data Service be generated from a model which is based on the novel Core Data Service modeling language?**
  - ER2CDS should allow generating a valid Core Data Service from a given model. Therefore, a model-to-text transformation has to be designed and implemented.

## 1.3 Methodology

This thesis will follow the methodological approach of Design Science Research (DSR) [HMPR04]. Adhering to the seven guidelines of DSR, Design an Artifact, Problem Relevance, Design Evaluation, Research Contribution, Research Rigor, Design as a Search Process, and Communication of Research, the following steps are derived:

1. **Existing Tools Evaluation & Requirements Analysis**
   CDS is constantly evolving, and the current state-of-the-art for developing CDS has to be explored. Furthermore, requirements can be directly derived from the features of existing tools. Additionally, best practices regarding the textual syntax of CDS have to be gathered. This step is necessary to answer **RQ1** and serves as a basis for the next steps.
2. **Conceptualization**
   Based on the results from the previous step, a concept for ER2CDS is developed. First, the modeling and tool requirements are elaborated. Then, the extended ER-modeling language's concrete textual and graphical syntax is defined **(RQ2)**. The concept should also include a specification of the LSP extension **(RQ3)**. Furthermore, a strategy to handle hybrid modeling and the synchronization between textual and graphical editors is determined. Also, the integration of an external data model is defined in this step **(RQ4)**. Finally, the model-to-text transformation for ER-diagrams to CDS is defined **(RQ5)**.
3. **Artifact Implementation**
   A novel modeling tool, ER2CDS, will be created based on the conceptualization.

4. **Artifact Evaluation**
   To evaluate ER2CDS an observational and experimental method will be used [HMPR04]:

   a) **Controlled Experiment**
      The experiment will evaluate the text output (CDS) resulting from the model-to-text transformation. Existing CDS will be modelled with ER2CDS. Afterward, a CDS will be generated with the tool, which will then be checked for syntactical correctness and the correctness of the query output.

   b) **Case Study**
      A case study will be used to evaluate the artifact in a business environment. Participants are asked to implement three different CDS with ER2CDS. Afterward, the group of participants are surveyed to gather feedback and insights.

   In combination with the first step, both methods contribute to answer **RQ1**.

## 1.4 Structure

This thesis is organized into the following chapters:

**Chapter 2** presents all the necessary background information that the thesis will build upon. Model-Driven Engineering (MDE) and Web-based modeling will be introduced. Furthermore, an overview of the SAP environment will be given. Finally, Entity Relationship and SAP Core Data Services are explained in detail.

**Chapter 3** discusses related work of this thesis. First, other existing web-based modeling tools are analyzed, and the results are discussed, in particular the contributions of two existing solutions, bigER [GB21] and bigUML [MB23]. Furthermore, the latest findings in the Graphical Language Server Protocol (GLSP) [Foub] are analyzed. At the end of Chapter 3, the latest development projects regarding Core Data Services are presented.

**Chapter 4** will elaborate requirements for ER2CDS. Currently, CDS are developed using ABAP Development Tools (ADT) in Eclipse [SEa]. The requirements for ER2CDS will be directly derived from this. Furthermore, additional modeling requirements and special tool requirements are presented.

In **Chapter 5**, the main concepts for ER2CDS are developed. The specifications for the textual and graphical syntax, as well as for the LSP extension, are presented. Furthermore, details regarding hybrid modeling and the synchronization of the textual and graphical model can be found in Chapter 5. Also, an overview of the integration of external data models is given. Finally, the resulting model-to-text transformation for ER to CDS will be discussed in this section.

**Chapter 6** discusses the artifact's implementation details. It presents the overall architecture, the used technologies, and the main components of ER2CDS. Finally, it showcases the realization of specific components and the tool's behavior.

**Chapter 7** evaluates the developed artifact. The main methods for evaluating ER2CDS are described, and the results are presented.

**Chapter 8** summarizes the thesis and the contributions. The research questions are answered, and an outlook for future research is given.

# CHAPTER 2

# Background

In this chapter, the terminology and background of this thesis is introduced. First, model engineering, model-driven software engineering, and model-to-text transformations are presented. Then, the theoretical foundation for web-based modeling is introduced. An overview of the Language Server Protocol will be given, as well as Langium and Sprotty as frameworks for the implementation of ER2CDS and the characteristics of Visual Studio Code as the targeted editor for ER2CDS. The following section will present all necessary background regarding the SAP environment. This includes an introduction to SAP Enterprise Resource Planning, SAP HANA, and Business Application Studio. SAP Core Data Services and Entity Relationship will be introduced at the end of this chapter since these concepts are the building blocks of the novel modeling tool developed in this thesis.

## 2.1 Model Engineering

While the human mind is continuously reworking reality, the abstraction mechanism is one of the most essential processes among them [BCW12]. As humans, we create a mental representation by finding commonalities in different observations [BCW12]. We can generalize specific features of natural objects (**generalization**), classify the objects into coherent clusters (**classification**), and aggregate objects into more complex ones (**aggregation**) to generate this mental representation [BCW12]. Furthermore, abstraction is widely used in science. For example, from the early days of computing, different languages and platforms helped developers to understand applications and manage their complexity without programming with machine code or directly on hardware [S$^+$06]. We often refer to this abstraction mechanism as *modeling* [BCW12].

### 2.1.1 Models

Models are critical tools for humans to manage complexity, especially in developing software and data-intensive systems [MBWM23]. In software engineering, a model represents a running system by abstracting details [WHR14]. Informally defined, one can say that models are a simplified and partial representation of reality [BCW12]. However, there are also multiple, more formal definitions of models, e.g.,

> *A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system.*[BG01]

> *A model is a description of a (part of) systems written in a well-defined language. A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer.*[KWB03]

However, by all these definitions, a model will only describe some of the details of reality. More detailed models implement three features by applying abstraction:

- **Mapping Feature**: Models are based on an original system, which is abstracted and generalized.
- **Reduction Feature**: Models focus on aspects of interest and reflect only the relevant selection of the properties from the modeled system.
- **Pragmatic Feature**: Models can be used in place of an original concerning some purpose.

Additionally, each model can have a different purpose. In [BCW12], models are classified into having *descriptive purpose*, to represent the reality of a system or *prescriptive purpose*, to define the scope of a system or the depth of detail a problem shall be studied. Additionally, with the increasing importance of data analysis, the *predictive purpose* of models to predict and generate yet-to-be-seen data, as described in [KJ+13], became more and more critical.

In this work, we will use models as the primary abstraction mechanism to handle the complexity of low-level applications, such as Core Data Services, which will be introduced in Section 2.4.

### 2.1.2 Model-Driven Software Engineering

Since software is becoming exponentially more complex, it is generally agreed that in order to handle this, software should be developed by using appropriate abstraction mechanisms [WHR14]. The current state-of-the-art establishes Model-Driven Engineering (MDE) and especially Model-Driven Software Engineering (MDSE) as the primary abstraction methodology [WHR14]. In the context of this work, we will further focus on MDSE, as one of the main contributions is a tool for software development. In MDSE, *"everything is a model"* is one of the most important statements, and therefore, models are used as the central artifact of software development [BCW12]. In other words, models are used as first-class citizens [BCW12].

The two main concepts of MDSE are *models* and *transformations*. While we already defined models, transformations can be seen as operations performed on a model. To further illustrate the importance of these two concepts, the authors in [BCW12] use them to recreate the famous equation of Niklaus Wirth:

$$\textit{Algorithms} + \textit{Data Structures} = \textit{Programs}$$

Using the two main concepts of MDSE:

$$\textit{Models} + \textit{Transformations} = \textit{Software}$$

The models and transformations are expressed in some notation, called *modeling language* [BCW12]. Applying the core statement *"everything is a model"*, the definition of such modeling languages, so-called *meta-modeling language*, can be again seen as a model, creating a hierarchical pattern [BCW12].

In Fig. 2.1, the resulting architecture of MDSE is presented. It can be divided into conceptualization (columns), dealing with the definition of models that describe the reality and implementation (rows), dealing with the creation of running components from models [BCW12]. The layers are briefly described in the following:

**Conceptualization**

- **Application**: Models are created and transformed, resulting in running components.
- **Application domain**: Modeling language, transformations, and implementation platforms are defined.
- **Meta-Level**: Meta-modeling language and transformation language are defined.

**Implementation**

- **Modeling**: Definition of models.
- **Automation**: Mapping of models to artifacts.
- **Realization**: Running artifacts.

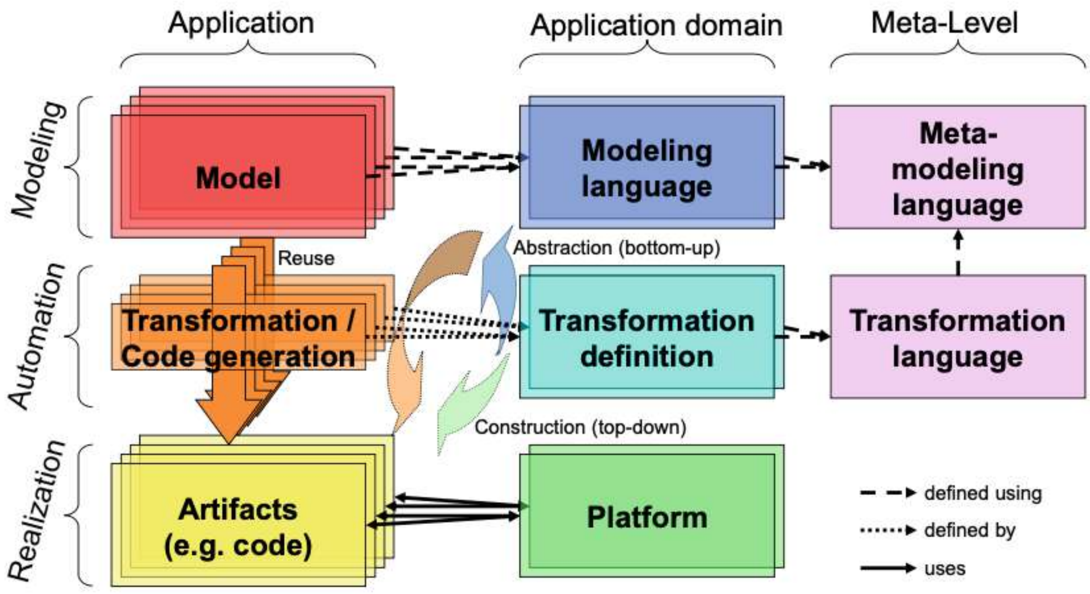


Figure 2.1: Model-Driven Software Engineering Architecture [BCW12].

This work will contribute to MDSE by creating a domain-specific modeling language (DSML) and a transformation on the application domain level. The transformation will be implemented as a model-to-text transformation, further introduced in Section 2.1.3.

### 2.1.3 Model-to-Text Transformations

As stated above, *transformations* can be seen as operations performed on models. One can also interpret transformations as a mapping between different models [BCW12]. In Fig. 2.2, the architecture of such a transformation is presented. The model **Ma** confirms to the meta-model **MMa**, which furthermore confirms to the meta-meta-model **MMM**. In order to create a transformation from model **Ma** to **Mb**, one needs to define a transformation **MT** which uses the meta-model **MMa** and meta-model **MMb** to map the model **Ma** to model **Mb**. The transformation itself is again defined in a meta-transformation language **MT**, which also conforms to **MMM** and is executed by the **MT Execution Engine**, which reads the source model, applies the transformation and writes the target model.

In the context of MDSE, model transformations are used to either map one model to another or to generate code as illustrated in Fig. 2.1. We can, therefore, divide model transformations into two top-level categories: *model-to-model* transformations and *model-to-text* transformations [BCW12, CH+03]. In this work, we will focus on model-to-text transformation, as this will be one of the contributions of this thesis and a part of the novel modeling tool. Furthermore we can differentiate model-to-text by the approach used to generate the text output [BCW12, CH+03]:

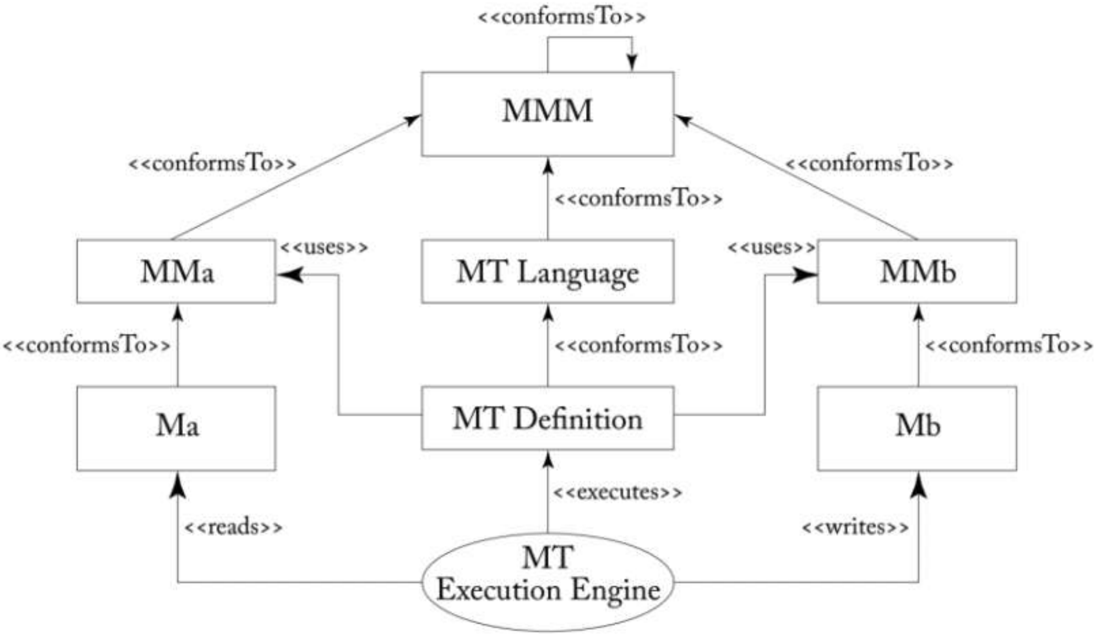


Figure 2.2: Architecture of a model transformation[BCW12].

- **Visitor-Based Approach**: A generic approach in which the model is traversed and the code is written to a text stream.

- **Template-Based Approach**: Use templates with code snippets enriched with model information.

While both approaches can be used to generate text from a given model, most of the existing transformation tools use the template-based approach, e.g., Xtend[1], JET[2], MOFScript[3], Acceleo[4]. In [HKGV10], another approach to generate code from models is presented. The authors propose a novel technique of *code generation by model transformation*, where a first-order term target model is created, which then can be serialized by term rewriting. In addition, the authors of [BCD10] propose a *graph-based transformation approach*, which can also be used to generate text output from models.

In this work, we will use the template-based approach. Therefore, a more detailed definition is presented below. A *template* consists of *text fragments* and *meta-markers*, where text fragments are static text that is used in the output as is, and meta-markers are

[1] `https://eclipse.dev/Xtext/xtend/`, last accessed: 04.08.2024

[2] `https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=extensions-creating-jet-transformations`, last accessed: 04.08.2024

[3] `https://umt-qvt.sourceforge.net/mofscript/docs/MOFScript-User-Guide.pdf`, last accessed: 04.08.2024

[4] `https://wiki.eclipse.org/Acceleo/User_Guide` , last accessed: 04.08.2024

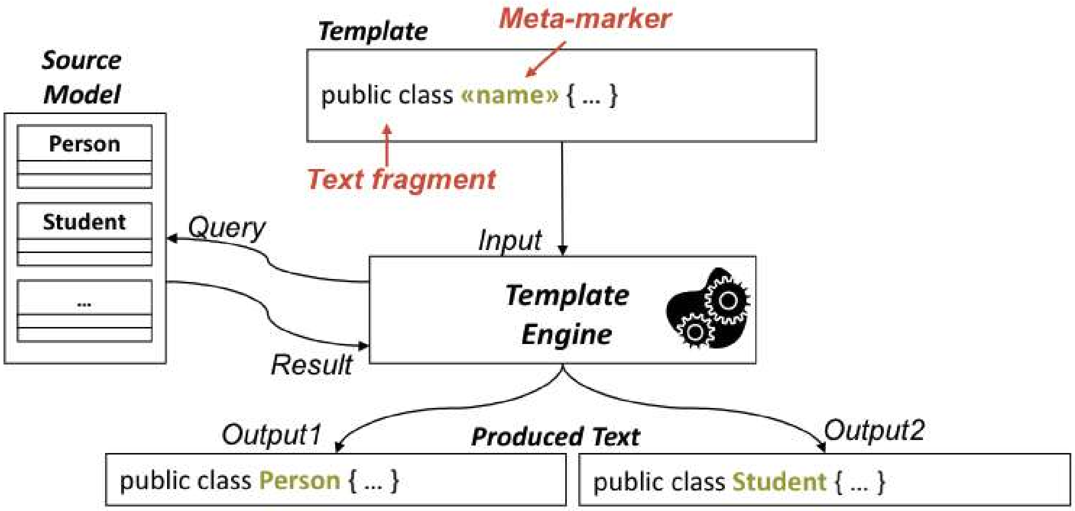


Figure 2.3: Architecture of a template-based model-to-text transformation[BCW12].

used to query additional data sources, in our case the source model of the transformation [BCW12]. The *template engine* is then responsible for replacing the meta-markers with the actual data of the data source and producing the output. The architecture of the template-based model-to-text transformation is shown in Fig. 2.3

## 2.2 Web-based Modeling

In recent years, web-based modeling tools emerged to create more flexible and richer graphical user interfaces using the web technology stack [MBWM23, DCLB22]. Furthermore, one of the most essential advantages of web-based modeling tools is the browser-based client, which results in platform independence. Moreover, mainstream editors, e.g., Visual Studio Code, allow the integration of HTML components into the editor, allowing for even more seamless integration of web-based modeling tools [Micg].

### 2.2.1 Language Server Protocol

One of the most crucial building blocks of web-based modeling is the Language Server Protocol (LSP), created by Microsoft [BL23, Micf]. A *language server* can be defined as a server that can provide language-specific smarts [Mich]. LSP was created to standardize a language-neutral communication between a language server and specific development tools to enable the reuse of the language server for multiple development tools [Mich]. The language server and the development tool run in different processes and communicate over JSON-RPC, which makes this architecture highly flexible and enables diverse deployment opportunities [BLO23, RIWC18, Mich]. The basic concepts of LSP are presented in the following[Micf]:

- **Request Message**: A request between client and server. A response must be sent back to the sender for each request.
- **Response Message**: Sent to the client as a result of a request. The response must always be sent, even if a request does not have a result. The result property of the response message is then set to *null*.
- **Notification Message**: A message where no response message needs to be sent after processing.
- **Cancellation Support**: A message allowing cancellation of a request. Still, a response message must be returned, but partial results can be returned.
- **Progress Support**: A message that can report progress and return partial results.

Since not all features are supported by each development tool or language server, LSP uses so-called capabilities to determine the set of supported features [Micf]. The client and server exchange its capabilities during the *initialize* life-cycle message [Micf]. A further definition of life-cycle messages is provided next.

Life-cycle messages allow the client to handle the language server life-cycle [Micf]. A list of all life-cycle messages is provided below:

- **Initialize Request**: First request sent from the client to the server. The server responds with an *InitializeResult* response.
- **Initialized Notification**: Sent once to the server after the client receives the *InitializeResult*.
- **Register Capability Request**: Sent to the client to dynamically register capabilities.
- **Unregister Capability Request**: This is sent from the server to the client to unregister a previously registered capability.
- **Set trace Notification**: This can be sent from the client to the server to modify the server's trace setting.
- **Log trace Notification**: Sent from the server to the client to log the server's trace.
- **Shutdown Request**: Sent from the client to the server to ask the server to shut down. The server does not exit so that the response can be delivered correctly. Afterward, the client sends an *Exit* notification.
- **Exit Notification**: Sent from the client, asking the server to exit the process. Exit successfully if a *Shutdown* request was received prior; otherwise, exit with an error.

Furthermore, LSP specifies a set of messages to handle document synchronization. A list of all messages can be found in [Micf]. In the following, the three mandatory notifications of the protocol are presented [Micf]:

- **Did Open Text Document Notification**: Sent from the client to the server to signal a newly opened text document. Afterward, the tool manages the document's truth, not the file system. An opened document does not necessarily be visible in the editor.

- **Did Change Text Document Notification**: Sent from the client to the server to signal changes to the document. It can only be sent after the *Did Open Text Document* notification.

- **Did Close Text Document Notification** : Sent from the client to the server to signal the closing of a document. The truth is not handled in the tool anymore.

Additionally to the life-cycle and document synchronization messages, LSP defines even more messages for different types of language, workspace, and window features, for which a detailed specification can be found in [Micf].

To further illustrate the usage of LSP Fig. 2.4 presents an example of the communication between the language server and a development tool. This example provides insights into the process of a user opening and editing the file, executing the "Goto definition" functionality, and eventually closing the file. The process of Fig. 2.4 starts with the development tool sending a *textDocument/didOpen* notification to the language server. It is important to note that from there on, the truth of the file is stored in the internal memory of the tool and not on the file system anymore [Mich]. In the next step, the user edits the document, and a *textDocument/didChange* notification is sent to the language server, with the changes as parameters. The language server uses these modifications to update the internal representation of the document. Furthermore, the current state is analyzed, and errors and warnings are emitted to the development tool using the *textDocument/publishDiagnostics* notification. In the next step, the user executes the "Goto definition" functionality of the development tool. In order to resolve the location of the requested definition, the development tool sends a *textDocument/definition* request to the language server, which responds with *textDocument/definition* and the result. The user closes the document at the end, and a *textDocument/didClose* notification is sent to the language server.

As an example, Listing 2.1 presents the payload for the *textDocument/didOpen* notification.

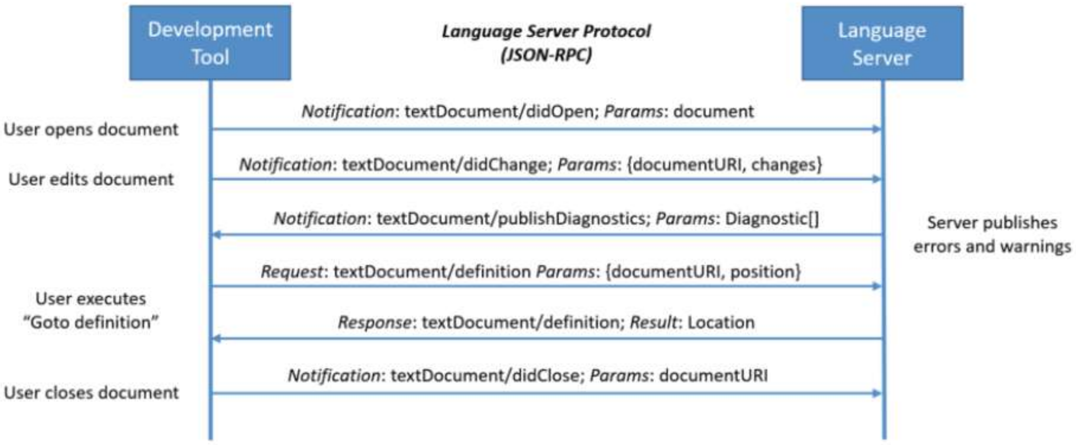

Development Tool
Language Server Protocol (JSON-RPC)
Language Server
User opens document
Notification: textDocument/didOpen; Params: document
User edits document
Notification: textDocument/didChange; Params: {documentURI, changes}
Notification: textDocument/publishDiagnostics; Params: Diagnostic[]
Server publishes errors and warnings
Request: textDocument/definition Params: {documentURI, position}
User executes "Goto definition"
Response: textDocument/definition; Result: Location
Notification: textDocument/didClose; Params: documentURI
User closes document

Figure 2.4: Example of the communication between language server and development tool[Mich].

```json
{
    "jsonrpc": "2.0",
    "id" : 1,
    "method": "textDocument/didOpen",
    "params": {
        "textDocument": {
            "uri": "file:///example/example.er2cds"
        }
    }
}
```

Listing 2.1: Payload of textDocument/didOpen notification.

LSP also allows the definition of custom messages, which will be used in this thesis to extend the standard protocol in order to support hybrid modeling of Core Data Services [Micf].

### 2.2.2 Langium

Langium[5] is a open source framework *"with first-class support for the Language Server Protocol, written in Typescript and running in Node.js."* [Typd]. Langium provides the possibility to create domain-specific languages together with an out-of-the-box Typescript-based language server that can be easily integrated into VS Code as an extension or other web applications and can be arbitrarily customized to meet the language creators' needs. With its pre-built implementations, Langium simplifies language tasks such as parsing, Abstract Syntax Tree (AST) generation, validation, scoping, cross-referencing,

[5] `https://langium.org/`, last accessed: 30.07.2024

and more. The effectiveness of creating Langium-based modeling tools has been shown in several recent works [BLRB24, GGC22, Pet22, PKvH23, PLV23]. The broad spectrum of features will be analyzed in the following.

Langium allows to define a custom language using the *Langium Grammar* language [Typb]. Furthermore, it can generate a parser for the defined language, transforming an input string into an abstract syntax tree (AST) [Typb]. Additionally, the framework generates type definitions in TypeScript, which allow to traverse the AST in a type-safe manner [Typb]. The language declaration starts with a header, declaring the name of the language as shown in Listing 2.2 for the language named *MyLanguage* [Typc].

```
grammar MyLanguage
```

Listing 2.2: Langium grammar language declaration.

*Lexing* defines the first step of the parsing process, in which the input string is transformed into a stream of tokens, each matching a *terminal rule* as shown in the first two lines of Listing 2.3 [Typc]. Langium supports defining these rules using the Extended Backus-Naur Form (EBNF) Expressions and Regular Expressions [Typc]. These tokens are atomic and have to return a primitive TypeScript type such as *string*, *number*, *boolean*, *bigint* or *Date* [Typc]. Due to the lexer trying to match each character in the input string to a terminal rule, one can define *hidden terminal rules*, which are ignored during processing [Typc]. An example of these can be found in the last three lines of Listing 2.3.

```
terminal ID returns string: /[_a-zA-Z][\w_]*/;
terminal INT returns number: /[0-9]+/;

hidden terminal WS: /\s+/;
hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

Listing 2.3: Langium grammar terminal rules.

In the next processing step, the *parser* creates the AST using defined sequences of tokens [Typc]. These valid sequences of tokens are defined using *parser rules*, which are written using EBNF [Typc]. A simple *parser rule* is shown in lines 1 and 2 of Listing 2.4. This rule will create an object of type *Person* with an attribute *name* that matches the terminal rule *ID*. It is important to note that *'person'* is a keyword of the defined language and interpreted as an *inline terminal rule* [Typc].

Furthermore, the langium grammar allows to define an *entry rule*, as shown in lines 4 and 5 of Listing 2.4, which serves as the starting point for the parser [Typc]. For this example, the parser will try to parse objects of type *Person* or *Greeting* and add them to the *persons* and *greetings* array. Moreover, the grammar allows using cardinalities, such as *?* for zero or one, *** for zero or many and *+* for one or many, as well as alternatives using the / operator in these *parser rules* [Typc].

```
Person:
    'person' name=ID;

entry Model:
    (persons+=Person | greetings+=Greeting)*;
```

Listing 2.4: Langium grammar parser rules.

In addition, Langium supports to define *cross-referencing* directly in the grammar [Typc]. In Listing 2.5, an example is given for such a *cross-reference*. The *Greeting* parser rule expects the keyword *'Hello'* followed by a string, which is equivalent to the name of an existing *Person* object and eventually followed by the keyword *'!'*.

```
Person:
    'person' name=ID;
Greeting:
    'Hello' person=[Person:ID] '!';
```

Listing 2.5: Langium grammar cross-referencing.

By definition the following example in Listing 2.6 will be parsed successfully:

```
person Bob
Hello Bob !
```

Listing 2.6: Example for a valid cross-reference.

While the example in Listing 2.7 will produce an error, since there is no *Person* object with the *name* 'Laura' defined, even though 'Laura' is a valid token of type *ID*.

```
person Bob
Hello Laura !
```

Listing 2.7: Example for an invalid cross-reference.

An exhaustive list of features supported by the Langium grammar can be found in [Typc].

Another feature of Langium is the scoping mechanism [Typb]. Scoping eases the process of *linking*, which is used to resolve references between elements within the language [Typb]. One only needs to define the scoping behavior of the defined language, and the linking process will be done by the framework [Typb].

In order to get a deeper understanding of how the framework works, the document life cycle is presented below. The *LangiumDocument* serves as the central data structure of the framework with the primary purpose of holding the AST [Typa]. However, the *LangiumDocument* has to be built further after the input is parsed, which happens depending on the state of the *LangiumDocument* [Typa]. The possible states are listed

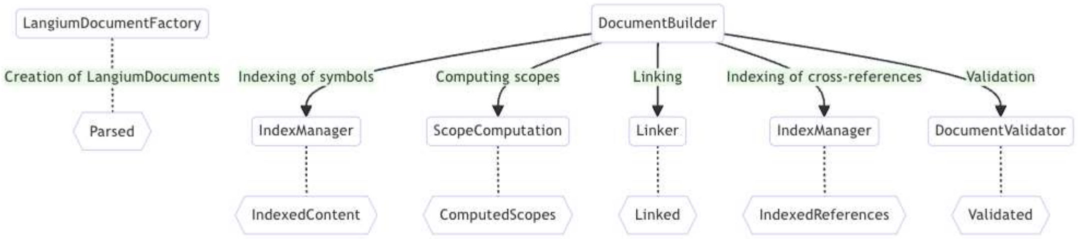


Figure 2.5: Build stages of a *LangiumDocument* [Typa].

below, with the first one being the initial state after the parsing process and the last one used to mark documents as invalid after a source text modification [Typa]:

1. **Parsed** AST has been created from the input string.
2. **IndexedContent** *IndexManager* has processed the AST nodes.
3. **ComputedScopes** *ScopeComputation* has created the local scopes.
4. **Linked** Cross-references were resolved by the *Linker*.
5. **IndexedReferences** *IndexManager* has indexed the references.
6. **Validated** *DocumentValidator* has validated the document.
7. **Changed** Document has been modified.

The build stages of a *LangiumDocument* depending on the state of the document is furthermore illustrated in Fig. 2.5.

Finally, Langium has first-class support for LSP [Typb]. This allows us to easily create a language server that can then be used by multiple development tools [Typb]. The framework supports multiple features of LSP out of the box but also allows the extension or implementation of custom capabilities of the language server.

This thesis will use the Langium framework to implement the language server of ER2CDS. Furthermore, the modeling tool's textual syntax will be defined using the Langium grammar.

### 2.2.3 Sprotty

Sprotty[6] is a framework designed for diagramming within web technologies [Fouc]. The framework is implemented in TypeScript and uses SVG renderings, supporting compatibility with all modern browsers [Fouc]. Furthermore, Sprotty supports integration with Langium and LSP, as well as Visual Studio Code [Fouc].

[6] `https://sprotty.org/`, last accessed: 04.08.2024

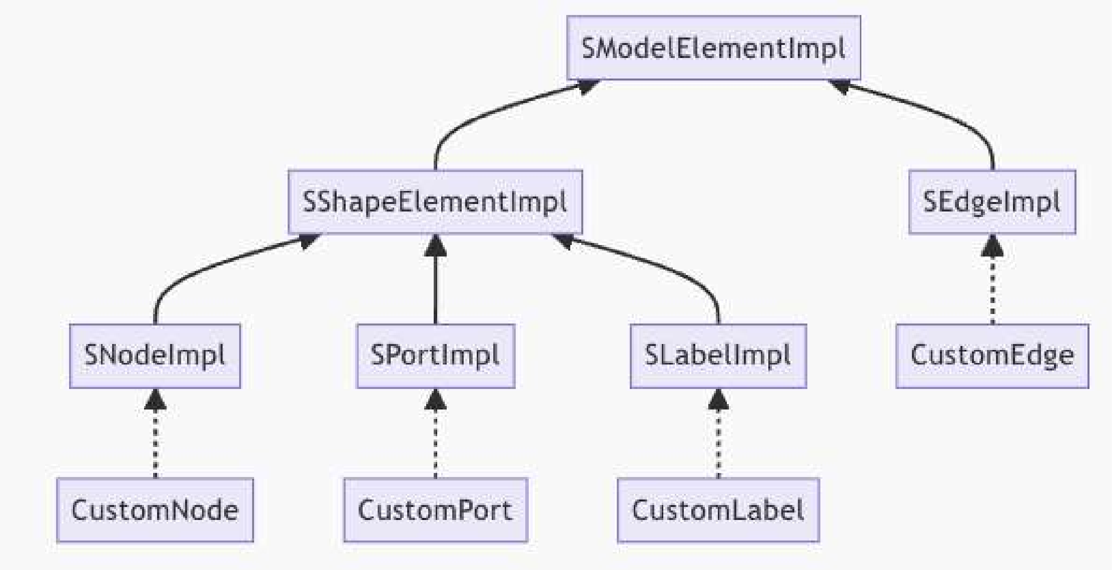


Figure 2.6: Tree structure of the Sprotty *SModel*[Foud].

The architecture of Sprotty consists of three main components: the *ActionDispatcher*, the *CommandStack*, and the *Viewer* [Foud]. In the following, a short overview of each component will be given.

The primary purpose of the *ActionDispatcher* is to receive an *Action*, a JSON structure that describes what should happen without defining any behavior, transform it to a *Command*, and transmit the resulting *Command* to the *CommandStack* [Foud]. Furthermore, the *ActionDispatcher* can communicate with the *ModelSource*, which represents either a local model or delegates to a remote model [Foud]. The *Action* can also be serialized and serve as an LSP message [Foud].

The *CommandStack* is responsible for executing a *Command* received by the *ActionDispatcher* [Foud]. A *Command* defines the behavior of the corresponding *Action* by implementing the methods *execute*, *undo* and *redo* and modifying the model respectively [Foud]. The *CommandStack* itself is responsible for keeping an undo/redo stack and producing a graph model, which is then forwarded to the *Viewer* for rendering [Foud]. The internal model used by Sprotty is called *SModel* and is organized in a tree structure, derived from the *parent* and *children* properties of each element [Foud]. The base tree structure of such a model can be seen in Fig. 2.6. This also illustrates the possibility of introducing domain-specific elements to the *SModel* [Foud].

The responsibility of the *Viewer* is to render the *SModel* it receives by the *CommandStack* [Foud]. To optimize the number of modifications to the Document Object Model (*DOM*), the translation is not performed directly, but by first creating a *VirtualDOM* and then patching the actual *DOM* [Foud]. The *VirtualDOM* is created by traversing the *SModel* and applying the corresponding *View* for each element, defined by the *ViewRegistry*

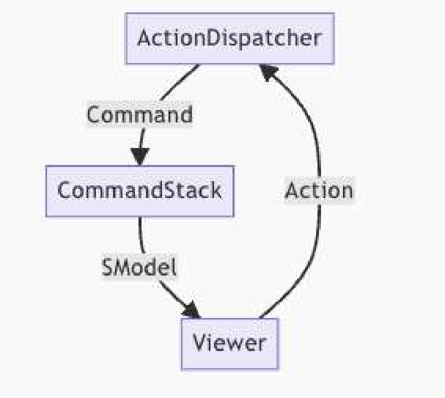


Figure 2.7: Architecture and information flow of the Sprotty framework [Foud].

[Foud]. Furthermore, the *Viewer* is responsible for attaching event listeners to the *DOM*, transforming the received events to *Action*s and eventually communicating them to the *ActionDispatcher* [Foud].

The resulting architecture and information flow of Sprotty is illustrated in Fig. 2.7.

This work will use Sprotty as a framework for the graphical modeling part of ER2CDS.

### 2.2.4 Visual Studio Code

Visual Studio Code (VS Code)[7] is a source code editor that offers built-in support for JavaScirpt[8], TypeScript[9] and Node.js[10] [Micc]. More importantly, VS Code offers an extension application programming interface (API) that allows customization of the editor itself [Micb]. The two capabilities this work will exhaustively use are its language extension [Micd] and webview API [Micg].

One of the main concepts in VS Code is *commands*, which can interact with the editor [Mica]. It also allows users to interact with functionality exposed by the extension [Mica]. For VS Code to expose extension commands in the editors *Command Palette*, the extension registers the Command as a contribution [Mica]. It is then possible to react on executed commands accordingly [Mica].

The language extension of VS Code offers declarative language features and programmatic language features [Micd]. The declarative language features offer support for the following language-specific features using configuration files [Micd]:

- Syntax highlighting

[7] `https://code.visualstudio.com/`, last accessed: 04.08.2024
[8] `https://js.org/`, last accessed: 04.08.2024
[9] `https://www.typescriptlang.org/`, last accessed: 04.08.2024
[10] `https://nodejs.org/`, last accessed: 04.08.2024

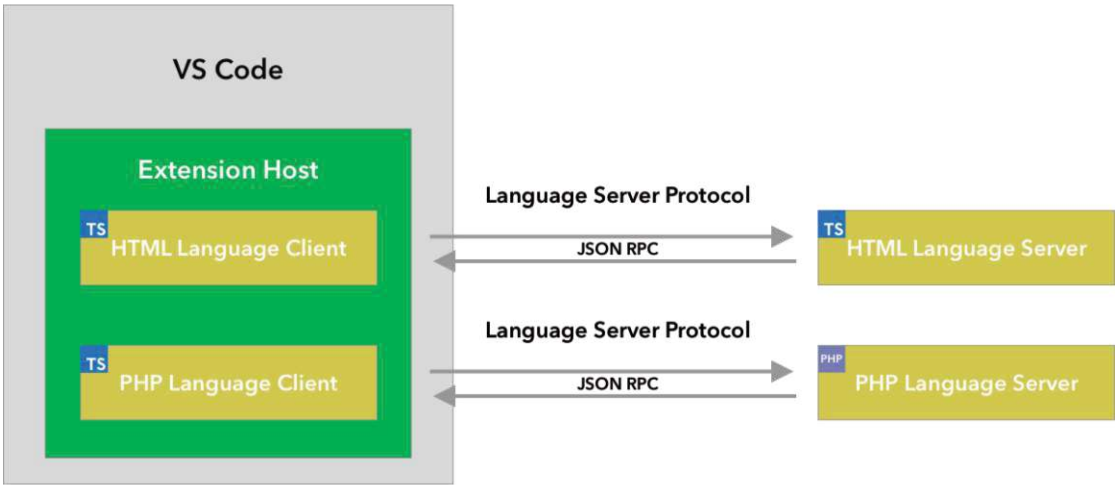


Figure 2.8: VS Code extension with two different language clients and language servers [Mice].

- Snippet completion
- Bracket matching
- Bracket autoclosing
- Bracket auto surrounding
- Comment toggling
- Auto indentation
- Folding (by markers)

For the programmatic language features, it is either possible to implement it using the language API directly or by utilizing a language server and LSP [Mice]. For the latter, VS Code will serve as the language client, which communicates to the language server using LSP, while the language server will run in a separate process [Mice]. Furthermore, it is possible to run multiple language servers simultaneously [Mice]. Due to the separation in processes, it is also possible to use different technologies for the specific language servers [Mice]. Fig. 2.8 illustrates an example of an extension with one HTML language client communicating with an HTML language server, both written in TypeScript, as well as a PHP language client communication with a PHP language server, where the language server is written using a different technology stack, in specific PHP.

Another feature of VS Code extensions is the webview API, which can be used to create custom views within VS Code [Micg]. *Webviews* can render HTML content and can be compared to an *iframe* [Micg]. *Webviews* can either be used as custom editors, for custom visualization or user interfaces, or as views rendered in the sidebar or panel areas of VS Code [Micg]. The possibility to render almost every HTML element enables the creation of highly customized views with web technologies, e.g., for diagramming [Micg].

In this thesis, ER2CDS will be developed as a VS Code extension, utilizing the language server extension for the textual modeling interface and the webview API to create a graphical modeling interface.

## 2.3 SAP Environment

SAP SE was founded in 1972 by five former IBM employees and specialized in developing programs for materials management, financial accounting, and auditing [O'R15]. Since 1980, the company has been based in Walldorf, Baden-Württemberg. By developing a standardized software solution, SAP SE is regarded as one of the inventors of standard software [O'R15].

The first standard solution was presented in 1973 as SAP R/98. In 1979, R/2 replaced its predecessor with improvements in material management and production planning [O'R15]. In 1992, R/3, based on a client-server architecture for the first time, was introduced. SAP ERP later replaced R/3 [O'R15]. Since 2015, SAP SE's main product line has been SAP Business Suite 4 SAP HANA (S/4HANA) [SEr]. In 2024, SAP SE will be one of the largest software companies, with a turnover of 31.2 billion euros in 2023 [SEq].

### 2.3.1 SAP S/4HANA

In 2015, SAP Business Suite 4 SAP HANA, or S/4HANA for short, was introduced, replacing SAP ERP. The switch to the new software generation includes the mandatory introduction of the HANA database, an in-memory database developed by SAP SE, which will be introduced in the next section, and a new user interface, SAP Fiori [Sch16]. Similar to its predecessor, S/4HANA is a standardized enterprise resource planning (ERP) system, which allows for customization as well as extension of the standard software [SEs]. Furthermore, it is possible to develop custom applications using the integrated technologies, e.g., Core Data Services, which will be introduced in Section 2.4. In this thesis, we will need S/4HANA as the targeted system for the ER2CDS output and an optional backend component for ER2CDS.

### 2.3.2 SAP HANA

SAP HANA (High-performance Analytic Appliance) is an in-memory database designed for online analytical processing (OLAP), as well as online transactional processing (OLTP) [FML+12]. The in-memory design of SAP HANA allows for much faster queries while still complying with Atomicity, Consistency, Isolation, and Durability (ACID) standards [FCP+12]. However, the database still uses persistent storage supporting system restart and recovery [FCP+12]. The database provides standard interfaces and supports extended SQL [FCP+12]. An overview of the architecture is illustrated in Fig. 2.9.

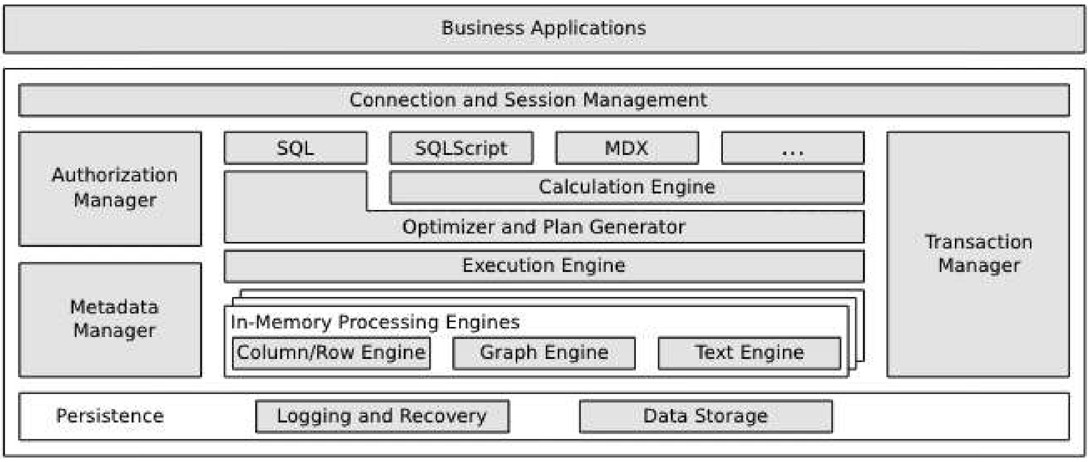


Figure 2.9: Architecture of the SAP HANA database [FCP+12].

SAP HANA supports CDS natively [Sar23]. This thesis will use SAP HANA indirectly by creating a novel modeling language for CDS and using CDS to expose the data model of an existing S/4HANA system for ER2CDS.

## 2.4 ABAP Core Data Services

A virtual data model (VDM) can provide a semantically rich, reusable, and stable data model while abstracting technical details and reducing complexity [Sar23]. Furthermore, in business informatics, a VDM can provide a business-oriented view of the underlying database [Sar23]. In the SAP S/4HANA, the VDM is implemented using Core Data Services (CDS) [SEc]. One of the main ideas behind CDS is to support a data-centric approach, pushing down computations to the database layer [SEc]. In detail, CDS consists of the following languages, enabling one to create these semantically enriched data models [Sar23]:

- **Data Definition Language (DDL)**: Allows to create domain-specific data models, CDS entities.
- **Query Language (QL)**: Allows to read data from defines models.
- **Date Control Language (DCL)**: Allows to control access to the data model.
- **Data Manipulation Language (DML)**: Allows to write data.

Furthermore, the CDS DDL allows to build hierarchies, which serves as a building block of the SAP VDM [SEp]. The VDM is presented in Fig. 2.10 and can be divided into three layers [SEt]:

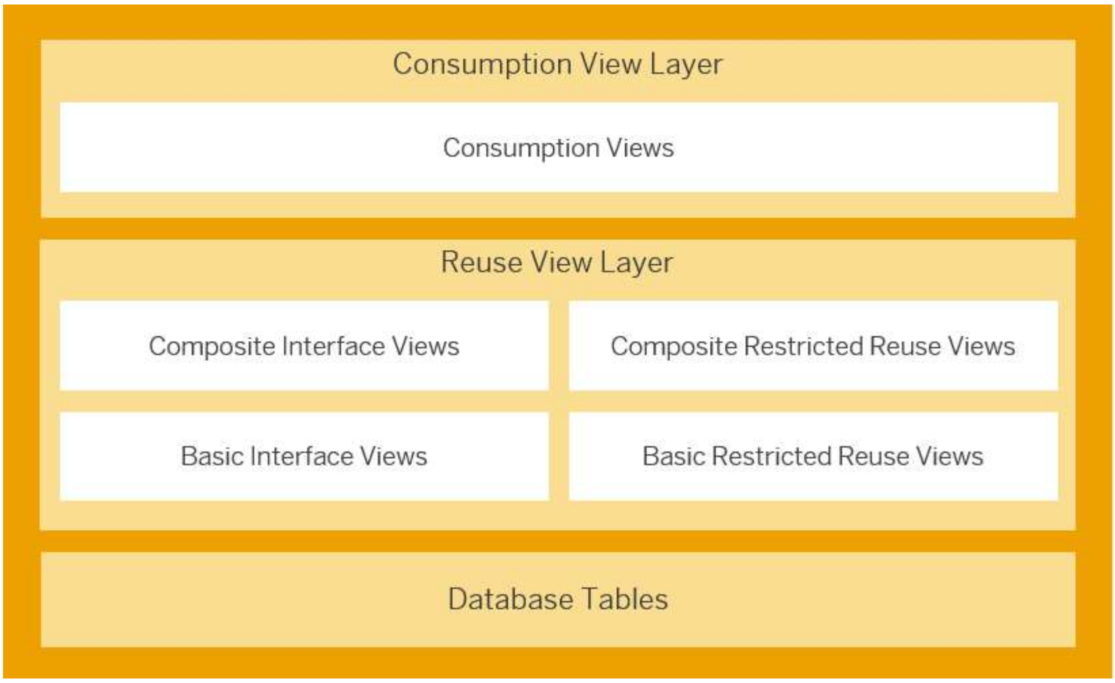


Figure 2.10: Layers of the VDM [SEt].

- **Consumption view layer**:
  - Consumption views: Placed on top of reuse views and built for a particular purpose and specific requirements. Database access is only provided indirectly through the reuse view layer.
- **Reuse view layer**:
  - Basic interface views: The only views that directly access the database and are therefore placed on top of the database.
  - Composite interface views: Placed on top of basic interface views and can also have associations with other composite views.
  - Restricted reuse views: Similar to basic and composite interface views, but not intended for reuse.
- **Database Layer**: VDM is built on top of these database tables.

In the remainder of this work, we will focus on CDS's DDL since ER2CDS aims to generate views adhering to this DDL.

Technically, CDS DDL is an enhancement of the SQL DDL [Kel]. In addition to the standard SQL DDL, CDS supports entities with structured and custom-defined types, associations for joins with simple path expression, calculated fields that can be predefined in the data model and annotations to enrich the data model with metadata [SEp]. In the remainder of this section, we introduce the CDS DDL, particularly CDS entities and CDS view entities.

One can use the syntax presented in Listing 2.8 [SEm] to define a CDS view entity in CDS DDL. Lines 1 to 6 describe the possibility of specifying additional metadata using annotations. In line 7, the *DEFINE VIEW ENTITY* keywords are used to specify a new CDS view entity, followed by the name of the view entity. The optional *ROOT* can be used in the ABAP RESTful Application Programming Model context to specify a root entity of a RAP business object [SEm]. The CDS view entity is eventually implemented by the *select_statement*, of which the syntax will be described afterward.

```
1 [@entity_annot1]
2 [@entity_annot2]
3 ...
4 [@view_entity_annot1]
5 [@view_entity_annot2]
6 ...
7 [DEFINE] [ROOT] VIEW ENTITY view_entity
8           AS select_statement [;]...
```

Listing 2.8: CDS view entity syntax.

As part of a CDS view entity, the select statement used to query a data source follows the syntax illustrated in Listing 2.9 [SEh]. The query begins with the *SELECT* keyword and is performed on the data sources defined in *data_source*, which can either be database tables, CDS view entities, CDS table functions, CDS hierarchies or the obsolete CDS DDIC-based views [SEh]. The *DISTINCT* keyword allows the removal of duplicate entries in the result set [SEh]. *Association1* and *association2* further define CDS associations for the current *SELECT* statement. In the *select_list*, the components of the view entity are listed. It is possible to access components of the *data_source* and components of the defined associations using path expressions [SEh]. Furthermore, the *clauses* in line 4 allows to define conditions, groupings, or set operators [SEh].

```
1 SELECT [DISTINCT] FROM data_source
2         [association1 association2 ...]
3             {select_list}
4         [clauses]
```

Listing 2.9: CDS select syntax.

For the *data_source* the syntax is presented in Listing 2.10 [SEk]. It is used to define the data source directly using the entity name or via a SQL path expression [SEk]. Furthermore, one can define an alias for the data source [SEk]. Also, it is possible to define joins (inner join, left outer join, right outer join), combining multiple data sources [SEk].

```
... entity | path_expr [AS alias] [join] ...
```

Listing 2.10: CDS data source syntax.

CDS associations are used to define relationships between CDS entities [SEi]. It allows to include fields of the association target in the current CDS view entity or expose these fields for reuse in other CDS entities [SEi]. In contrast to a classical join relationship, these associations are transformed internally to join expressions with the association source as the left-hand side and the association target as the right-hand side, but only on demand, [SEi]. In detail, the join is instantiated if, e.g., a field of the association target is used in the *element_list* of the CDS view entity [SEi]. Furthermore, it allows reusing these relationships [SEi]. A particular type of CDS association is the composition, which consists of a *to-child* association and a *to-parent* association [SEi]. The composition association can state an existential relationship between the composition child and the composition parent [SEi]. For associations, we distinguish three primary use cases [SEf]:

- **Exposing a CDS association**: An association is defined in the *element_list* of the CDS view entity. This exposes the association and allows the reuse in other CDS view entities without instantiating the join to the association target or selecting data from it. Therefore, it is also not part of the output.
- **Using a field from the association target**: All fields of the association target can be used either in the *FROM* clause, the *WHERE* clause, or the *element_list* of the current CDS view entity. The generated SQL statement thus includes the association target and the specified fields can be used accordingly.
- **Using an exposed association**: Similar to the previous use case, a CDS view entity can also use the fields of associations that are exposed by the *data_source* of the current CDS view entity.

The syntax of CDS associations is presented in Listing 2.11 [SEd, SEg, SEe]. The first two lines present the syntax of a simple association [SEd]. The statement begins with the *ASSOCIATION* keyword followed by the cardinality of the relationship, which can be expressed either in words (e.g., one to many) or using numbers in square brackets (e.g. [0..1]) while the latter is being used in the context of this thesis [SEd]. After the cardinality, the *TO* keyword is used to specify the association target [SEd]. It is also possible to introduce an alias using the *AS* keyword [SEd]. What follows are the conditions of the relationship, specified after the *ON* keyword [SEd]. Furthermore, associations allow extending conditions on a join using specific filter conditions on path expressions using the *WITH DEFAULT FILTER* keyword [SEd]. In line 4, the syntax for a composition *to-child* association is shown [SEg]. The statement starts with the *COMPOSITION* keyword followed by the cardinality and the target of the relationship [SEg]. It also allows introducing an alias for the relationship [SEg]. The *ON* condition can be automatically derived from the mandatory *to-parent* association of the composition child, and it is therefore not necessary to define it manually [SEg]. Finally, line 6 and line 7 present the syntax for an *to-parent* association [SEe]. One needs to define the association target of the relationship as well as the *ON* condition of the relationship, which requires the fields of the association source to be prefixed with *$projection* [SEe]. We can and will also use this *$projection* prefix in the simple association to define the *ON* clause. Similar to the other associations, the *to-parent* association also allows the introduction of an alias with the *AS* keyword [SEe]. It is important to note that the *to-parent* association needs to be defined first, and only afterward a *to-child* association can be introduced [SEe].

```
1 ... ASSOCIATION [cardinality] [TO] target [AS _assoc]
2     ON cds_cond [WITH DEFAULT FILTER cds_cond] ...
3
4 ... COMPOSITION [cardinality] [OF] target [AS _compos] ...
5
6 ... ASSOCIATION TO PARENT target [AS _assoc]
7     ON $projection.cds_cond ...
```

Listing 2.11: CDS association syntax.

The *select_list* is defined as a separated list of elements that can furthermore consist of an optional *KEY* keyword, defining a key element of the current CDS view entity, the field itself, and an optional alias, which can be defined with the *AS* keyword, followed by the alias [SEl]. Listing 2.12 shows the syntax of the *select_list* in line 1, followed by the syntax of a specific element in line 3 to line 22 [SEl].

```
 1 ... element1, element2, ...
 2
 3 ... { [@element_annot1]
 4       [@element_annot2]
 5       ...
 6         [KEY]   { field             [AS alias] }
 7               | { expose_assoc      [AS alias] }
 8               | { path_expr.element [AS alias] }
 9               | { literal            AS alias   }
10               | { parameter          AS alias   }
11               | { session_variable   AS alias   }
12               | { aggr_expr          AS alias   }
13               | { arith_expr         AS alias   }
14               | { builtin_func       AS alias   }
15               | { SQL-based scalar function
16                                      AS alias   }
17               | { reuse_exp          AS alias   }
18               | { case_expr          AS alias   }
19               | { cast_expr          AS alias   }
```

```
              | { EnumConstant       AS alias   }
                 ...
    }
```

Listing 2.12: CDS select list syntax.

The *clauses* of the CDS DDL closely follow the standard SQL DDL and are presented in Listing 2.13 [SEj]. We will, therefore, not introduce the usage of each further.

```
... [WHERE cds_cond]
    [GROUP BY field1, field2, ...]
    [HAVING cds_cond]
    [EXCEPT SELECT ...]
    [INTERSECT SELECT ...]
    [UNION [ALL] SELECT ...] ...
```

Listing 2.13: CDS clauses syntax.

Now, we will present some examples of CDS view entities to give further insight into the CDS DDL. In Listing 2.14, a CDS view entity with the name *DEMO_CDS_SCARR_SPFLI* is defined. The *data_source* is defined as an inner join between the tables *SPFLI* and *SCARR*. In lines 8 to 12, the *element_list* of the CDS view entity is defined, with *ID*, *CARRIER* and *FLIGHT* as the key of the resulting view.

```
 1 @AccessControl.authorizationCheck: #NOT_REQUIRED
 2 @EndUserText.label: 'CDS example 1'
 3 define view entity DEMO_CDS_SCARR_SPFLI
 4   as select from
 5     spfli
 6     inner join scarr on scarr.carrid = spfli.carrid
 7     {
 8       key spfli.carrid   as id,
 9       key scarr.carrname as carrier,
10       key spfli.connid   as flight,
11           spfli.cityfrom as departure,
12           spfli.cityto   as destination
13     }
```

Listing 2.14: CDS view entity example 1.

In Listing 2.15, a CDS view entity with the name *DEMO_SALES_CDS_SO_I_VE* is defined. The view defines *DEMO_SALES_SO_I* as the *data_source* and defines an association to the *DEMO_SALES_CDS_MATERIAL_VE* view to enrich a sales order item with material information. The field *material* from this association target is used in the *element_list* in line 18. This CDS view entity also defines a *to-parent* association to the *DEMO_SALES_CDS_SO_VE* in line 9 to line 11, which is then exposed in line

19. In other words, we define an existential relationship from the sales order item to a sales order. Also, a *to-child* association is defined in line 12 to line 13 to the view entity *DEMO_SALES_CDS_SO_I_SL_VE*, which is again exposed in the *element_list* of the current view in line 20. This can be interpreted as each sales order item having one or more schedule lines; the information about schedule lines is contained in the composition child.

```
 1  @AccessControl.authorizationCheck: #NOT_REQUIRED
 2  @EndUserText.label: ’CDS example 2’
 3  define view entity DEMO_SALES_CDS_SO_I_VE
 4    as select from
 5      demo_sales_so_i
 6      association [0..1] to DEMO_SALES_CDS_MATERIAL_VE
 7          as _Material
 8          on $projection.material = _Material.material
 9      association to parent DEMO_SALES_CDS_SO_VE
10          as _SalesOrder
11          on $projection.parent_key = _SalesOrder.so_key
12      composition [0..*] of DEMO_SALES_CDS_SO_I_SL_VE
13          as _ScheduleLine
14      {
15        key so_item_key,
16            parent_key,
17            posnr,
18            _Material.material as mat,
19            _SalesOrder,
20            _ScheduleLine
21      }
```

Listing 2.15: CDS view entity example 2.

We will omit further details of the CDS DDL as they will not be relevant to the scope of this thesis. Detailed documentation of the CDS DDL can be found in [SEb].

## 2.5 Entity Relationship

The entity-relationship (ER) model was first introduced in [Che76b] and is designed to represent crucial semantic information of the real world. Initially, it was designed to unify different data views, particularly the network, relational, and entity set model [Che76b]. The main concepts of the ER model are entities that [Che76b] describes as things that can be distinctly identified and contain attributes, as well as relationships, which the authors of [Che76b] describe as associations among entities. These ER models can visually be represented as ER diagrams, consisting of various shapes and labels. In the remainder of this section, we will introduce the ER models and their corresponding

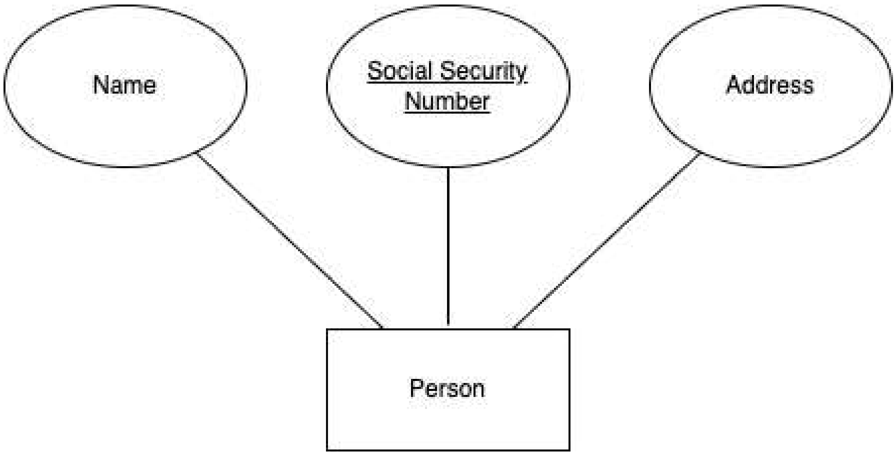


Figure 2.11: Entity with attributes and primary key.

representation in ER diagrams backed by [Che76b], [EN16] and [BE03] as it serves as the basis for the novel domain-specific language introduced by this thesis.

As already defined above, a *entity* is the representation of a real-world object that can be distinctly identified, while the object can be of physical (e.g., person, vehicle) or conceptual nature (e.g., company, job) [Che76b, EN16]. Furthermore, each entity has *attributes* which describe its properties [EN16]. For example, an entity PERSON may be described by the person's name, social security number, and address. A particular instance *p1* will have a value for all attributes defined by entity PERSON [EN16]. The minimal set of attributes that can be used to distinctly identify an entity is referred to as *primary key* [Che76b]. In case the *primary key* consists of multiple attributes, it is referred to as *composite key* [EN16]. Attributes can be further classified as presented in [EN16], but we will not introduce these concepts, as they will not be used in the scope of this thesis. In Fig. 2.11, the example mentioned above is modeled using the Chen-Notation, with the social security number as the primary key [Che76b].

*Relationships* define associations between entities referred to by a name and can contain descriptive attributes [Che76b]. Furthermore, a relationship defines a set of entities that are associated with this relationship [Che76b]. Formally, a relationship can be defined as the Cartesian product of $E_1, E_2, ..., E_n$ such that $R \subseteq E_1 \times E_2 \times ... \times E_n$, with $n$ denoted as the degree of the relationship [EN16]. In Fig. 2.12 a *binary relationship* (i.e. n = 2) and in Fig. 2.14 a *ternary relationship* (i.e. n = 3) is presented. Furthermore, a *recursive relationship* is presented in Fig. 2.13, as a binary relationship with $E_1 = E_2$ and additional *role names* to indicate the role within a relationship [BE03, EN16].

In addition to the degree of the relationship, one can define constraints on such, limiting the possible combinations of entities [EN16]. For these constraints, we distinguish between the *cardinality* ratio and the *participation* constraint [EN16].

The cardinality ratio defines the **maximum** a specific entity can participate in a relation-

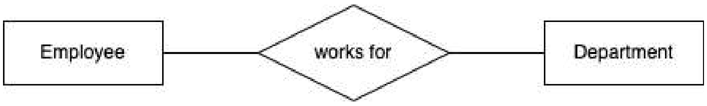


Figure 2.12: Binary relationship.

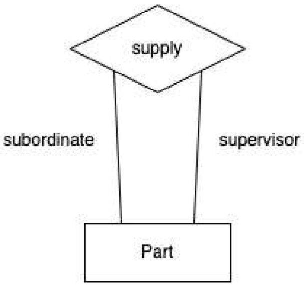


Figure 2.13: Recursive relationship with roles.

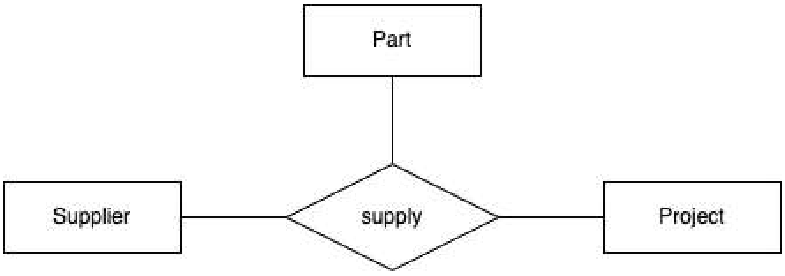


Figure 2.14: Ternary relationship.

ship [EN16]. Possible cardinality ratios are defined as *1:1*, *1:N* and *M:N*, with *N* and *M* as a unlimited maximum [EN16]. Revisiting the example of Fig. 2.12, one can constrain the relationship as 1:N, meaning an employee works for at most one department (1). In contrast, a department can be related to any number of employees (N). If we further constrain the relationship as 1:1, this would indicate that an employee works for at most one department (1) and a department has at most one employee assigned (1). In the case of an M:N relationship, the meaning of the relationship would change to an employee working for zero or more departments (M), and a department is related to zero or more employees (N).

On the other hand, the participation constraint defines the **minimum** number of relationship instances an entity can participate [EN16]. A *total participation* denotes that each entity has to participate in at least one relationship instance. In contrast, a *partial participation* states that some entities participate in relationship instances but not necessarily all [EN16]. For the example in Fig. 2.12, we can use total participation to define that each employee has to work for a department. We could further ease the relationship constraints by using partial participation, meaning that employees do not necessarily have to work for a department, but some of the employees do.

*Multiplicity* defines the combination of cardinality ratio and participation constraint [EN16]. We can use this notation to define the cardinality constraints on relationships. In Fig. 2.15, an example of a relationship with multiplicity is given. The model describes a relationship of zero or more employees (0..N) that work for exactly one department

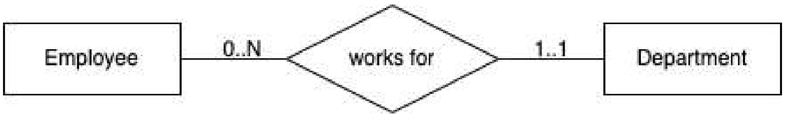


Figure 2.15: Relationship defined with multiplicity.

(1..1).

We will emit further ER details since they are irrelevant to this work's scope. It is important to note that the representation of ER greatly varies in different literature and notations. The remainder of this work will use a graphical syntax similar to the Chen-Notation. However, this thesis will use ER as a basis for the novel modeling language and will not cohere fully with the standard ER modeling language.

CHAPTER 3

# Related Work

This chapter introduces related work. First, existing web-based modeling tools will be presented. In more detail, BIGER, a modeling tool for entity relationship diagrams, the Graphical Language Server Protocol, and BIGUML, an example of a tool using the Graphical Language Server Protocol, are analyzed. The following section focuses on related work in Core Data Services. The current development tool for Core Data Services, Eclipse ADT, and existing Visual Studio Code extensions will be presented.

## 3.1 Web-based modeling tools

In the following section, two web-based modeling tools will be presented. BIGER was first introduced in [GB21] and is available as a VS Code extension for modeling entity relationship diagrams. BIGUML was presented in [MB23]. Again, this tool is available as a VS Code extension and allows the modeling of UML. BIGUML was developed using the Graphical Language Server Protocol, a framework following the pattern of LSP, with support for graphical modeling [Foua].

### 3.1.1 bigER

The BIGER tool was introduced in 2021 by [GB21] and is the first of its kind [GHHB22]. The tool allows hybrid modeling of ER in a textual or graphical manner within VS Code[1]. The implementation is based on LSP with a client-server architecture. For the client side, Sprotty[2] is used for the graphical interface, while VS Code provides the textual interface. BIGER introduced a domain-specific language implemented in Xtext[3] that also

[1] `https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.erdiagram`, last accessed: 30.07.2024

[2] `https://sprotty.org/`, last accessed: 30.07.2024

[3] `https://eclipse.dev/Xtext/`, last accessed: 30.07.2024

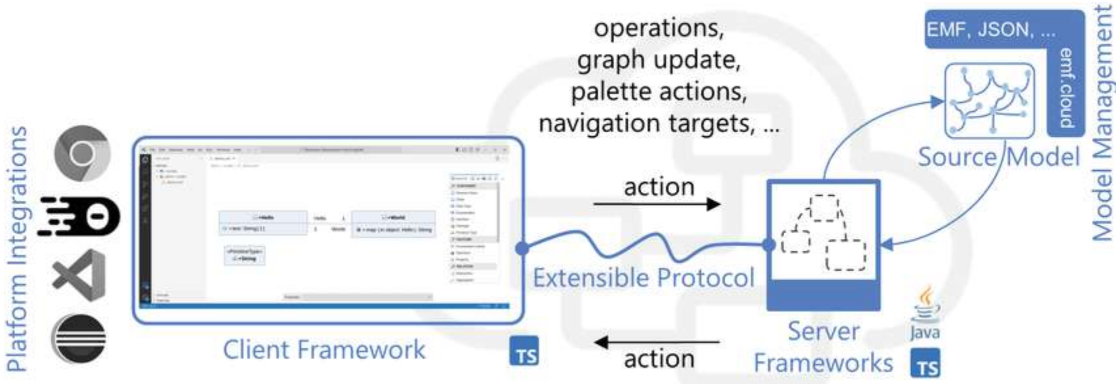


Figure 3.1: Overview of the GLSP architecture [BLO23].

serves as the basis for the Java-based language server. Another feature of bigER is the generation of SQL statements. In particular, it is possible to generate SQL statements to create the database schema based on the model artifact. One issue with the bigER tool is the need for a Java runtime due to the language server being implemented in Java. It is, therefore, not possible to run bigER in a browser-only scenario. The tool developed in this thesis will use a similar architecture as the bigER tool, with the difference of a TypeScript-based language server, to support such browser-only scenarios.

### 3.1.2 Graphical Language Server Protocol

The Graphical Language Server Protocol (GLSP)[4] is a framework for creating diagram editors based on web technologies, developed by EclipseSource [Foub]. GLSP is built using a client-server architecture with the client being responsible for rendering diagrams and handling user interactions and the server being responsible for managing the underlying source model as well as additional business logic [Lan]. In Fig. 3.1, the architecture of GLSP is illustrated.

The *server-side* is responsible for managing the source model and handling client requests or diagram states. Furthermore, the server exposes services to read or maintain the source model. The *source model* represents the diagram's underlying data structure. The client and server communicate using the *GLSP protocol*, similar to LSP, a set of messages that can be used to exchange information between the two components. For the implementation of the server-side GLSP supports TypeScript and Java. Additionally, multiple technologies are supported for the model source, e.g., EMF[5] and JSON[6].

The primary responsibility of the *client-side* is to present the diagram elements received by the language server visually. The rendering is based on web technologies, including HTML, CSS, TypeScript, and Scalable Vector Graphics (SVG). Additionally, the client

[4] https://eclipse.dev/glsp/, last accessed: 04.08.2024
[5] https://eclipse.dev/modeling/emf/, last accessed: 04.08.2024
[6] https://www.json.org/, last accessed: 04.08.2024

side provides a user interface to modify and interact with the diagram. GLSP supports different browser-like platforms. The framework provides integrations for Eclipse IDE[7], Eclipse Theia[8], VS Code[9], or browsers in general.

GLSP is built for graphical modeling only and is, therefore, not suitable for hybrid modeling. However, the advanced user interfaces and experience gained with graphical modeling will be used to create the novel modeling tool ER2CDS. Also, the GLSP protocol, as an example of an LSP-based protocol that supports graphical modeling, will significantly influence the extension of LSP used by ER2CDS.

### 3.1.3 bigUML

bigUML[10] is a modeling tool for different Unified Modeling Language (UML) diagram types, which was first reported on in [MB23]. bigUML is based on GLSP and supports VS Code and Eclipse Theia integration. The architecture of the bigUML tool is separated into a core module, diagram features, and tool features. It uses a Java-based implementation for the GLSP server; therefore, it is impossible to employ a browser-only environment. Furthermore, due to the restriction of GLSP, bigUML does not support hybrid modeling; it only supports graphical modeling. However, we will use the insights gained from this reference implementation using GLSP, as reported in [MB23], to implement ER2CDS.

## 3.2 Core Data Services

Core Data Services are the key technology of the VDM in the SAP environment, as introduced in Section 2.4. In this section, we will introduce existing tools for CDS development.

### 3.2.1 Eclipse ADT

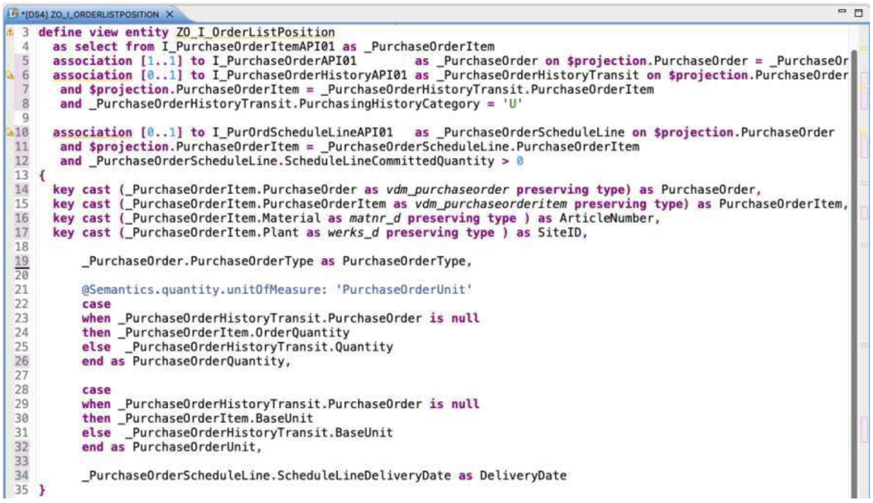
ABAP Development Tools (ADT) for Eclipse is the recommended development tool for CDS [SEb]. ADT offers an editor for the DDL, SDL, and DCL. The DDL editor supports the CDS DDL syntax and offers code completion, validation, and other features that are tightly integrated with the SAP environment. Furthermore, in a previous version of ADT, it was possible to display existing CDS entities graphically. However, this feature was discontinued, and ADT only offers a textual editor to create or modify CDS entities. The textual editor does not support a model-driven engineering process, and users are forced to implement CDS entities on the textual level. This establishes a high entry barrier for domain experts, rendering it unfeasible for them to understand and maintain

[7] `https://eclipseide.org/`, last accessed: 04.08.2024

[8] `https://theia-ide.org/`, last accessed: 04.08.2024

[9] `https://code.visualstudio.com/`, last accessed: 04.08.2024

[10] `https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.umldiagram`, last accessed: 04.08.2024



```
define view entity ZO_I_OrderListPosition
  as select from I_PurchaseOrderItemAPI01 as _PurchaseOrderItem
  association [1..1] to I_PurchaseOrderAPI01        as _PurchaseOrder on $projection.PurchaseOrder = _PurchaseOr
  association [0..1] to I_PurchaseOrderHistoryAPI01 as _PurchaseOrderHistoryTransit on $projection.PurchaseOrder
   and $projection.PurchaseOrderItem = _PurchaseOrderHistoryTransit.PurchaseOrderItem
   and _PurchaseOrderHistoryTransit.PurchasingHistoryCategory = 'U'

  association [0..1] to I_PurOrdScheduleLineAPI01   as _PurchaseOrderScheduleLine on $projection.PurchaseOrder
   and $projection.PurchaseOrderItem = _PurchaseOrderScheduleLine.PurchaseOrderItem
   and _PurchaseOrderScheduleLine.ScheduleLineCommittedQuantity > 0
{
  key cast (_PurchaseOrderItem.PurchaseOrder as vdm_purchaseorder preserving type) as PurchaseOrder,
  key cast (_PurchaseOrderItem.PurchaseOrderItem as vdm_purchaseorderitem preserving type) as PurchaseOrderItem,
  key cast (_PurchaseOrderItem.Material as matnr_d preserving type ) as ArticleNumber,
  key cast (_PurchaseOrderItem.Plant as werks_d preserving type ) as SiteID,

      _PurchaseOrder.PurchaseOrderType as PurchaseOrderType,

      @Semantics.quantity.unitOfMeasure: 'PurchaseOrderUnit'
      case
      when _PurchaseOrderHistoryTransit.PurchaseOrder is null
      then _PurchaseOrderItem.OrderQuantity
      else  _PurchaseOrderHistoryTransit.Quantity
      end as PurchaseOrderQuantity,

      case
      when _PurchaseOrderHistoryTransit.PurchaseOrder is null
      then _PurchaseOrderItem.BaseUnit
      else  _PurchaseOrderHistoryTransit.BaseUnit
      end as PurchaseOrderUnit,

      _PurchaseOrderScheduleLine.ScheduleLineDeliveryDate as DeliveryDate
}
```

Figure 3.2: Illustrative example for the usage of ADT within the Eclipse IDE.

the CDS directly and without the support of developers. Fig. 3.2 illustrates the usage of ADT within the Eclipse IDE. It becomes clear that domain experts are not ready to use such a notation.

### 3.2.2 Existing VS Code extensions

Several VS Code extensions for CDS development exist. On the one hand, the ABAP CDS Language Support[11] provides syntax support for CDS in VS Code. However, it does not offer advanced functionality, e.g., validation and auto-completion. Furthermore, the extension is not maintained anymore, and therefore, it lacks the latest CDS updates. On the other hand, SAP CDS Language Support[12] and core data services graphical modeler for VS Code[13] provide full language support, as well as graphical modeling support, but for SAP Cloud Application Programming (CAP), therefore not supporting SAP S/4HANA CDS [SEn, fS].

[11] https://marketplace.visualstudio.com/items?itemName=hudakf.cds

[12] https://marketplace.visualstudio.com/items?itemName=SAPSE.vscode-cds

[13] https://marketplace.visualstudio.com/items?itemName=SAPSE.vscode-wing-cds-editor-vsc

CHAPTER 4

# Requirements Analysis

This chapter describes the main idea and the requirements for ER2CDS. These are derived directly from industry, particularly the current development cycle for CDS view entities, as well as the learnings of the presented work in Chapter 3. First, the main idea and general approach of ER2CDS are presented. Then, we will analyze the requirements for the modeling part. Afterwards, the general requirements for the tool are described. The resulting requirements will then serve as a basis for the implementation of ER2CDS and will be further presented in Chapter 5 and Chapter 6.

## 4.1 Main idea and general approach

ER2CDS is designed to serve as the primary tool for a model-driven engineering process in CDS development. The tool shall support graphical and textual modeling of a domain-specific modeling language. The resulting model can then be transformed into a CDS view entity using a model-to-text transformation. To further ensure high usability, ER2CDS should allow users to connect SAP S/4HANA systems to the tool. However, ER2CDS should be usable with or without a connection to a SAP S/4HANA system. This will enable additional validation and modeling support for the user. Furthermore, existing CDS view entities can be imported into ER2CDS for analysis or to extend existing models further.

ER2CDS is intended to be used within VS Code and Business Application Studio (BAS), a browser-based development environment in the SAP environment. Both have access to the Visual Studio Marketplace and support the installation of extensions. Therefore, ER2CDS should be distributed as an extension in the Visual Studio Marketplace. No further dependencies should be required to run the extension for ease of use. In general, ER2CDS will be developed using web technologies only.

The following two sections describe a non-exhaustive list of the main requirements that ER2CDS should meet.

## 4.2 Modeling Requirements

We will now present the modeling requirements for the artifact of this thesis. First, we require a domain-specific language that enables graphical and textual modeling. The grammar for the modeling language has to support these concepts:

- **Root element definition**: The grammar needs to define a notation that indicates the root element of the model, which furthermore allows naming the model.
- **Entity definition**: The grammar needs to support the concept of entities. This includes the name of an entity, as well as an alias. Additionally, it should be possible to assign attributes to entities.
- **Attribute definition**: The grammar must support attributes defined with a name, a data type, and an optional alias. Furthermore, it should provide a notion of defining key attributes and attributes that should not appear in the output of the generated CDS view entity.
- **Relationship definition**: The grammar needs to support relationships between entities. They shall be defined with a source entity, a target entity, and conditions of the relationship, so-called join clauses. Furthermore, it must support a distinction between join-, association-, association-to-parent-, and composition-relationships, which is directly derived from the CDS DDL. Finally, the grammar should support different cardinalities in relationships.
- **Join clause definition**: The grammar must support the definition of join clauses for relationships. These are necessary to define conditions in relationships.

ER2CDS will support a textual and graphical user interface for modeling. Both interfaces should support all functionality for creating and modifying ER2CDS models. Additionally, ER2CDS should enable hybrid modeling. Therefore, the graphical and textual representation should be synchronized accordingly.

In order to ensure the quality of the models, an extended validation shall be implemented. The validation has to ensure the correctness of the model according to the defined grammar and, if connected to an SAP S/4HANA system, the correctness of the used entities and attributes.

In order to create a CDS view entity for an ER2CDS model, a model-to-text transformation must be implemented. For this transformation, the following requirements shall be met:

- **Validity**: The generated CDS view entity should adhere to the CDS DDL and, therefore, not contain any syntactical errors.
- **Correctness**: The output should represent the intentions of the ER2CDS model. In more detail, the entities and relationships should be correctly mapped to the CDS syntax, creating a CDS view entity that corresponds to the model.
- **Readability**: The generated CDS view entity should contain readable code that follows best practices used in the industry.

To import existing CDS view entities, a parser that extracts the entity's information from a serialized schema stored within a SAP S/4HANA system's database must be implemented. We expect the following requirements to hold:

- **Validity**: The parser should produce a valid ER2CDS model from the input data with no syntactical errors.
- **Correctness**: The resulting ER2CDS model should represent the entities and relationships of the input CDS view entity. It should not represent additional, automatically generated entities but only the ones present in the source code of the input CDS view entity.

To ensure usability, ER2CDS should also implement features widely common in textual and graphical modeling. These features include:

- **Syntax support**: The textual editor should provide completions and keyword support. Additional keywords should be visually highlighted in the textual editor.
- **Rename provider**: The rename feature should be implemented on a model level rather than on the textual level. This ensures correctness while renaming.
- **Layout support**: The graphical editor should support automatic layout and helper lines as alignment help.
- **Auto-completion/Value Help**: If an SAP S/4HANA system is connected to the tool, a value help should be presented for entities and attributes based on the system-specific data model. In addition, the tool should support the import of all attributes of an entity when connected to an SAP S/4HANA system.
- **Tool integration**: User interactions beyond modeling should be implemented within the native features of VS Code and BAS, namely the command palette.

## 4.3 Tool Requirements

We will now discuss further requirements for ER2CDS. These will primarily be technical considerations and requirements necessary to support optimal integration into the SAP environment.

Firstly, as already mentioned, ER2CDS should be distributed using the Visual Studio Marketplace. Therefore, the tool has to be implemented as a VS Code extension. This implicitly allows the integration in BAS as well. Furthermore, no additional dependencies should be needed. Only then can the tool be used within BAS, which runs in the browser. Compared to bigER and bigUML, both presented in Chapter 3, which use the Java-based EMF technology and a language server implemented in Java, ER2CDS should only use web technologies. For this, the language server has to be implemented in JavaScript (or, in this case, TypeScript). We will use the Langium framework, which is also implemented in TypeScript and supports browser-only deployment. Sprotty, again a framework based on web technologies, will be used for the diagramming part, so no further dependencies are introduced.

As mentioned in the previous section, the tool should support connecting to an SAP S/4HANA system. Due to the restrictions of a browser-only deployment, the communication shall be implemented using the Hypertext Transfer Protocol (HTTP). The SAP S/4HANA system exposes a web service that can be used by the tool to retrieve information about the data model. In order to define a system connection, ER2CDS needs to provide the possibility to maintain and store system information. Furthermore, secrets used for authentication in the web service should be managed securely.

Finally, the tool should be usable with or without an SAP S/4HANA system. All features that require an active system connection, e.g., import, validation, and value help, have to be implemented so that the tool still fully works in case no system connection is maintained or the system is not reachable.

CHAPTER 5

# Conceptualization

In this chapter, we will define the concepts of the novel modeling tool ER2CDS. First, the textual language specification will be presented. In particular, the grammar for the domain-specific language is defined. Then, a concrete graphical syntax for the DSL is introduced. First, we will define the graphical elements of the DSL. Then, the modeling interactions for ER2CDS will be described. The following section will present an extension for LSP, including additional actions. Afterward, the hybrid modeling approach and the synchronization between textual and graphical models will be explained. Then, the integration of ER2CDS in the SAP ecosystem will be analyzed. Finally, the model-to-text transformation and the concept behind the import of existing CDS view entities will be presented.

## 5.1 Textual Language Specification

This section will focus on the definition of the domain-specific syntax and the textual concrete syntax, which can be directly derived from the grammar definition of the DSL. The grammar will be defined using the Langium grammar, as presented in Section 2.2.2.

In Listing 5.1, the final grammar definition of the ER2CDS DSL is presented.

The grammar's root element is described in Lines 3 to 5. It starts with the keyword *er2cds* followed by the model's name. Afterward, multiple entities and relationships can be defined (line 5).

The definition of an entity starts with the keyword *entity* followed by the entity's name (line 7). Furthermore, an alias, using the *alias* keyword followed by the alias value (line 9), as well as multiple attributes (line 10), can be defined within the curly braces to complete the entity definition.

Attributes consist of an attribute type, either *key* for key attributes of the resulting CDS view entity or *no-out*, for attributes used while modeling, that should not appear in the output of the CDS view entity, the attribute name, a data type, and an optional alias, again using the *alias* keyword (line 14).

Lines 16 to 17 describe the definition of a data type. The grammar describes a data type matching the ID terminal rule defined in line 67. We will further restrict data types on the SAP predefined data types during the implementation phase.

In lines 19 to 24, the notion of a relationship is defined. It starts with an optional relationship type, *association*, *association-to-parent*, or *composition*, defining the relationship kind (line 20). If not defined, the relationship is interpreted as a join. Then, the keyword *relationship* followed by the name has to be stated (line 20). In between curly braces follow the source entity and the target entity of the relationship, separated by the -> keyword (line 21). A relationship entity is defined in lines 26 to 29. It consists of the entity (line 27), in particular, the name of the entity (cross-referenced by the ID), and the cardinality type, *1 or 0..N* (line 54), between brackets. Both source and target are defined as optional to support the incremental creation of the relationship, even though a relationship is only valid if the source entity and target entity are given. Furthermore, the keyword *join order* allows us to define the join order of the resulting CDS view entity (line 22). Finally, the join clauses are defined, consisting of an attribute of the source entity and an attribute of the target entity, separated by the = keyword (line 32). Both attributes are cross-references to an existing attribute identified by the ID (attribute name). However, additional scoping needs to be implemented to further restrict the attributes to attributes of the source entity or target entity. This will be presented in detail in Chapter 6.

Line 67 and line 68 define terminals. In line 67, the ID terminal is defined, which is used for the name and alias. It, therefore, defines valid names for entities, relationships, attributes, and the model itself. The terminal INT matches all whole numbers and defines the join order of the relationships.

The hidden terminal in line 70 is necessary since the Langium parser needs all characters of the input string to be matched, as described in Section 2.2.2. The ones in lines 71 to 72 define comments over multiple lines (line 71) or a single line (line 72).

```
grammar ER2CDS

entry ER2CDS:
        'er2cds' name=ID
        (entities+=Entity | relationships+=Relationship)*;

Entity:
        'entity' name=ID '{'
                ('alias' alias=ID)?
                (attributes+=Attribute)*
        '}';

Attribute:
        (type=AttributeType)? name=ID (':' datatype=DataType)?
           ('as' alias=ID)?;

DataType:
        type=ID;

Relationship:
        (type=RelationshipType)? 'relationship' name=ID '{'
                ((source=RelationshipEntity)? (('->' target=
                   RelationshipEntity))?)?)?
                ('join' 'order' joinOrder=JoinOrderType)?
                (joinClauses+=RelationshipJoinClause)*
        '}';

RelationshipEntity:
        target=[Entity] ('['
                cardinality=CardinalityType
        ']')?;

RelationshipJoinClause:
        (firstAttribute=[Attribute:ID] '=' secondAttribute=[
           Attribute:ID]);

type AttributeType = 'key' | 'no-out';
AttributeType returns AttributeType:
        KEY | NO_OUT
;
KEY returns string:
        'key';
NO_OUT returns string:
        'no-out';

type RelationshipType = 'association' | 'association-to-parent' | 'composition';
RelationshipType returns RelationshipType:
        ASSOCIATION | ASSOCIATION_TO_PARENT | COMPOSITION
;
ASSOCIATION returns string:
        'association';
ASSOCIATION_TO_PARENT returns string:
        'association-to-parent';
COMPOSITION returns string:
        'composition';

type CardinalityType = '1' | '0..N';
CardinalityType returns CardinalityType:
        ONE | ZERO_MANY
;
ONE returns string:
        '1';
ZERO_MANY returns string:
        '0..N';

JoinOrderType returns number:
        INT | ONE
;

terminal ID: /[_/a-zA-Z][\w_/]*/;
terminal INT returns number: /[0-9]+/;

hidden terminal WS: /\s+/;
hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

Listing 5.1: ER2CDS DSL grammar implemented using the Langium grammar.

In Listing 5.2, we present an example containing an entity *Employee* and *Department*, which are related through the *manages* relationship. Furthermore, it illustrates the usage of the *key* and *no-out* keywords and the use of an alias and cardinalities. Additionally, the *manages* relationship is defined as the first relationship, using the *join order* keyword.

```
er2cds EmployeeManagesDepartment

entity Employee {
    key PERNR : NUMC
    FNAME : CHAR as FirstName
    LNAME : CHAR as LastName
    DEPARTMENT_ID : NUMC
}

entity Deparment {
    no-out DEPARTMENT_ID : NUMC
    NAME : CHAR as Name
}

relationship manages {
    Employee[1] -> Deparment[1]
    join order 1
    DEPARTMENT_ID = DEPARTMENT_ID
}
```

Listing 5.2: An example for the usage of the textual ER2CDS DSL

## 5.2 Graphical Language Specification

We will now focus on the graphical language specification. In the following sections, the elements of the graphical concrete syntax will be introduced. Furthermore, the required modeling interactions are defined.

### 5.2.1 Element Definition

The graphical concrete syntax of ER2CDS is based on the Chen-Notation of entity relationship diagrams. The main elements are entities, represented as rectangulars, and relationships, represented as diamonds.

Entities are represented by the entity's name in the component's header, followed by the attributes, separated by a line. Each attribute is displayed using its name, the datatype, and, if present, the alias. A colon separates the name from the datatype. Furthermore, key attributes are underlined, and no-out attributes are displayed and crossed out. The example presented in Fig. 5.1 describes an entity named Employee, with the field *PERNR* as key. The attributes *FNAME* and *LNAME* show the use of an alias. In contrast, the

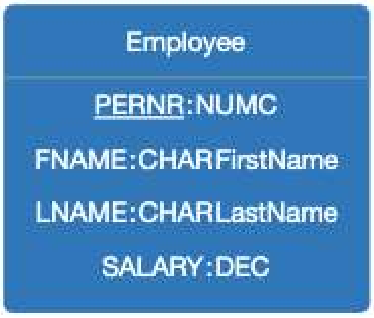


Figure 5.1: Entity in the ER2CDS graphical concrete syntax.

Figure 5.2: Relationship in the ER2CDS graphical concrete syntax.

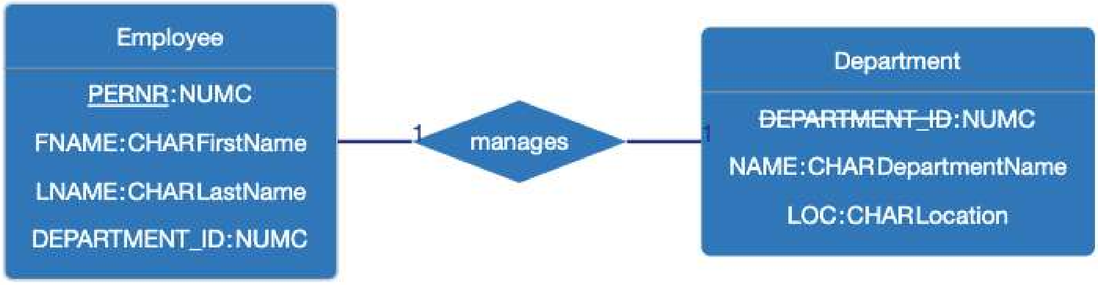


Figure 5.3: A relationship between two entities in the ER2CDS graphical concrete syntax.

*SALARY* attribute is defined as a no-out attribute and will therefore not be presented in the *element_list* of the generated CDS view entity. Relationships are represented by their names only. Fig. 5.2 illustrates an example of an relationship with the name *works_for*.

An undirected edge illustrates the associations between elements. Since we require a source and a target entity to form the only valid relationship, we further restrict relationships to binary relationships. A relationship can also define cardinalities for the source and the target entity. These are represented as a label above the corresponding edge. An example of a relationship is presented in Fig. 5.3. For this *Employee* acts as the source entity, which is related to *Department* via the *manages* relationship. In this case, the cardinalities are both one and can be interpreted as exactly one employee managing one department.

### 5.2.2 Modeling Interactions

We will focus on graphical modeling for the modeling interactions since the textual modeling uses the native functionality of VS Code. Specifically, we separate the requirements presented in Chapter 4 into four different kinds of interactions that need to be supported by ER2CDS.

- **Creation of elements**: The tool should support the creation of all model elements in specific entities, relationships, attributes, and join clauses. This includes the creation of the edges between entities and relationships.
- **Deletion of elements**: The tool should support deleting all model elements.
- **Maintaining attributes of elements**: The tool should support creating and updating attributes of different model elements.
- **Additional layout support**: The tool should support further assistance for creating models. This includes layout support, helper lines, and multi-selecting of elements.

In order to implement these requirements, the tool exposes two main interaction components, the *tool palette* and the *property palette*. The tool palette allows the user to select and use tools for general modeling, like *selection*, *deletion*, or *validation*, as well as tools used to create different elements of the model. The property palette is responsible for modifying the attributes of existing elements. The tool palette supports specific features that do not change during runtime. On the other hand, the property palette adapts to the selected element and allows for editing of the related attributes.

## 5.3 LSP Extension

LSP has to be extended to support the graphical and textual creation of ER2CDS. More detailed, additional actions are explicitly defined for graphical modeling. For the textual part, ER2CDS relies on the standard actions of LSP. Table 5.1 describes the custom actions defined by ER2CDS.

More details about implementing these extensions can be found in Chapter 6.

## 5.4 Hybrid Modeling

ER2CDS supports graphical as well as textual modeling. Therefore, it is necessary to synchronize these models accordingly to enable hybrid modeling. To overcome this, we must define a leading system that holds the only truth in case of discrepancy. As presented in Section 2.2.2, the Langium framework holds an internal model, which acts as the single source of truth. Therefore, all updates should be committed to this model.

We distinguish between two use cases: the update through the textual representation of the model and the one through the graphical modeler of ER2CDS. For the first use case, the framework addresses all main issues. Upon editing, the model is updated using standard LSP messages. Furthermore, the graphical model is re-rendered after the update is completed. The latter use case relies on the custom-defined LSP actions. The language server handles the create, update or delete actions by modifying the current Langium

| Name | Type | Parameter | Usage |
|---|---|---|---|
| CreateElementAction | Action | elementType: string | Used to create a specific element type. The *elementType* parameter contains the information on which kind of element should be created. |
| CreateEdgeAction | Action | sourceElementId: string<br>targetElementId: string | Used to create an edge from the source element, identified by *sourceElementId*, and the target element, identified by the *targetElementId*.<br>One of the elements has to be an entity, the other one a relationship. |
| CreateAttributeAction | Action | elementId: string | Creates an attribute for an entity, defined by the *elementId*. This action can only be successfully executed if the *elementId* corresponds to an entity. |
| CreateJoinClauseAction | Action | elementId: string | Creates a join clause on a relationship, which is identified by *elementId*. The action can be successfully executed if and only if *elementId* identifies a relationship. |
| UpdateElementPropertyAction | Action | sourceElementId: string<br>propertyId: string<br>value: string | Used to update a property of an element, corresponding to the *elementId*. The property is identified by the *propertyId*.<br>Furthermore, the *value* parameter defines the new value of the property. |
| DeleteElementAction | Action | elementIds: string[] | Deletes all elements identified by *elementIds*. It allows to delete multiple elements at the same time. |
| RequestAutoCompleteAction | RequestAction | elementId: string<br>type: string<br>search: string | Allows to retrieve value suggestions from the backend. The *elementId* identifies the selected element. The *type* defines the type of the autocomplete request.<br>In more detail, two types of auto-complete requests are defined. First, to retrieve suggestions for entity names, where the selection is restricted by the *search* value.<br>And secondly, to retrieve attribute suggestions for a specific entity. The request is therefore restricted to the attributes of the specific entity, as well as the *search* value.<br>The server responds with a SetAutoCompleteAction |
| SetAutoCompleteAction | ResponseAction | elementId: string<br>values: AutoCompleteValue[] | Sent in response to RequestAutoCompleteAction. The *elementId* identifies the element to which it corresponds.<br>The values is defined as an array of *AutoCompleteValue*, a type which contains a label. |
| RequestPopupConfirmModelAction | RequestAction | elementId: string<br>bounds: Bounds | Sent to request a popup from the server. Used to confirm the creation of an entity from the external data source, identified by *elementId*.<br>The bounds correspond to a default LSP type, defining the location of the popup.<br>The server responds with a SetPopupModelAction predefined in LSP. |
| CreateElementExternalAction | Action | elementId: string | Is sent after the manual popup confirmation, as a result of RequestPopupConfirmModelAction, to load an entity from the external data source. |
| RequestMarkersAction | RequestAction |  | Sent to validate the model. The server responds with a SetMarkersAction. |
| SetMarkersAction | ResponseAction | markers: Marker[] | Sent as a response to the RequestMarkersAction. It triggers a validation of the model and contains the validation result in the form of markers.<br>These are defined with a *elementId* to identify the element, *kind*, defining the severity (info, warning, error), and a *description*, a textual explanation. |

Table 5.1: Custom LSP actions defined by ER2CDS for graphical modeling.

model. In order to synchronize the textual representation, the model is serialized and written to the file system afterward. The subsequent save triggers the framework to start the same process as modifying the textual representation.

Using this process allows us to rely heavily on the framework. We only need to implement the serialization of the Langium model to a textual representation, which can be implemented straight-forward. In Chapter 6, the serialization will be presented in more detail.

## 5.5 SAP Integration

As presented in Chapter 4, ER2CDS should integrate into the SAP environment to further improve the modeling experience. We will now define the concrete integration possibilities. In particular, a SAP S/4HANA system can be connected to ER2CDS. For this, the system needs to expose a custom web service that can then be used to access the data model. This service is implemented as a CDS view entity, which is then exposed as a web service using the SAP S/4HANA infrastructure. A more detailed insight into the implementation can be found in Chapter 6. The following two sections describe the application areas for this access to the external data model within ER2CDS.

### 5.5.1 Value Helps

First, we can use the external data model to enable user value-helps. More specifically, the value-helps are presented for input fields, particularly the name of an entity or attributes. For example, to model a CDS view entity, all existing and valid data sources or associations are retrieved from the connected SAP S/4HANA system and used as possible entity names, as these are the only valid names in this context. This not only improves the usability of ER2CDS but also helps to ensure the quality of the models. As stated above, this will be used for the names and attributes of specific entities. Additional information, such as the datatype, can be retrieved from the system and used accordingly. It is important to note that we do not rely on the value-help for the join clauses, as they can be computed from cross-referencing.

### 5.5.2 Validation

Another use case for connecting to an SAP S/4HANA system is during the validation process. Access to an external data model allows the validation of the entities and attributes against it. The validations are closely related to the value help integration. Namely, all entities' names are checked for existence in the corresponding system, and all attributes are checked for existence within an entity. For example, if the entity corresponds to a database table, we check if a corresponding field exists on the database table for each attribute of the entity. This enables an advanced validation mechanism, which helps to ensure the validity of the final generated CDS view entity.

## 5.6 Model-to-Text Transformation

The model-to-text transformation is one of the central parts of ER2CDS. It allows the transformation of an ER2CDS model into a CDS view entity. We will use a template-based approach to generate the textual output. As already defined in Chapter 2, the template-based approach uses code snippets enriched with model information. The generation of a CDS view entity can be divided into five steps, which are directly derived from the syntax of CDS.

- **Generation of header annotations**: Each CDS view entity can define optional annotations applied to the entire view. This includes authorizations, metadata, and even end-user labels. For ER2CDS, we define the following default annotations. *@AccessControl.authorizationCheck: #CHECK* ensures that the result of the CDS view entity can only be accessed with suitable authorization. *@Metadata.ignorePropagatedAnnotations: true* is used that the derivation of metadata is performed on the current view, and *@EndUserText.label: 'Generated by ER2CDS'* adds a semantic label for the view.

- **Generation of header**: The header of the CDS view entity defines the name of the generated CDS view entity. The name results from the name of the ER2CDS model.

- **Generation of from clause**: In Chapter 2 the *data_source* is defined as an *entity* or *path_expression* that can consist of joins. For the transformation, we analyze the model for relationships with no defined association type. Since the join order is important, the relationships must be sorted according to the *join order* in the ER2CDS model. Furthermore, the join type is defined resulting from the specified cardinalities of the relationship, as presented in Table 5.2. The join clauses are again directly derived from the ER2CDS model.

- **Generation of associations**: The associations are generated similarly to the *from_clause*, with the difference that an association type has to be defined for the relationship. Again, the cardinalities of the relationship define the cardinality of the relationship. The concrete mapping is illustrated in Table 5.2. It is important to note that cardinalities are only applied for *association* and *composition*, but not for *association-to-parent* since the CDS syntax does not require them.

- **Generation of attributes**: Attributes are generated as defined by the entities of the ER2CDS model. First, the key attributes are generated with the appropriate *key* keyword of CDS. Then, the rest of the attributes follow, keeping in mind that *no-out* attributes of the ER2CDS are not added to the CDS view entity. Finally, the CDS view entity exposes the associations defined by the relationships.

| Source Cardinality | Target Cardinality | Join Type | Association Type |
|---|---|---|---|
| 1 | 1 | Inner Join | [1..1] |
| 1 | 0..N | Left Join | [1..*] |
| 0..N | 1 | Right Join | [0..1] |
| 0..N | 0..N | Left Join | [0..*] |
| n.d | n.d. | Inner Join | [0..1] |

Table 5.2: Mapping of cardinalities to the respective join/association type.

## 5.7 Import of Existing CDS View Entities

The import of existing CDS view entities to ER2CDS is implemented indirectly, using the serialized data of the view instead of the CDS view entity directly. SAP parses and stores all data related to a CDS view entity in the following tables: *DDCDS_FROMCLAUSE*, *DDCDS_SELECTLIST*, *DDCDS_ASSOC_DEF* and *DDCDS_CONDITION*. Like the value-help and validation, the web service defines and exposes a CDS view entity for each table. When importing an existing CDS view entity, the data of all entities is requested and used to create an ER2CDS model. After converting the SAP representation to an ER2CDS model, the resulting model is serialized and written in the file system. This also triggers the creation of a graphical model, as described in the update process.

*DDCDS_CONDITION* is the main table that stores the structure of the view entity. It is used to create the ER2CDS entities and the ER2CDS relationships. For all entries with *CONDITION_TYPE* 'FROM' or 'ASSOC_DEFINITION' and *EXPR_TYPE* 'TABLE_DATASOURCE' an entity is generated. The relationships are created using again *CONDITION_TYPE* 'FROM' for relationships defined using joins, and *CONDITION_TYPE* 'ASSOC_DEFINITION' for association relationships. The remaining properties are created by extracting information of entries with *EXPR_TYPE* 'ATOMIC', 'COMPARISON' or 'JOIN_DATASOURCE' for join relationships and *EXPR_TYPE* 'ASSOC_ELEMENT', 'COMPARISON', 'ASSOCIATION', 'TO_PARENT_ASSO' or 'COMPOSITION' for association relationships.

The data stored in *DDCDS_FROMCLAUSE* describe the *data_source* of the CDS view entity. It defines the join order of relationships in the ER2CDS model.

*DDCDS_SELECTLIST* holds information used to create entities' attributes. Furthermore, it generates the join clauses for relationships, respecting aliases.

The *DDCDS_ASSOC_DEF* database table stores data regarding associations of a CDS view entity. For the import, we can use it to extract a possible alias of the association and the cardinalities of the association relationship.

Finally, we can emit details about attributes handled internally by SAP S/4HANA. We will remove attributes that hold information about the SAP client, which separates the system into self-contained units. More implementation-specific details will be presented in the following chapter.

CHAPTER 6

# ER2CDS

This chapter will present the details of the implementation of ER2CDS. We will use the knowledge and the gathered requirements of the previous chapter to design and implement the artifact. First, an overview of the architecture is presented. A detailed analysis of the three main components follows the VS Code extension for the integration into VS Code, the language server as the backbone of the domain-specific language, and the webview for the diagramming.

## 6.1 Architecture

The architecture of ER2CDS is closely related to the standard LSP architecture. Fig. 6.1 illustrates a high-level view of the architecture, on the one hand, VS Code and the extension host, that includes the ER2CDS language client, as well as the webview used for diagramming and on the other hand, the ER2CDS language server. The SAP S/4HANA system is illustrated as an optional component. It is important to note that the language server runs in a different process, and the communication between the language client and the language server is based on LSP. In general, the language client orchestrates all communication to the language server. This includes the Sprotty integration, which sends and receives messages through the language client. The communication to the SAP S/4HANA system is based on HTTP.

The following sections will provide insights into the implementation of these three components.

## 6.2 Visual Studio Code Extension

First, we will present the configuration files needed for the VS Code extension. Since ER2CDS is built using Node.js, the most important configuration file is the *package.json*.

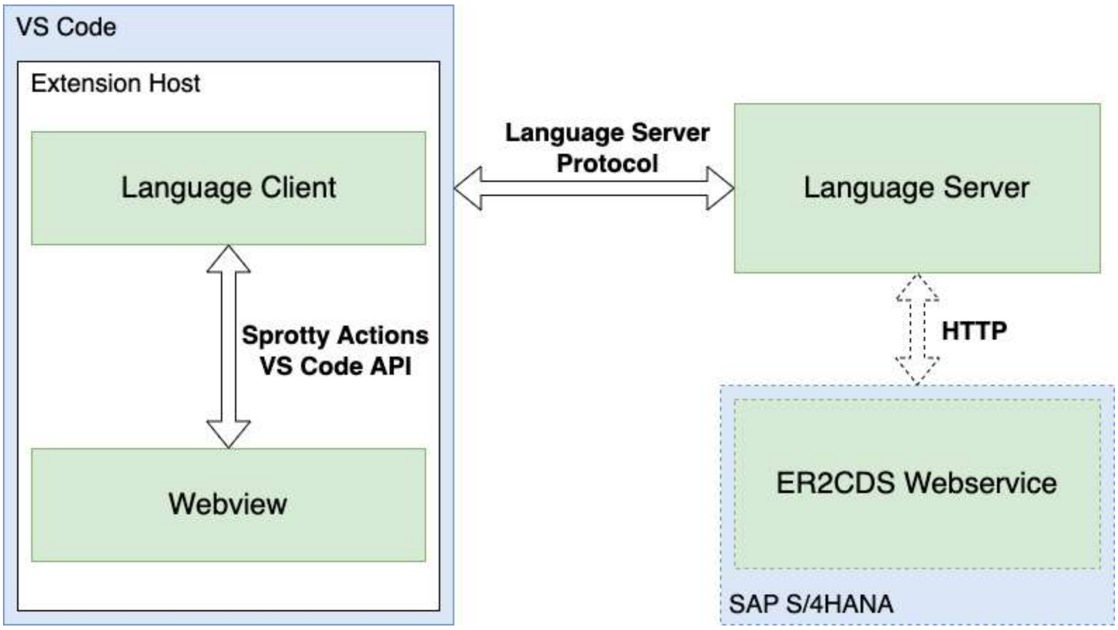


Figure 6.1: Architecture of ER2CDS with all three main components.

Moreover, for VS Code extensions, the *package.json* file also allows defining so-called contributions and other dependencies. It defines the language and grammar for the textual editor and the specific editors and views used by the extension. Additionally, commands that are then available in the command palette of VS Code can be defined. In Listing 6.1, the definition of four commands to generate a CDS view entity, import a CDS view entity, and add or remove an SAP system connection is presented (line 18 - line 40). These are the main actions supported by the ER2CDS. The start of the extension and creation of a model itself is done implicitly by creating a file with the *.er2cds* file extension. This behavior is defined in the languages section, as in lines 11 to 13.

```
1  {
2  ...
3    "contributes": {
4      "languages": [
5        {
6          "id": "er2cds",
7          "aliases": [
8            "er2cds",
9            "er2cds"
10         ],
11         "extensions": [
12           ".er2cds"
13         ],
14         "configuration": "./language-configuration.json"
```

```
      }
    ],
    ...
    "commands": [
      ...,
      {
        "command": "er2cds.generate.cds.proxy",
        "title": "Generate␣CDS",
        "category": "ER2CDS␣Diagram"
      },
      {
        "command": "er2cds.import.cds.proxy",
        "title": "Import␣CDS",
        "category": "ER2CDS␣Diagram"
      },
      {
        "command": "er2cds.add.system.proxy",
        "title": "Add␣SAP␣System",
        "category": "ER2CDS␣Diagram"
      },
      {
        "command": "er2cds.remove.system.proxy",
        "title": "Remove␣SAP␣System",
        "category": "ER2CDS␣Diagram"
      }
    ],
    ...
  },
  ...
}
```

Listing 6.1: Contributions in the package.json

In addition to the contributions presented above, the *package.json* also references other configuration files that can be used for customizing the textual representation of the language. The *language-configuration.json* is used for comments and bracket behavior. It defines symbols for comments, supported brackets, auto-closing, and surrounding pairs. For ER2CDS, we define // as line comments and /* respectively */ as block comments. Furthermore, { and } act as brackets that can be used to surround a selection and shall be auto-closed by the editor.

Furthermore, the *package.json* references *er2cds.tmLanguage.json* for a grammar definition. The Langium grammar generates this file and defines keywords for the ER2CDS language. The text editor of VS Code then highlights these accordingly. Similar to the definition in *language-configuration.json*, this file also includes a definition for comments. In contrast, only the symbols can be defined. Therefore, ER2CDS uses both but with identical symbols for obvious reasons.

We will now focus on the implementation of the VS Code extension, starting with the entry point. Each VS Code extension is triggered using an *activate* function, presented in Listing 6.2. On activation, the language client is created in line 3. Then, the webview is created using the context and language client. Additionally, for the webview, we register default commands for opening a diagram, fitting it to the screen, centering and exporting the diagram in line 16. Furthermore, in line 17, a text editor synchronization is registered. This indicates that the tool automatically opens the diagram each time the text editor's document is switched. In lines 19 - 20, handlers for the commands defined in the *package.json* are registered. Finally, in lines 22 - 29, we check if an SAP S/4HANA connection is maintained, and if so, send the credentials to the language server. This is crucial for the features requiring the external system, e.g., validation.

```
1  export async function activate(context: vscode.ExtensionContext
      ) {
2      ...
3      languageClient = await createLanguageClient(context);
4
5      const webviewPanelManager = new ER2CDSWebViewPanelManager(
6          context,
7          {
8              extensionUri: context.extensionUri,
9              languageClient,
10             supportedFileExtensions: ['.er2cds'],
11             singleton: true,
12             messenger: new Messenger({ ignoreHiddenViews: false
                   }),
13         }
14     );
15
16     registerDefaultCommands(webviewPanelManager, context, {
          extensionPrefix: 'er2cds' });
17     registerTextEditorSync(webviewPanelManager, context);
18
19     context.subscriptions.push(vscode.commands.registerCommand(
          'er2cds.generate.cds.proxy', generateCDSHandler));
20     ...
21
22     const sapUrl = await context.secrets.get('sapUrl');
23     const sapClient = await context.secrets.get('sapClient');
24     const sapUsername = await context.secrets.get('sapUsername'
          );
```

```
    const sapPassword = await context.secrets.get(’sapPassword’);
    if (sapUrl && sapClient && sapUsername && sapPassword) {
        await sendToServer(addSystemCommand, [sapUrl, sapClient, sapUsername, sapPassword]);
        ...
    }
}
```

Listing 6.2: Contributions in the package.json

The creation of the language client is implemented straightforwardly using the *vscode-languageclient* library by first defining server options, containing the location of the language server and client options, containing information regarding the VS Code integration. Then, the language client is instantiated and started.

To handle webviews, Sprotty defines two classes, the *WebviewPanelManager* and the *WebviewEndpoint*. The *WebviewPanelManager* handles all opened webviews and keeps track of the currently active webview. Furthermore, it is responsible for creating new webviews. The *WebviewEndpoint* is a wrapper class around a webview, which takes care of the communication between the webview and the extension host. Specifically for the language server protocol, Sprotty defines th *LspWebviewPanelManager* and the *LspWebviewEndpoint*. We further extend these two classes to tailor them to our needs. In more detail, the *ER2CDSWebViewPanelManager* overrides the *createWebview* method to define the location of the webview implementation, as presented in Listing 6.3.

```
protected override createWebview(identifier: SprottyDiagramIdentifier): vscode.WebviewPanel {
    const extensionPath = this.options.extensionUri.fsPath;
    return this.createWebviewPanel(identifier, {
        localResourceRoots: [createFileUri(extensionPath, ’.’)],
        scriptUri: createFileUri(extensionPath, ’.’, ’out’, ’webview.js’)
    });
}
```

Listing 6.3: Implementation of the createWebview method within the ER2CDSWebViewPanelManager

Also, we override the *createEndpoint* method to create a *ER2CDSWebViewPanelManager* instead of the default *LspWebviewEndpoint* and to intercept the workspace edit actions emitted from the diagram to save the documents. This is part of the synchronization between graphical and textual models as described in Chapter 5. In Listing 6.4, the implementation of the method is presented (lines 1 - 19) with the creation of the webview (lines 5 - 14) and the registration of the action handler (line 16), as well as the implementation of the handler for the workspace edit actions (line 21 - 30).

```
 1  protected override createEndpoint(identifier:
       SprottyDiagramIdentifier): LspWebviewEndpoint {
 2      const webviewContainer = this.createWebview(identifier);
 3      const participant = this.messenger.registerWebviewPanel(
           webviewContainer);
 4
 5      const webview = new ER2CDSWebviewEndpoint(
 6          this.context,
 7          {
 8              languageClient: this.languageClient,
 9              webviewContainer,
10              messenger: this.messenger,
11              messageParticipant: participant,
12              identifier
13          }
14      );
15
16      webview.addActionHandler(WorkspaceEditAction.KIND, this.
           handleWorkspaceEditAction);
17
18      return webview;
19  }
20
21  protected async handleWorkspaceEditAction(action:
       WorkspaceEditAction) {
22      await vscode.workspace.applyEdit(convertWorkspaceEdit(
           action.workspaceEdit));
23
24      const changes = action.workspaceEdit.changes;
25      if (changes) {
26          for (const uri in changes) {
27              await vscode.workspace.save(vscode.Uri.parse(uri));
28          }
29      }
30  }
```

Listing 6.4: Implementation of the createEndpoint method within the ER2CDSWebViewPanelManager

*ER2CDSWebviewEndpoint* acts as an additional wrapper class for the *LspWebviewEndpoint*. This allows for further interception of messages from the language server or sending messages to the language server. Currently, the implementation serves as an entry point for future implementations.

Finally, we will present how commands are handled within the extension. As presented above, the commands are defined in the *package.json* file and can be used with the VS Code command palette. All ER2CDS commands are defined with the postfix **.proxy**, which already gives a hint regarding the implementation. In the *activate* function presented in Listing 6.2, an example of the registration of a handler is shown. In more detail, the *generateCDSHandler* function is registered for the **er2cds.generate.cds.proxy**. This function subsequently executes the command **er2cds.generate.cds** with additional arguments, which is then handled by the language server. In Listing 6.5, the implementation of *generateCDSHandler* is presented, as well as the *sendToServer* function, which furthermore triggers the execution of the command with the parameters. The implementation of the other commands follows roughly the same pattern.

```
export const generateCDSHandler = async () => {
    const activeEditor = vscode.window.activeTextEditor;

    if (activeEditor?.document?.languageId === 'er2cds') {
        sendToServer(generateCdsCommand, [vscode.window.
            activeTextEditor?.document.uri.toString()]);
    } else {
        vscode.window.showErrorMessage('Error!␣Invalid␣file');
    }
};

export const sendToServer = async (command: string, args?: any)
    => {
    const response: string | undefined = await vscode.commands.
        executeCommand(command, ...args);
    if (response) {
        if (response.startsWith('Error')) {
            vscode.window.showErrorMessage(response);
        } else {
            vscode.window.showInformationMessage(response);
        }
    }
};
```

Listing 6.5: Generate CDS command implementation with the proxy pattern and additional arguments.

A special case is the **er2cds.add.system.proxy** command. In order to connect a SAP S/4HANA system to ER2CDS, the following information is necessary:

- **System Uniform Resource Locator (URL)**: The location of system gateway.
- **SAP Client**: The client for which the connection is created.
- **SAP Username**: A valid user within the specified client of the system.
- **SAP Password**: A valid password for the user.

In order to request this information, ER2CDS uses the *vscode.window.showInputBox* function, which prompts the user to an input field. After the input, the system URL is validated by sending a request to the specified URL. Also, the username and password are validated by sending a request directly to the ER2CDS service, which checks for the existence of the web service as well. To securely store this information, we use the *vscode.ExtensionContext.secrets.store*. Additionally, while running, the language server stores the information as well. Further details are described in the following sections.

## 6.3 SAP S/4HANA Web Service

The ER2CDS web service itself is implemented as a CDS view entity. It exposes five entity sets for the modeling and import function and is used by the language server to request additional information about the system-specific data model:

- **Entities**: Existing entities of the data model.
- **Attributes**: Attributes of entities in the data model.
- **ImportSelectList**: Exposes the *DDCDS_SELECTLIST* table enriched with the datatype of the fields.
- **ImportCondition**: Exposes the *DDCDS_CONDITION* table enriched with the datatype of the fields.
- **ImportAssocDef**: Exposes the *DDCDS_ASSOC_DEF* table.
- **ImportFromClause**: Exposes the *DDCDS_FROMCLAUSE* table.

Implementing the web service as a CDS view entity, functionality for filtering, searching, or pagination is handled automatically by the SAP S/4HANA system. The authentication is based on the standard user management of SAP. The username and password can be maintained within ER2CDS and are used to authenticate the client.

## 6.4 Language Server

The language server is implemented using the Langium framework and relies on dependency injection. This section will give an overview of the implementation.

The entry point is the *server.ts* file, where the connection to the client is built, the services and language-specific services are injected, and the language server is started accordingly. Furthermore, a diagram handler for *DiagramActionNotification* and *DiagramDidCloseNotification* messages is added to the language server. The implementation is presented in Listing 6.6.

```
1  // Create a connection to the client
2  export const connection = createConnection(ProposedFeatures.all
      );
3
4  // Inject the shared services and language-specific services
5  export const { shared } = createER2CDSServices({ connection,
      ...ER2CDSFileSystem });
6
7  // Start the language server with the shared services
8  startLanguageServer(shared);
9
10 // Add a diagram handler
11 addDiagramHandler(connection, shared);
```

Listing 6.6: Entry point of the ER2CDS language server.

In line 5 of Listing 6.6, the *createER2CDSServices* function is called, which is responsible for defining the services and capabilities of the language server, for this Langium separates between language independent, so-called shared services and language-specific services. The function is illustrated in Listing 6.7.

```
export function createER2CDSServices(context:
   DefaultSharedModuleContext): {
    shared: LangiumSprottySharedServices,
    ER2CDS: ER2CDSServices
} {
    const shared = inject(
        createDefaultSharedModule(context),
        ER2CDSGeneratedSharedModule,
        ER2CDSSprottySharedModule
    );
    shared.lsp.ExecuteCommandHandler = new ER2CDSCommandHandler
        ();

    const ER2CDS = inject(
        createDefaultModule({ shared }),
        ER2CDSGeneratedModule,
        ER2CDSModule
    );
    shared.ServiceRegistry.register(ER2CDS);

    registerValidationChecks(ER2CDS);

    return { shared, ER2CDS };
}
```

Listing 6.7: Dependency injection of the shared and language-specific services.

It first defines the shared services by injecting the default shared module, which is predefined by the framework, the *ER2CDSGeneratedSharedModule* and the *ER2CDSSprottySharedModule*. The *ER2CDSGeneratedSharedModule* is generated by the framework using the defined language. It includes an object which provides generic access to the structure of the AST. The *ER2CDSSprottySharedModule*, on the other hand, is implemented by ER2CDS and defines the diagram services. In Listing 6.8, the implementation of *ER2CDSSprottySharedModule*, as well as the implementation of the *ER2CDSDiagramServerFactory* is presented. In line 3, the *ER2CDSDiagramServerFactory* is used as the *diagramServerFactory*. Furthermore, the factory instantiates a custom diagram server *ER2CDSDiagramServer*, which will serve as the primary communication point between the diagramming module and the language server.

```
 1  const ER2CDSSprottySharedModule: Module<
       LangiumSprottySharedServices, SprottySharedServices> = {
 2      diagram: {
 3          diagramServerFactory: ER2CDSDiagramServerFactory,
 4          DiagramServerManager: (services) => new
               DefaultDiagramServerManager(services),
 5      }
 6  };
 7
 8  const ER2CDSDiagramServerFactory = (services:
       LangiumSprottySharedServices): ((clientId: string, options?:
        DiagramOptions) => ER2CDSDiagramServer) => {
 9      const connection = services.lsp.Connection;
10      const serviceRegistry = services.ServiceRegistry;
11
12      return (clientId, options) => {
13          const sourceUri = options?.sourceUri;
14
15          if (!sourceUri)
```

```
            throw new Error("Missing 'sourceUri' option in request.");

        const language = serviceRegistry.getServices(URI.parse(sourceUri as string)) as ER2CDSServices;

        if (!language.diagram)
            throw new Error(`The '${language.LanguageMetaData.languageId}' language does not support diagrams.`);

        return new ER2CDSDiagramServer(async action => { connection?.sendNotification(DiagramActionNotification.type, { clientId, action }); }, language);
    };
};
```

Listing 6.8: Injection of the custom diagram server of ER2CDS.

Furthermore, in line 10 of Listing 6.7, we define a custom command handler for the VS Code commands sent by the client. As shown in Listing 6.9, the command handler accepts commands defined in the extension implementation and executes functions accordingly. For the **er2cds.add.system** and **er2cds.remove.system** command, the implementation modifies attributes of a global namespace, allowing all language server modules to access the system information when needed.

```
export class ER2CDSCommandHandler extends AbstractExecuteCommandHandler {
    registerCommands(acceptor: ExecuteCommandAcceptor): void {
        acceptor('er2cds.generate.cds', args => {
            generateCDS(args[0]);
        });

        acceptor('er2cds.import.cds', args => {
            importCDS(args[0], args[1]);
        })

        acceptor('er2cds.add.system', args => {
            ER2CDSGlobal.sapUrl = args[0];
            ER2CDSGlobal.sapClient = args[1];
            ER2CDSGlobal.sapUsername = args[2];
            ER2CDSGlobal.sapPassword = args[3];
        });

        acceptor('er2cds.remove.system', args => {
            ER2CDSGlobal.sapUrl = undefined!;
            ER2CDSGlobal.sapClient = undefined!;
            ER2CDSGlobal.sapUsername = undefined!;
            ER2CDSGlobal.sapPassword = undefined!;
        });
    }
}
```

Listing 6.9: The command handler of ER2CDS language server, used to handle VS Code commands.

Finally, the *createER2CDSServices*, presented in Listing 6.7, defines the language-specific services. Again, the default module is defined by the framework, and the *ER2CDSGeneratedModule* is generated using the DSL. For the *ER2CDSModule*, the implementation is presented in Listing 6.10. It defines a custom module for validation, *ER2CDSValidator*, and a custom scope provider, *ER2CDSScopeProvider*. Furthermore, a diagram generator is injected for diagramming, *ER2CDSDiagramGenerator*, which is responsible for creating the diagram elements using the language server model. Also, we define a layout engine for the diagram. For this, we rely on the Eclipse Layout Kernel (ELK), using an external library (elkjs).

```
export const ER2CDSModule: Module<ER2CDSServices,
   PartialLangiumServices & SprottyDiagramServices &
   ER2CDSAddedServices> = {
    validation: {
        ER2CDSValidator: (services) => new ER2CDSValidator(
           services)
    },
    references: {
        ScopeProvider: (services) => new ER2CDSScopeProvider(
           services),
    },
    diagram: {
        DiagramGenerator: (services: any) => new
           ER2CDSDiagramGenerator(services),
        ModelLayoutEngine: (services: any) => new
           ElkLayoutEngine(services.layout.ElkFactory, services
           .layout.ElementFilter, services.layout.
           LayoutConfigurator) as any
    },
    layout: {
        ElkFactory: () => () => new ElkConstructor({ algorithms
           : ['layered'] }),
        ElementFilter: () => new DefaultElementFilter,
        LayoutConfigurator: () => new DefaultLayoutConfigurator
    }
};
```

Listing 6.10: Definition of the language-specific services.

After presenting the main modules, we will now introduce them in more detail. First, the diagram generator is responsible for creating the diagram elements for a given model. The entry point of the *ER2CDSDiagramGenerator* class is the *generateRoot* method, which is passed a *LangiumDocument* as an argument. The contained parsing result, an AST, is then used to create the diagram elements. In Listing 6.11, the *generateRoot* method is presented.

```
    protected generateRoot(args: GeneratorContext<ER2CDS>): ER2CDSRoot {
        const { document } = args;
        const sm = document.parseResult.value;

        const graph: ER2CDSRoot = {
            type: GRAPH,
            id: sm.name ? args.idCache.uniqueId(sm.name) : args.idCache.uniqueId('root'),
            name: sm.name,
            children: []
        };

        graph.children?.push(...sm.entities.map(e => this.generateEntity(e, args)));
        ...
        return graph;
    }
```

Listing 6.11: The entry point for the diagram generator, creating the diagram elements.

Furthermore, the *generateEntity* method is illustrated in Listing 6.12 as an example of creating such a diagram element. One can easily see that the information stored in the AST is used to create Sprotty elements. Also, it is important to note that each element is identified using a unique identification. The information defined in these elements can then be used in the webview to render the diagram visually. In this example, an entity of ER2CDS is used to create a *EntityNode*, which is a subclass of the Sprotty *SNode*. Also, a *SCompartment* with the entity's attributes is created and added as a child to the *EntityNode*.

```
    protected generateEntity(entity: Entity, { idCache }: GeneratorContext<ER2CDS>): EntityNode {
        const entityId = idCache.uniqueId(entity.name, entity);
        const node = <EntityNode>{
            type: NODE_ENTITY,
            id: entityId,
            layout: 'vbox',
            layoutOptions: {
                VGap: 10.0,
            },
            children: []
        };
        ...
        const attributesCompartment = <SCompartment>{
            type: COMP_ATTRIBUTES,
            id: idCache.uniqueId(entityId + '.attributes'),
            layout: 'vbox',
            layoutOptions: <LayoutOptions>{
                HAlign: 'left',
                VGap: '1.0'
            },
            children: entity.attributes.map(a => this.
               generateAttributeLabels(a, entityId, idCache))
        };
        node.children?.push(attributesCompartment);

        return node;
    }
```

Listing 6.12: An example for the creation of a diagram element.

We emit further details of the diagram generation since it follows the same principles presented above. However, it is again important to note that only the information defined in these elements will be available in the webview. Therefore, the generation has to consider all aspects of the ER2CDS language.

The diagram server is the communication gateway between the language server and the webview. The primary method implemented in the *ER2CDSDiagramServer* is the *handleAction* method. It takes an LSP action as an argument and handles the action accordingly. A handler is defined for each of the custom messages defined by ER2CDS. The action is propagated to the corresponding handler, which then implements the behavior of the action. The implementation of *handleAction* is shown in Listing 6.13. The *handleAction* method determines the type of the action before calling the correct handler accordingly.

```
export class ER2CDSDiagramServer extends DiagramServer {
    ...
    protected override handleAction(action: Action): Promise<void> {
        switch (action.kind) {
            case CreateElementAction.KIND:
                new CreateElementActionHandler().handle(action as CreateElementAction, this, this.services);
                break;

            ...
        }

        return super.handleAction(action);
    }
}
```

Listing 6.13: The handleAction method of the diagram server, responsible to call the correct handler depending on the action kind.

An example of a handler implementation, the *CreateElementActionHandler*, is presented in Listing 6.14. In the *handle* method of the *CreateElementActionHandler*, the current model is retrieved using the server state. Then, a new entity is created and added to the entities in the current model. In order to trigger the synchronization between the graphical and textual model, the resulting model is then serialized and written to the document.

```
export class CreateElementActionHandler {
    public handle(action: CreateElementAction, server: ER2CDSDiagramServer, services: ER2CDSServices): Promise<void> {
        const sourceUriString = server.state.options?.sourceUri?.toString();
        if (!sourceUriString)
            return Promise.resolve();

        const sourceUri = URI.parse(sourceUriString);
        if (!sourceUri)
            return Promise.resolve();

        const document = services.shared.workspace.LangiumDocuments.getOrCreateDocument(sourceUri);
        if (!document)
            return Promise.resolve();

        const model = document.parseResult.value as ER2CDS;

        switch (action.elementType) {
            case NODE_ENTITY:
                const newEntity: Entity = {
                    $type: ’Entity’,
                    $container: model,
                    name: this.getNewEntityName(’Entity’, model
                       .entities),
                    attributes: []
                }

                model.entities.push(newEntity);
            ...
        }

        return synchronizeModelToText(model, sourceUri, server,
            services);
    }
    ...
}
```

Listing 6.14: An example for the handler implementation of custom LSP actions sent by the graphical editor.

All handlers for actions modifying the model are implemented similarly. For the request actions, instead of modifying the model and serializing it to the document, the language server responds with an action using the *ER2CDSDiagramServer*'s *dispatch* method. The definition of all custom messages can be found in Chapter 5.

As part of synchronizing the graphical and textual representation, the model is serialized by the language server and written to the document. The *synchronizeModelToText* function implements this logic accordingly. As presented in Listing 6.15, first, the model is serialized. Then, the resulting source code is written to the document using a *WorkspaceEditAction*, which can be dispatched using the *DiagramServer*. As presented in Section 6.2, the *WorkspaceEditAction* is intercepted in the *ER2CDSWebViewPanelManager*, and the edits are applied and saved accordingly.

```
export function synchronizeModelToText(model: ER2CDS, sourceUri
   : URI, server: ER2CDSDiagramServer, services: ER2CDSServices
   ): Promise<void> {
    const source = serialize(model);
    ...
    const workspaceEdit = {
        changes: {
            [sourceUri.toString()]:
                [
                    {
                        range: Range.create(Position.create(0,
                           0), Position.create(document.
                           textDocument.lineCount + 1, 0)),
                        newText: source
                    }
                ]
        }
    }

    const workspaceEditAction: WorkspaceEditAction = {
        kind: WorkspaceEditAction.KIND,
        workspaceEdit: workspaceEdit
    }

    server.dispatch(workspaceEditAction);

    return Promise.resolve();
}
```

Listing 6.15: The serializeModelToText function as part of the synchronization strategy used by ER2CDS.

The serialization of the model follows a template-driven approach. To further illustrate this, Listing 6.16 presents the entry point, the *serialize* function, as well as the *serializeEntity* function as an example for the implementation of the serialization. Both methods use static code snippets, e.g., 'er2cds', 'entity', 'alias', and information of the ER2CDS model to generate the textual representation of the model.

```
export function serialize(model: ER2CDS): string {
    return expandToString`
        er2cds ${model.name}

        ${serializeEntities(model.entities)}

        ${serializeRelationships(model.relationships)}
    `;
}

export function serializeEntity(entity: Entity): string {
    return expandToString`
        entity ${entity.name} {
                ${entity.alias ? `alias ${entity.alias}` :
                    undefined}
                ${entity.attributes.length > 0 ? entity.attributes.
                    map(a => serializeAttribute(a)).join('\n') :
                    undefined}
            }
        `;
}
```

Listing 6.16: The serialization of a ER2CDS model to the textual representation of the model.

We follow a similar approach to the serialization for the generation of the CDS view entities. A template-driven model-to-text serialization transforms an ER2CDS model into a CDS view entity. The generation is triggered by a VS Code command, handled by the *ER2CDSCommandHandler* as presented in Listing 6.9. The VS Code command takes the URI of the currently opened document as an argument, which is furthermore propagated to the *generateCDS* function within the *ER2CDSCommandHandler*. The *generateCDS* function can be divided into the following steps:

1. **Extract model of document**
2. **Validate model**
3. **Generate CDS view entity source code**
4. **Write source code to file system**

The entry point of the source code generation is presented in Listing 6.17. All remaining functions follow the template-driven model-to-text transformation approach, similar to the *serializeEntity* function presented in Listing 6.16. The complete implementation of the *generateCDS* function can be found in Appendix A.

```
function generateSourceCode(model: ER2CDS): string | undefined
    {
    ...
    return expandToString`
        ${generateHeaderAnnotations(model)}
        ${generateHeader(model)}
            ${generateFromClause(model)}
            ${generateJoins(model)}
            ${generateAssociations(model)}
            ${generateAssociationsToParent(model)}
            ${generateCompositions(model)}
        {
            ${generateKeyAttributes(model)}${model.entities.find(e => e.attributes.find(a => a.type === 'key')) && model.entities.find(e => e.attributes.find(a => a.type !== 'key')) ? ',' : ''}
            ${generateAttributes(model)}${model.entities.find(e => e.attributes.find(a => a.type !== 'key')) && model.relationships.find(r => r.type === 'association' || r.type === 'association-to-parent' || r.type === 'composition') ? ',' : ''}
            ${generateAssociationAttributes(model)}
        }
    `;
}
```

Listing 6.17: Generation of a CDS view entity using a template driven model-to-text transformation.

Another key feature of ER2CDS is the import of existing CDS view entities. For this, we use the web service as presented in Section 6.3. The function is again triggered by a VS Code command, which is handled and propagated to the *importCDS* function in the *ER2CDSCommandHandler*. The command takes the name of the CDS view entity and the currently opened document as arguments. The latter is used to generate the output path of the imported entity. The CDS view entity name is required to request the information from the backend. On a high level, the *importCDS* function works as follows:

1. **Request CDS view entity information from web service**
2. **Transform response to ER2CDS model**
3. **Serialize the ER2CDS model to textual representation**
4. **Write ER2CDS source code to file system**

The transformation of the web service response to an ER2CDS model is based on the four entity sets *ImportSelectList*, *ImportCondition*, *ImportAssocDef*, and *ImportFromClause* presented in Section 6.3. The transformation can furthermore be divided into two steps: the transformation of the from-clause and the transformation of the associations. For both, we first create the entities of ER2CDS. This information is stored and retrieved using the *ImportCondition*. The entity set describes the structure of the CDS view entity, using the attribute *ExprType* to define each entry. To create the entities, we use *ExprType* equals 'TABLE_DATASOURCE'. Furthermore, for entities of the from-clause, we can create the attributes of an entity using the *ImportSelectList* entity set, again filtering entries on *ExprType*, in this case, 'ATOMIC'. The transformation of relationships is more complex and differs between from-clause and associations. In a nutshell, we again use the *ImportCondition* entity set to create a relationship incrementally. Depending on the *ExprType*, specific information about the relationship is extracted and added before finally adding the complete relationship to the model. For the from-clause relationships, we additionally use the *ImportFromClause* to define the join order; for the associations, we need to use the *ImportAssocDef* entity set in order to resolve aliases correctly. The full implementation of the *importCDS* is given in Appendix A.

To ensure the correctness of the models, we defined custom validation checks. As presented in line 19 of Listing 6.7 during the initial creation of the services, the *registerValidationChecks* function is called. The implementation of it can be found in Listing 6.18. We define and assign a method of the *ER2CDSValidator* class for each language construct.

```
export function registerValidationChecks(services:
   ER2CDSServices) {
    const registry = services.validation.ValidationRegistry;
    const validator = services.validation.ER2CDSValidator;

    const checks: ValidationChecks<ER2CDSAstType> = {
        ER2CDS: validator.checkER2CDS,
        Entity: validator.checkEntity,
        Attribute: validator.checkAttribute,
        DataType: validator.checkDataType,
        Relationship: validator.checkRelationship,
        RelationshipEntity: validator.checkRelationshipEntity,
        RelationshipJoinClause: validator.
           checkRelationshipJoinClause
    };

    registry.register(checks, validator);
}
```

Listing 6.18: Registration of validation checks.

These methods are then responsible for validating the given instance. In the most simple case, the validation of the ER2CDS root node is presented in Listing 6.19. It checks for the existence of the name of the node and, in case of an error, reports an error, including an error message. The remaining checks are implemented similarly. If a system is connected to ER2CDS, these validations also include checks against the data model of the SAP S/4HANA system. By default, the Langium framework publishes the validation result in the VS Code problems view.

```
export class ER2CDSValidator {
    ...
    async checkER2CDS(er2cds: ER2CDS, accept:
       ValidationAcceptor): Promise<void> {
        if (!er2cds.name) {
            accept('error', 'Name␣for␣ER2CDS␣missing', { node:
               er2cds, property: 'name' });
        }
    }
    ...
}
```

Listing 6.19: Validation on the root node of a ER2CDS model.

We also define a function for the graphical modeling interface to create so-called markers. Sprotty can display these markers accordingly. For the creation, the function analyzes the validation result of the language server and identifies the unique element identification of each. Furthermore, the severity of each diagnostic is mapped to the corresponding marker kind. The implementation is presented in Listing 6.20.

```
export function createMarkersForDocument(document:
   LangiumDocument): Marker[] | undefined {
    return document.diagnostics?.map(diagnostic => {
        const model = document.parseResult.value as ER2CDS;
        const elementId = findElementIdByRange(model,
           diagnostic.range);

        let kind: string;
        switch (diagnostic.severity) {
            case 1:
                kind = MarkerKind.ERROR;
                break;

            case 2:
                kind = MarkerKind.WARNING;
                break;

            case 3:
                kind = MarkerKind.INFO;
                break;

            default:
                kind = MarkerKind.INFO;
                break;
        }

        return <Marker>{
            kind: kind,
            elementId: elementId,
            description: diagnostic.message,
        }
    });
}
```

Listing 6.20: Creation of markers for the graphical modeling interface.

Finally, the ER2CDS language server implements a custom scope provider. This is primarily relevant for the textual modeling interface, as well as the validation of the model. In more detail, we define a custom behavior for the *firstAttribute* and *secondAttribute* properties of relationships. For the *firstAttribute* property of the relationship, the scope differs between relationships and associations; for associations, all attributes of all entities within the model are valid, while for relationships, only the attributes of the *source* entity of the relationship are within the scope. This follows the semantics of CDS. On the other hand, for the *secondAttribute* property, the scope is within the attributes of the *target* entity of the relationship. In all other cases, we rely on the default implementation of the Langium framework, which is furthermore based on the specified Langium grammar. The implementation of the *ER2CDSScopeProvider* is presented in Listing 6.21.

```
export class ER2CDSScopeProvider extends DefaultScopeProvider {
    ...
    public override getScope(context: ReferenceInfo): Scope {
        if (context.property === 'firstAttribute') {
            const relationship = context.container.$container
               as Relationship;
            let scope: Scope = null!;

            if (relationship.type === 'association' ||
               relationship.type === 'association-to-parent') {
                let attributeStreams: Stream<AstNodeDescription
                   > | undefined;

                relationship.$container.entities.forEach(e => {
                    const attributeStream = stream(e.attributes
                       .map(attr => this.services.workspace.
                       AstNodeDescriptionProvider.
                       createDescription(attr, attr.name)));

                    if (!attributeStreams) {
                        attributeStreams = attributeStream;
                    } else {
                        attributeStreams = attributeStreams.concat(attributeStream);
                    }
                })

                if (attributeStreams) {
                    scope = this.createScope(attributeStreams);
                }
            } else {
                const entity = relationship.source?.target.ref;
                if (entity)
                    scope = this.createScope(stream(entity.attributes.map(attr => this.services.workspace.AstNodeDescriptionProvider.createDescription(attr, attr.name))));
            }

            return scope;
        }

        if (context.property === 'secondAttribute') {
            const entity = (context.container.$container as Relationship).target?.target.ref;

            if (entity)
                return this.createScope(stream(entity.attributes.map(attr => this.services.workspace.AstNodeDescriptionProvider.createDescription(attr, attr.name))));
        }

        return super.getScope(context);
    }
}
```

Listing 6.21: Custom scope provider of ER2CDS, overriding the default behavior for relationship properties.

We rely on VS Code as an editor for the textual modeling interface and the Langium interface with the presented enhancements. The graphical modeling interface, on the other hand, has to be built from scratch using web development tools. The following section focuses on the implementation of this graphical modeling interface.

## 6.5 Webview

The webview is implemented using the Sprotty framework. The entry point is defined in the *webview.ts* file, which defines the *ER2CDSSprottyStarter* class. The *ER2CDSSprottyStarter* implements two methods *createDiagramContainer*, which defines the model elements and injects the different modules of the webview and *addVscodeBindings* to rebind default implementations to custom extensions. Furthermore, the *ER2CDSSprottyStarter* is used to start Sprotty eventually. In more detail, the *createDiagramContainer* is defined in a *di.config.ts* class used throughout the module for dependency injection. The implementation is illustrated in Listing 6.22. First, all model elements are bound to a specific view. Then, the default Sprotty modules are injected, following all custom modules used within the webview, e.g., *DiagramModule*. Finally, the base element and additional options are specified, and the resulting container is returned.

```
export default (containerId: string) => {
    const DiagramModule = new ContainerModule((bind, unbind,
       isBound, rebind) => {
        rebind(TYPES.ILogger).to(ConsoleLogger).
           inSingletonScope();
        rebind(TYPES.LogLevel).toConstantValue(LogLevel.info);

        const context = { bind, unbind, isBound, rebind };

        // Graph
        configureModelElement(context, GRAPH, ER2CDSRoot,
           ER2CDSRootView);

        // Nodes
        configureModelElement(context, NODE_ENTITY, EntityNode,
            EntityNodeView, { enable: [expandFeature] });
        configureModelElement(context, NODE_RELATIONSHIP,
           RelationshipNode, RelationshipNodeView);
        ...
    });

    const container = new Container();

    loadDefaultModules(container);

    container.load(DiagramModule);
    ...

    overrideViewerOptions(container, {
        needsClientLayout: true,
        needsServerLayout: true,
        baseDiv: containerId,
        hiddenDiv: containerId + '_hidden'
    });

    return container;
}
```

Listing 6.22: Dependency injection of the webview entry point, defining all model elements and modules.

The views themselves are implemented using Scalable Vector Graphics (SVG). Each view class implements a *render* method for displaying the element. In Listing 6.23, an example of such an implementation is given. The *EntityNodeView* is responsible for displaying an ER2CDS entity. It is rendered as a rectangle, as specified in Chapter 5.

```
@injectable()
export class EntityNodeView extends RectangularNodeView {
    override render(node: Readonly<EntityNode>, context:
       RenderingContext): VNode | undefined {
        if (!this.isVisible(node, context))
            return undefined;

        const height = 35;
        const rhombStr = 'M␣0,' + height + '␣␣L␣' + node.bounds
           .width + ',' + height;

        return (
            <g>
                <rect class-sprotty-node={true} class-mouseover
                   ={node.hoverFeedback} class-selected={node.
                   selected} x='0' y='0' rx='5' ry='5' width={
                   Math.max(node.bounds.width, 0)} height={Math
                   .max(node.bounds.height, 0)} />
                {context.renderChildren(node)}
                {(node.children[1] && node.children[1].children
                   .length > 0) ? <path class-comp-separator={
                   true} d={rhombStr} /> : ''}
            </g>
        );

    }
}
```

Listing 6.23: The view implementation for an ER2CDS entity using SVG.

For the ER2CDS webview we define a custom *ER2CDSDiagramServer*, handling the communication to the language server, a custom *ER2CDSDiagramWidget* to add custom logic on initialization and *ER2CDSKeyTool*, *ER2CDSMouseTool* and *ER2CDSScrollMouseListener* for custom behavior on user input. The Sprotty VS Code integration already defines all these; therefore, we have to rebind them in the *addVscodeBindings* method. However, not all defaults are suitable for our use case. In the following section, we specify the usage of these custom implementations.

The *ER2CDSDiagramServer* is necessary to define the extension of the LSP messages. All custom messages must be registered at the *ER2CDSDiagramServer*. Furthermore, it allows us to define the handling of these messages. Since Sprotty is also based on communicating with *Actions*, one has to define which messages are handled locally and which are sent to the language server. Listing 6.24 exemplifies the implementation of the *ER2CDSDiagramServer*. The *initialize* method defines the custom messages. In contrast, the *handleLocally* method defines which *Action* is handled locally or by the language server. It is important to note that *true* corresponds unintentionally to handling by the language server rather than locally.

```
@injectable()
export class ER2CDSDiagramServer extends VscodeLspEditDiagramServer {
    public override initialize(registry: ActionHandlerRegistry): void {
        super.initialize(registry);

        registry.register(CreateElementAction.KIND, this);
        ...
    }

    public override handleLocally(action: Action): boolean {
        switch (action.kind) {
            case CreateElementAction.KIND:
                return true;
            ...
            default:
                return super.handleLocally(action);
        }
    }
}
```

Listing 6.24: Implementation of the ER2CDSDiagramServer within the Sprotty webview.

The *ER2CDSDiagramWidget* is used to dispatch custom actions on initialization. In particular, the *EnableToolPaletteAction* and *EnableEditorPanelAction* are dispatched to extend the user interface. Furthermore, the method is responsible for initially requesting the current model from the language server. Additionally, it executes a *FitToScreenAction* after the model is returned. The implementation itself is straightforward and shown in Listing 6.25.

```
@injectable()
export class  ER2CDSDiagramWidget extends VscodeDiagramWidget {
    @postConstruct()
    override initialize(): void {
        super.initialize();
    }

    protected override initializeSprotty(): void {
        if (this.modelSource instanceof DiagramServerProxy)
         this.modelSource.clientId = this.diagramIdentifier.clientId;

        this.requestModel().then(() => this.actionDispatcher.dispatch(FitToScreenAction.create([])));

        this.actionDispatcher.dispatch(EnableToolPaletteAction.create());
        this.actionDispatcher.dispatch(EnableEditorPanelAction.create());
    }
}
```

Listing 6.25: Implementation of the ER2CDSDiagramWidget within the Sprotty webview.

The *ER2CDSKeyTool* and *ER2CDSMouseTool* both extend the default *KeyTool*/*MouseTool* implementation without overriding anything. However, it is necessary in particular for the *ER2CDSKeyTool*, since the default *addVscodeBindings* implementation uses a *DisabledKeyTool*, which does not allow any key input. Therefore, it is not possible to intercept any keyboard events. Also, the implementation serves as a placeholder for future work.

Finally, the *ER2CDSScrollMouseListener* is implemented to prevent scrolling while the marquee tool is enabled. This allows the user to select multiple elements. The remainder of this section presents a more detailed overview of all implemented tools.

ER2CDS also implements central services, which can be used throughout the module. Specifically, to notify different modules about model changes from the language server, we implemented a custom *ER2CDSCommandStack*, as well as *DiagramEditorService*, as the central model service.

The *ER2CDSCommandStack* overrides the default Sprotty *CommandStack*. The *CommandStack* is responsible for executing all commands, as presented in Chapter 2. For our purposes, we intercept the *SetModelCommand* and *UpdateModelCommand* and notify the *DiagramEditorService* about the new model root, as presented in Listing 6.26. One can also see the usage of the lazy injector, which handles circular dependencies.

```
@injectable()
export class ER2CDSCommandStack extends CommandStack {
    @inject(LazyInjector)
    protected lazyInjector: LazyInjector;

    override execute(command: ICommand): Promise<SModelRootImpl> {
        const result = super.execute(command);

        if (command instanceof SetModelCommand || command instanceof UpdateModelCommand)
            result.then(root => this.diagramEditor.notifyModelRootChanged(root, this));

        return result;
    }

    get diagramEditor(): DiagramEditorService {
        return this.lazyInjector.get(DiagramEditorService);
    }
}
```

Listing 6.26: Custom Sprotty CommandStack notifying the DiagramEditorService about changes on the model.

The *DiagramEditorService* then exposes methods to access the current model root. Furthermore, it can inform subscribers of any model changes using events. Additionally, the *DiagramEditorService* is responsible for keeping track of all currently selected elements within the diagram. In Listing 6.27, the most essential methods are shown. Lines 4 - 19 illustrate methods to inform about model changes and the current model, while lines 21 - 27 concern currently selected elements within the model.

```
1  @injectable()
2  export class DiagramEditorService {
3      ...
4      notifyModelRootChanged(root: Readonly<SModelRootImpl>, notifier: object): void {
5          if (!(notifier instanceof CommandStack)) {
6              throw new Error(’Invalid␣model␣root␣change␣notification.␣Notifier␣is␣not␣an␣instance␣of␣‘CommandStack‘.’);
```

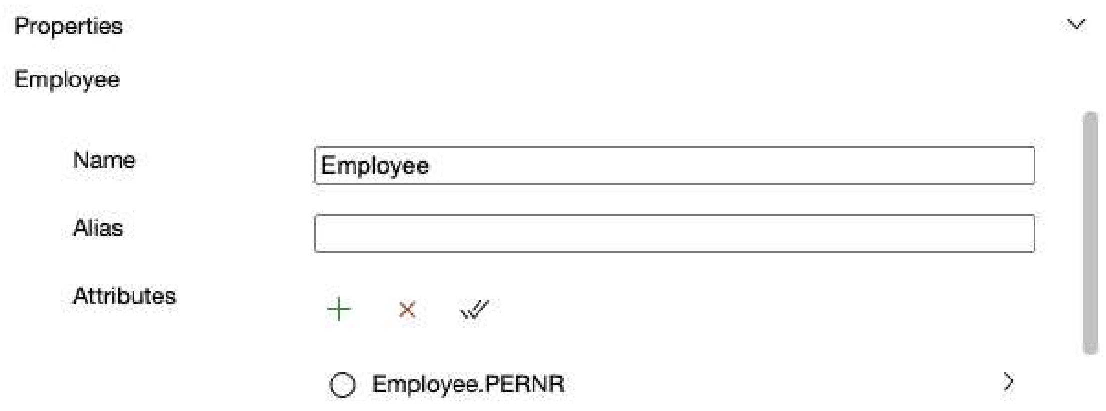
```
        }

        this.root = root;
        this.onModelRootChangedEmitter.fire(root);
    }

    getModelRoot(): Readonly<SModelRootImpl> {
        return this.root;
    }

    get onModelRootChanged(): Event<Readonly<SModelRootImpl>> {
        return this.onModelRootChangedEmitter.event;
    }

    updateSelection(newRoot: Readonly<SModelRootImpl>, select:
       string[], deselect: string[]): void {
        ...
    }
    ...
    getSelectedElementIDs(): string[] {
        return [...this.selectedElementIDs];
    }
    ...
}
```

Listing 6.27: DiagramEditorService as the central service for handling the model changes and current selections.

These central services, especially the *DiagramEditorService*, are available throughout the module using dependency injection.

We will now focus on the user interface. First, the main elements, the editor panel and tool palette, are presented. Then, additional elements concerning the user interface are introduced.

The *EditorPanel* serves as the main container for the diagram. It is divided into two parts, the main modeling container, and the property panel at the bottom of the screen. The modeling container itself is an empty container in which the current model is rendered. On the other hand, the property palette contains different elements depending on the currently selected element. For this, the palette gets refreshed on each model change, using the *DiagramEditorService* and the *SelectAction* propagated by Sprotty. A custom implementation for all different element types exposes input elements within the property palette. Currently, the property palette supports an input element with value help, requesting the values from the language server, a checkbox for boolean values, a select option for predefined choices, an input without any value help, and a reference element



Properties

Employee

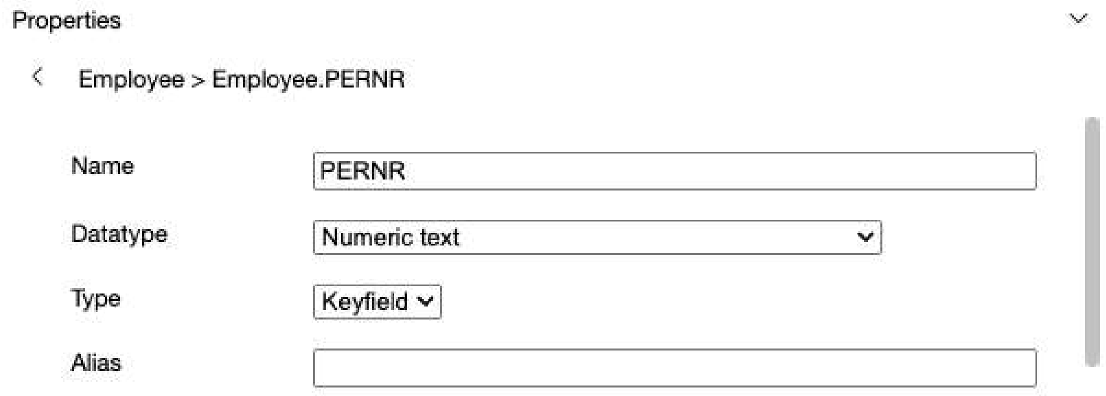
Name: Employee

Alias:

Attributes

○ Employee.PERNR ›

Figure 6.2: Property palette for a ER2CDS entity.



Properties

‹ Employee > Employee.PERNR

Name: PERNR

Datatype: Numeric text

Type: Keyfield

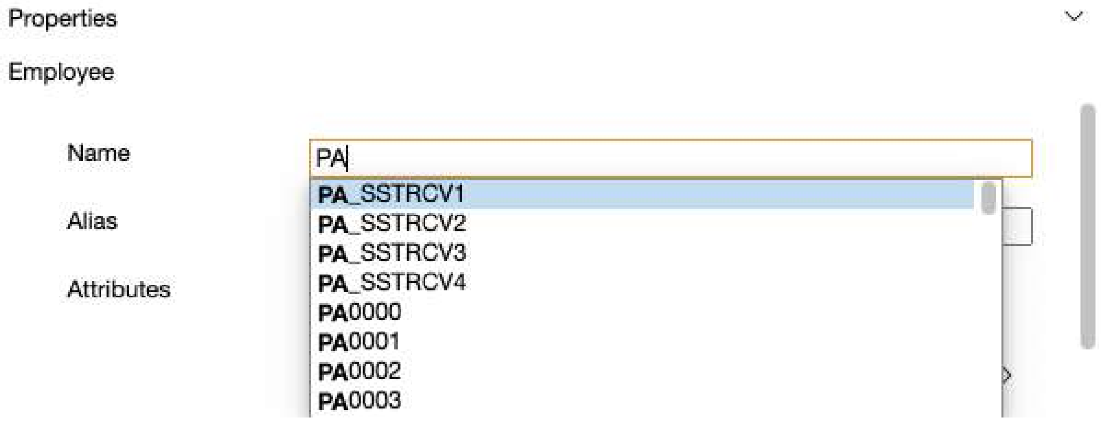
Alias:

Figure 6.3: Property palette for a ER2CDS entity attribute.

for multiple attributes and navigation purposes. An example of a property palette with two inputs and a reference component, allowing to add, delete, and navigate to the detail view, is presented in Fig. 6.2. Fig. 6.3 then shows the detailed view of one attribute, with two more input fields and two select options. In Fig. 6.4, an example of using the value-help with values requested from the language server is presented.

The tool palette is yet another user interface element that hosts the model editing tools for ER2CDS. It is shown in Fig. 6.5. The tool palette is separated into two parts: the header bar for element-independent tools; the tool palette body for element-dependent tools. For the header bar, the following tools are available starting from the left:

- **Default tools**: Tools that are enabled by default. ER2CDS defines the *MarqueeKeyTool* and *DeleteKeyTool* as default tools. The first allows to enable the *MarqueeMouseTool* using 'Shift', the latter allows to delete elements using the 'Delete' or 'Backspace' key.



Properties

Employee

Name PA

PA_SSTRCV1
PA_SSTRCV2
PA_SSTRCV3
PA_SSTRCV4
PA0000
PA0001
PA0002
PA0003

Alias

Attributes

Figure 6.4: Property palette with an input field and value help.

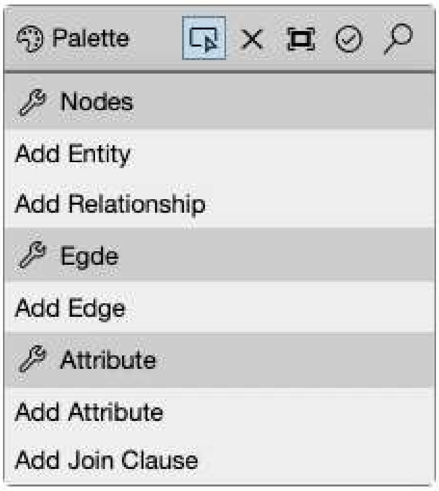


Figure 6.5: Tool palette hosting the modeling tools of ER2CDS.

- **DeleteMouseTool**: Deletion of an element on mouse click.
- **MarqueeMouseTool**: Selection of multiple elements by dragging the mouse.
- **Validation**: Triggers a validation request to the language server.
- **Search**: Allows to search within the tool palette.

For the body of the tool palette, the tools are separated by the corresponding elements. All of the buttons trigger the enabling of the corresponding tool, except *Add Entity* and *Add Relationship*, which directly create a *CreateElementAction*. Generally, the tools are managed by the *ToolManagerActionHandler*, which injects all available tools, intercepts all enable-actions, and calls the enable or disable method of the tool accordingly.

An example implementation for such a tool is presented in Listing 6.28. First, the *ToolManagerActionHandler* intercepts the *EnableDeleteMouseToolAction* and calls the enable method on the *DeleteMouseTool* (line 9 - line 19). On enable, the *DeleteMouseTool* registers a custom mouse listener on the central *ER2CDSMouseTool* (line 30 - line 32), which listens for *mouseUp* events and triggers a *DeleteElementAction* with the selected element (line 38 - line 48).

```
 1 @injectable()
 2 export class ToolManagerActionHandler implements IActionHandler
       {
 3     ...
 4     @inject(DeleteMouseTool)
 5     private deleteMouseTool: DeleteMouseTool;
 6     ...
 7
 8     handle(action: Action): void | ICommand | Action {
 9         this.disableAllTools();
10
11         switch (action.kind) {
12             ...
13             case EnableDeleteMouseToolAction.KIND:
14                 this.deleteMouseTool.enable();
15                 break;
16             ...
17         }
18     }
19     ...
20 }
21
22 @injectable()
23 export class DeleteMouseTool {
24     @inject(ER2CDSMouseTool)
25     protected mouseTool: ER2CDSMouseTool;
26
27     protected deleteMouseToolListener: DeleteMouseToolListener
           = new DeleteMouseToolListener();
28
29     enable(): void {
30         this.mouseTool.register(this.deleteMouseToolListener);
31     }
32     ...
33 }
34
35
```

```
@injectable()
export class DeleteMouseToolListener extends MouseListener {
    override mouseUp(target: SModelElementImpl, event: MouseEvent): Action[] {
        const deletableParent = findParentByFeature(target, isDeletable);
        if (deletableParent === undefined) {
            return [];
        }

        const result: Action[] = [];
        result.push(DeleteElementAction.create([deletableParent.id]));

        return result;
    }
}
```

Listing 6.28: Implementation of the DeleteMouseTool as an example for the tool implementations.

All remaining tools are implemented similarly. The mapping between the tool palette description and tool implementation is presented in Table 6.1.

| Tool palette description | Tool implementation |
|---|---|
| Add Entity | - |
| Add Relationship | - |
| Add Edge | EdgeCreateTool |
| Add Attribute | AttributeCreateMouseTool |
| Add Join Clause | JoinClauseCreateMouseTool |

Table 6.1: Mapping between the tool palette descriptions and the tool implementation.

Another user interface component is the popup to confirm the retrieval of all attributes of an entity from the SAP S/4HANA system. The popup is created using the *SetPopupModelAction* sent from the language server. Within the webview, we rebind the default *PopupMouseTool* to the custom *ER2CDSPopupMouseTool*, which intercepts the user decision and sends *CreateElementExternalAction* message to the language server in case of a positive decision. Fig. 6.6 shows the rendering of the popup, while Listing 6.29 illustrates the implementation of the *ER2CDSPopupMouseTool*. It is important to note that the language server creates and returns the popup elements in the form of the

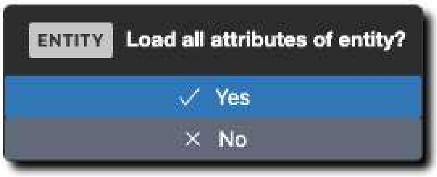


Figure 6.6: Popup to confirm the loading of all attributes of an entity.

*SetPopupModelAction*. For this particular use case, the *SetPopupModelAction* is returned for the custom *RequestPopupConfirmModelAction* message.

```
@injectable()
export class ER2CDSPopupMouseTool extends PopupMouseTool {
    constructor(@inject(TYPES.IActionDispatcher) protected actionDispatcher: IActionDispatcher, @multiInject(TYPES.PopupMouseListener) @optional() protected override mouseListeners: MouseListener[] = []
    ) {
        mouseListeners.push(new PopupButtonListener(actionDispatcher));
        super(mouseListeners);
    }
}

@injectable()
export class PopupButtonListener extends MouseListener {
    ...
    override mouseDown(target: SModelElementImpl, event: MouseEvent): (Action | Promise<Action>)[] {
        if (!(target instanceof PopupButton)) return [];

        if (target.type === 'button:yes') {
            this.actionDispatcher.dispatch(CreateElementExternalAction.create(target.target));
        }

        const actions: Action[] = [];
        actions.push(SetPopupModelAction.create(EMPTY_ROOT));
        return actions;
    }
}
```

Listing 6.29: Implementation of the ER2CDSPopupMouseTool handling the user decision for loading attributes from the SAP S/4HANA system for an entity.

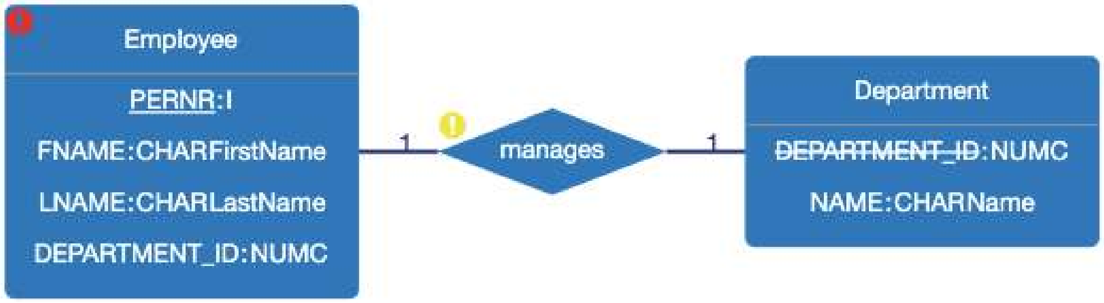


Figure 6.7: Validation markers for warnings and errors.

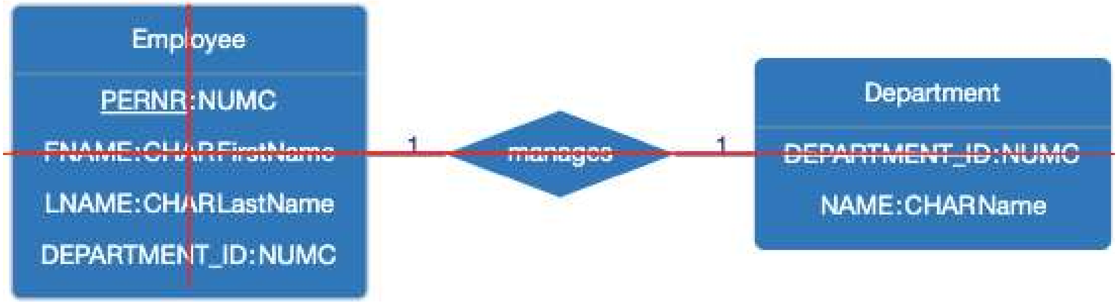
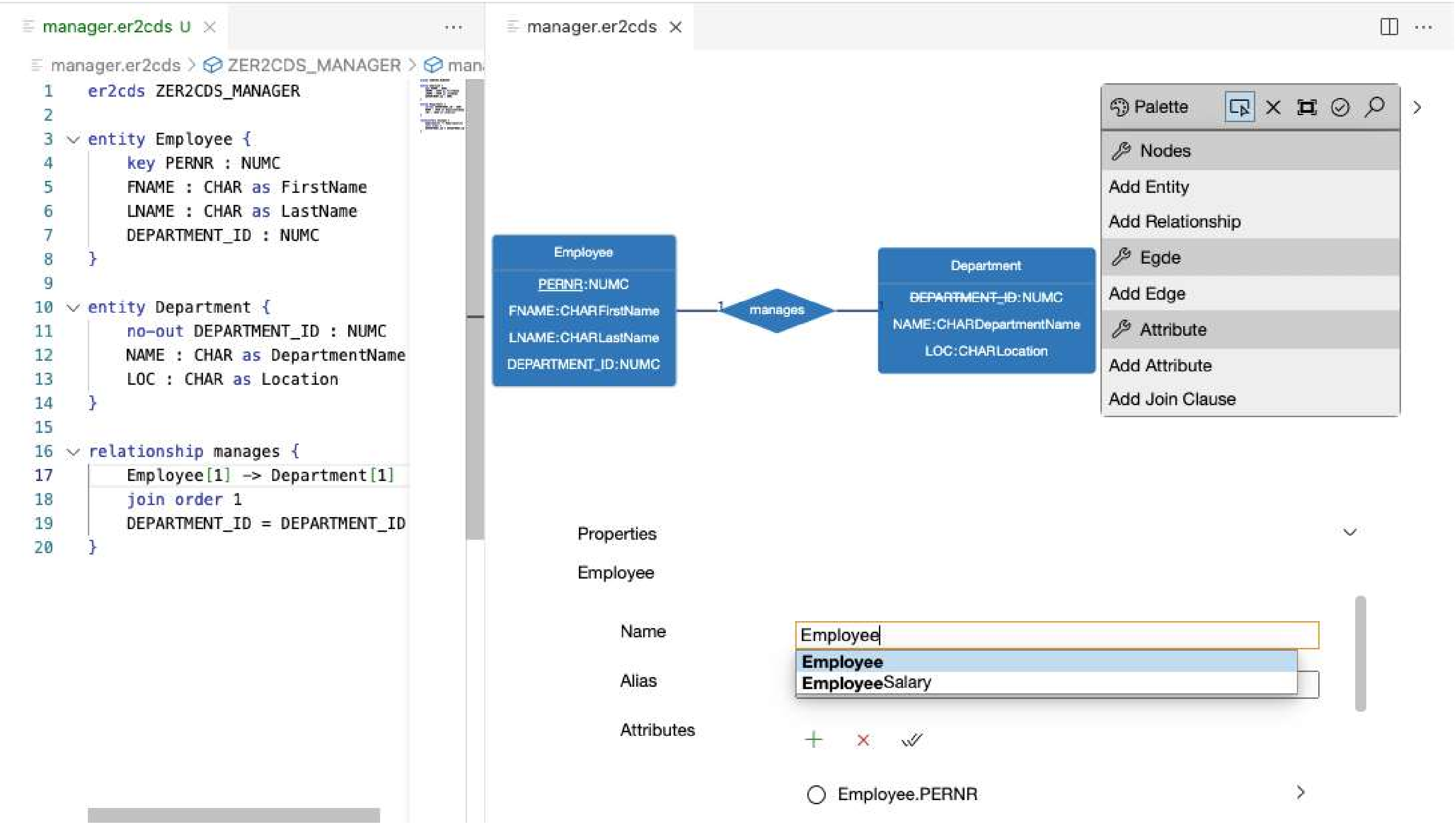


Figure 6.8: Helper lines to support the alignment of model elements.

The validation is displayed in the webview using markers. For this, a custom *SetMarkersActionHandler* is implemented, which intercepts all *SetMarkersAction* issued by the language server, as presented in Section 6.4. First, all previous markers are deleted from the elements using the *DeleteMarkersAction*, which corresponds to a Sprotty action and is handled locally. Then, the *ApplyMarkersAction* with all markers received by the language server is dispatched. This creates an issue marker on each corresponding element. Furthermore, the *SetMarkersActionHandler* is responsible for requesting the markers on each model change using the custom *RequestMarkersAction*. In Fig. 6.7, an example of an error marker on the *Employee* entity and a warning marker on the *manages* relationship is shown.

Finally, ER2CDS implements a helper line tool, which improves the graphical modeling experience. It renders additional lines, which help to align the model elements. For this, the *HelperLineManager* intercepts all Sprotty *MoveAction* and *SetBoundsAction*. For the *MoveAction*, the *finished* attribute decides whether the current helper lines are removed or new helper lines should be created. Both behaviors again correspond to a Sprotty action, *RemoveHelperLinesAction* and *DrawHelperLinesAction*. For the latter, the corresponding *DrawHelperLinesCommand* calculates the helper line positions and adds them to the model root. On the other hand, the *RemoveHelperLinesCommand* removes all elements of the helper line type. The result of the helper line tool is illustrated in Fig. 6.8. The red helper lines should support the alignment of the selected *Employee* entity.

Finally, Fig. 6.9 presents a complete overview of the ER2CDS modeling tool, integrated into VS Code with the textual and graphical editor, as presented in this chapter.



```
er2cds ZER2CDS_MANAGER

entity Employee {
    key PERNR : NUMC
    FNAME : CHAR as FirstName
    LNAME : CHAR as LastName
    DEPARTMENT_ID : NUMC
}

entity Department {
    no-out DEPARTMENT_ID : NUMC
    NAME : CHAR as DepartmentName
    LOC : CHAR as Location
}

relationship manages {
    Employee[1] -> Department[1]
    join order 1
    DEPARTMENT_ID = DEPARTMENT_ID
}
```

Properties

Employee

Name: Employee
Employee
EmployeeSalary

Alias

Attributes

○ Employee.PERNR

Figure 6.9: The ER2CDS modeling tool integrated in VS Code.

# CHAPTER 7

# Evaluation

This chapter focuses on the evaluation of ER2CDS. The evaluation itself is twofold. First, a controlled experiment is conducted. The creation and import of CDS view entities are evaluated. For the creation, 20 existing entities are modeled, and the corresponding CDS view entity is generated. The result is then compared to the existing implementation. For the import, 100 view entities are imported to ER2CDS and evaluated. The second part of the evaluation is a case study. The main focus is hereby on the usability of the tool. The users are presented with three different tasks that shall be modeled using ER2CDS. Afterward, a survey is conducted to collect feedback regarding the modeling experience. Finally, the last section of this chapter discusses the experiment and case study results.

## 7.1 Controlled Experiment

This section presents the controlled experiment, focusing on creating CDS view entities using ER2CDS and importing existing CDS view entities to ER2CDS. First, the experiment setup is described, and then the results are presented. The discussion of the results can be found in the last section of this chapter.

### 7.1.1 Setup

For the experiment, we focus on the two main functionalities: the creation and modeling feature and the import feature of ER2CDS.

For the first part of the experiment, regarding the creation of CDS view entities, 20 existing view entities are analyzed and modeled using ER2CDS (see Table 7.1 for descriptive statistics of the dataset). Afterward, the model is used to generate a CDS view entity. The textual output is then compared to the original view entity. Furthermore, the resulting output is imported to the SAP S/4HANA system of the original view and evaluated regarding the correctness of syntax and the output of the CDS view entity compared to

| Metric | Modeled/Exported Models | | | | Imported Models | | | | Case Studies | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Min. | Max. | Avg. | Std. Dev. | Min. | Max. | Avg. | Std. Dev. | Task1 | Task2 | Task3 |
| **Relations** | 0 | 23 | 5.55 | 6.12 | 0 | 30 | 3.89 | 5.00 | 0 | 2 | 6 |
| **Elements** | 3 | 128 | 31.05 | 30.79 | 2 | 325 | 34.38 | 49.35 | 5 | 8 | 13 |
| **Lines of Code** | 16 | 221 | 65.30 | 53.51 | 31 | 1115 | 116.76 | 132.40 | 16 | 20 | 27 |

Table 7.1: Descriptive statistics of the models in our evaluations

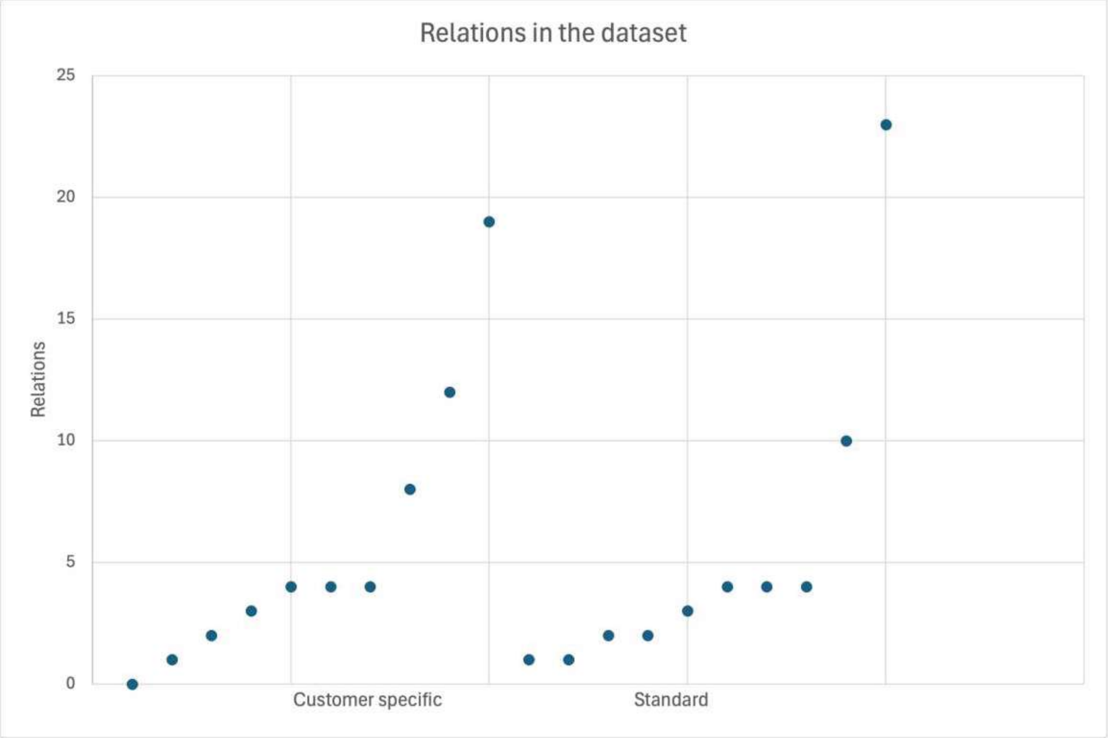


Figure 7.1: Dataset for creating CDS view entities.

the original. The dataset for this part of the experiment consists of 10 standard CDS view entities created by SAP and ten custom CDS view entities implemented by companies using an SAP S/4HANA system. The dataset is additionally described using relations corresponding to a CDS view entity's count of relations. Fig. 7.1 presents an overview of the dataset used. Within the first part of the diagram, the customer-specific CDS view entities are described concerning the count of the relations of each. The second part presents the standard CDS view entities, each with the count of relations.

The second part of this experiment focuses on the import of existing CDS view entities to ER2CDS. One hundred randomly selected standard CDS view entities are imported

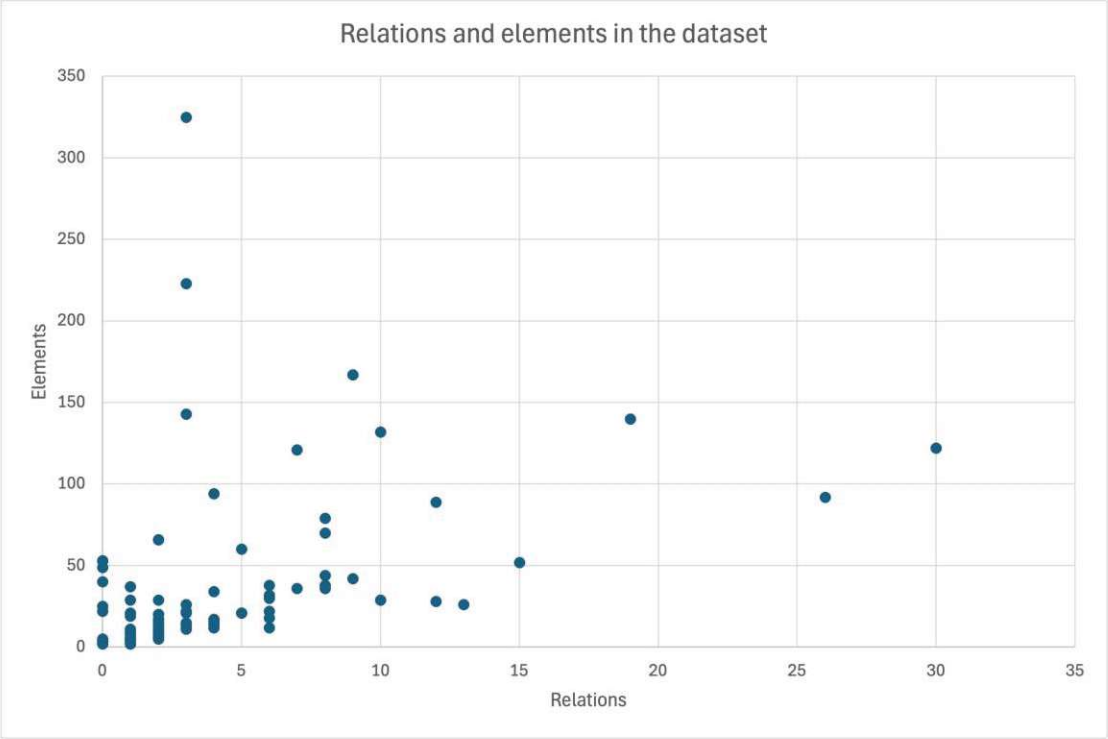


Figure 7.2: Dataset for the import of CDS view entities.

to ER2CDS and evaluated for this (see Table 7.1 for descriptive statistics of the dataset). The evaluation focuses on the capability to import and display the CDS view entity. Furthermore, the errors and warnings issued by ER2CDS are documented. Finally, the correctness of the imported CDS view entity is checked, and the resulting model is compared to the original representation and intention. The dataset again is categorized using the count of relations. Additionally, the count of elements a CDS view entity has is used to classify the import. The count of elements results in the attributes and the associations that are reexported. In Fig. 7.2, the dataset is presented. The x-axis describes the count of relations, while the y-axis corresponds to the count of elements.

### 7.1.2 Results

The results are again presented separately. First, we focus on the creation of CDS view entities. The results are presented in Fig. 7.3 and Fig. 7.4. As one can easily see, all modeled CDS view entities output a syntactically correct CDS view entity. However, compared to the original view, only 15 of 20 are evaluated positively regarding the correctness of the output itself. ER2CDS can model and generate all the customer-specific view entities but only 5 of 10 standard view entities.

The main reason for the incorrect output for some of the standard CDS view entities is the lack of support for specific, less commonly used, language constructs. These issues,

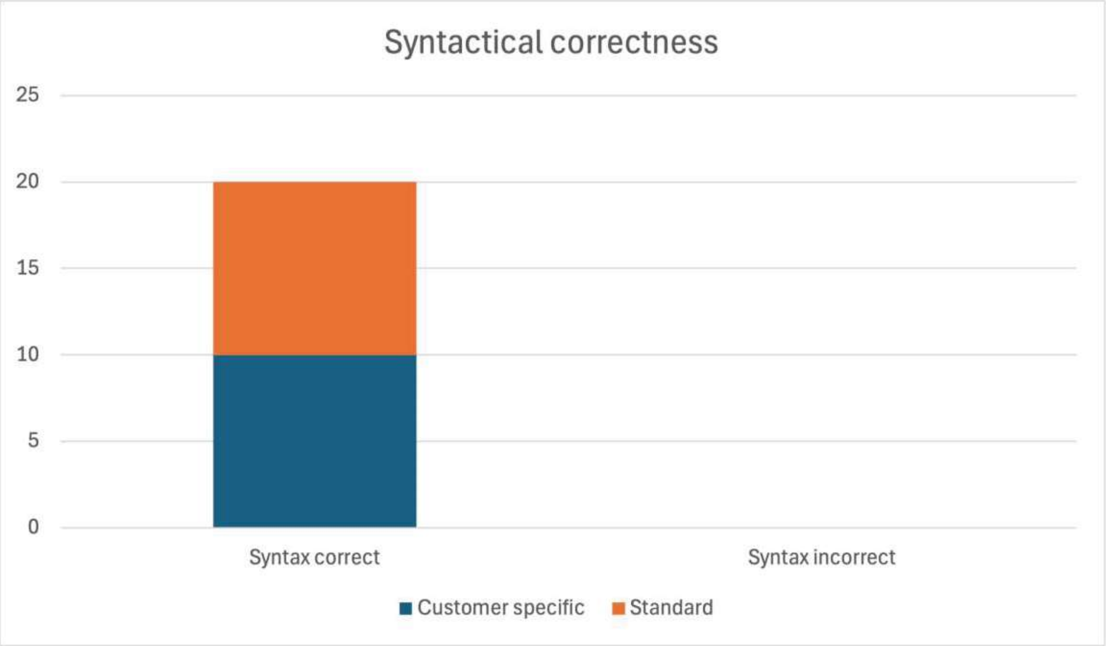


Figure 7.3: Correctness of syntax for the created CDS view entities using ER2CDS.

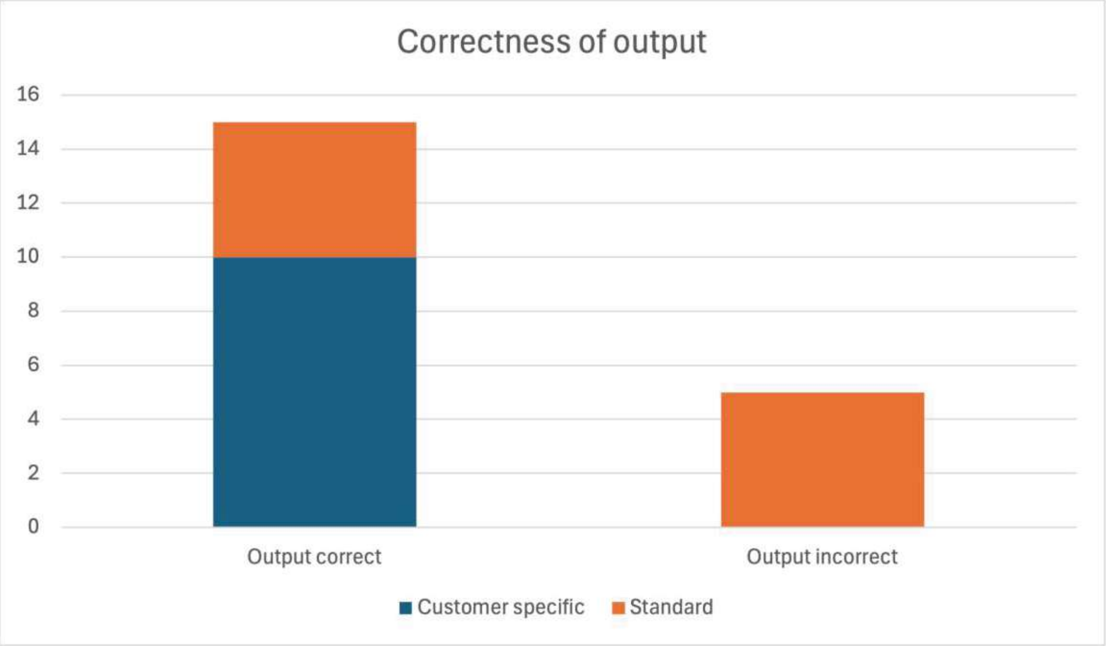


Figure 7.4: Correctness of output for the created CDS view entities using ER2CDS.

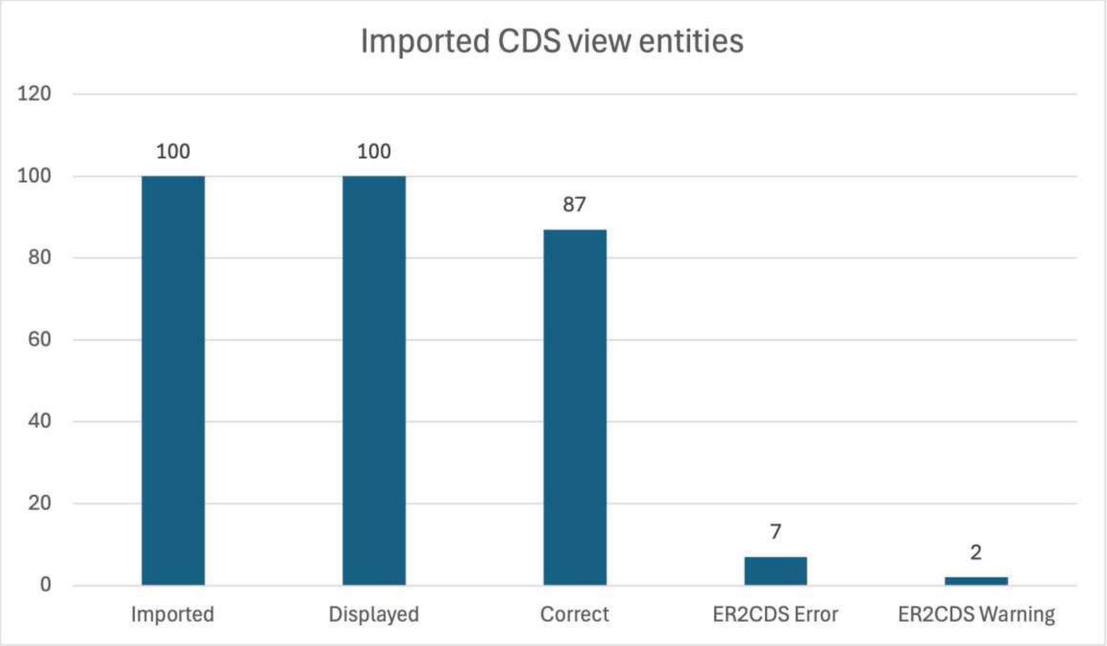


Figure 7.5: Results of the import of existing CDS view entities to ER2CDS.

or even full support of all language features of CDS, should be addressed in future work. Firstly, ER2CDS currently does not support *WHERE*-Clauses. In the next version of the tool this will be supported by an additional attribute on the root element level. Secondly, the reexport of sub-associations, as well as the usage of specific attributes of associations is currently not supported. Again, this will be supported in a future version of the tool, by allowing explicit definition of attributes and association from associated views instead of exporting the association completely. Thirdly, no join clause with fixed value is allowed in ER2CDS. This can be fixed by allowing literals for join clauses, instead of attributes only. Finally, multiple associations to the same table are not resolved correctly in ER2CDS, due to faulty handling of aliases. A table with the detailed results and descriptions of the encountered problems can be found in Appendix B.

Regarding the import of existing CDS view entities, the full result is presented in Fig. 7.5. ER2CDS was able to import all view entities of the dataset successfully. Furthermore, all of the evaluated examples could also be rendered. However, the import was incorrect for 13 of 100 CDS view entities. The reasons, as well as possible fixes, are listed below:

- **Join clause with fix value not supported**: As discussed before, join clauses should support literals, instead of attributes only. If supported the import feature can be adapted straight forward.
- **Join clause with fix value not supported**: As discussed before, join clauses should support literals, instead of attributes only. The import can then

- **Join clause on association attribute not supported**: Similar to the *element_list*, attributes of association are supported in join clauses. The import needs to resolve the origin of the attribute correctly and add it to the corresponding entity subsequently.

- **Upper-/Lowercase of alias in from clause**: Currently, the aliases provided from the SAP service are case-insensitive. In the future the web service shall provide these case-sensitive.

- **Hierarchical associations not supported**: Due to the faulty handling of aliases discussed before, hierarchical structures in associations can not be modelled. However, the import does resolve them correctly. By addressing the issue within the modeling language itself, this problem is also resolved.

- **Multiple Relations to the same entity-name. Alias is not resolved correctly**: Similar problem, as well as solution, to the previous point.

- **Self-associations not supported**: Similar problem, as well as solution, to the previous two points.

- **Only Equal-Sign is supported in Join Clause**: Currently, the ER2CDS join clause only supports equal-signs. The fix can be addressed straight forward, by adding other possibilities to the language.

- **Union is not supported**: This issue has to be addressed on modeling language level first. Currently, there is no syntax supporting a *UNION*. This feature will be supported in a future release of ER2CDS.

It is easy to see that some of the problems occur on multiple occasions. Furthermore, most problems result from a lack of support for this version of ER2CDS. These shall be addressed in future work, striving for full language support for CDS. Moreover, due to the incorrect import of some CDS view entities, ER2CDS also reports errors for seven models and warnings for two of the models. Regarding the errors ER2CDS reports *Relationship has no join clauses* for seven models, which can be directly traced back to the issues with join clauses discussed above. Additionally, ER2CDS reports incompatible attributes, based on their datatype, used within join clauses for two models as a warning. The detailed report of the results can be again found in Appendix B.

## 7.2 Case Study

In this section, the case study conducted within this thesis will be presented. First, the setting of the case study is introduced. Then, the survey to obtain the users' feedback is presented. Finally, we analyze the results of the case study.

### 7.2.1 Setting

For the case study, the participants are presented and asked to model three tasks with increasing difficulty with ER2CDS. Afterward, we surveyed the group of participants to gather feedback and insights. The task description for the case study is presented in Appendix B. The survey can be divided into three parts. After the consent to participate, the first question asks for the participant's role, which is either developer or business expert. Then, six questions regarding the three tasks of the case study, for each, if the participant could implement the given task using ER2CDS and if the participant could implement the given task using the standard textual syntax of CDS. Each question provides possible answers using the Likert scale [JKCP15]. Following are the questions of the system usability score (SUS) as introduced in [B+96]. This enables us to quantify the usability of ER2CDS. Afterward, the participant is asked for a preference regarding ER2CDS or the standard textual syntax of CDS. Finally, the survey ends with three open questions for positive or negative feedback and suggestions for future improvement of ER2CDS. The complete survey can be found in Appendix B.

### 7.2.2 Results

The case study was conducted with eight participants: four CDS developers (three with three to five years relevant experience, one with 10+ years of relevant experience) and four business experts (one with up to three years relevant experience, one with five to ten years of relevant experience, and two with more than 10 years of relevant experience). As introduced above, each participant implemented three different tasks using ER2CDS and then took part in the survey. We will now analyze the survey results.

First, the responses regarding the three tasks are presented in Fig. 7.6 and Fig. 7.7. As one can easily see, all of the participants were able to create the CDS view entity for Task 1 and Task 2 using ER2CDS, while only five of the participants thought they could implement the same tasks using the textual syntax of CDS. Even fewer participants (2) assessed themselves as being able to implement Task 3 using the textual syntax of CDS. In contrast, seven out of eight participants could still implement it using ER2CDS.

Next, we will calculate SUS using the survey responses. The calculation follows the scheme presented below and results directly from the structure of questions, with odd numbered question being formulated positively and even numbered questions being formulated negatively[B+96]:

1. **Calculate each score contribution**:
   - **Odd numbered questions**: Scale position - 1
   - **Even numbered questions**: 5 - scale position
2. **Sum up score contributions**
3. **Multiply sum of scores by 2,5**

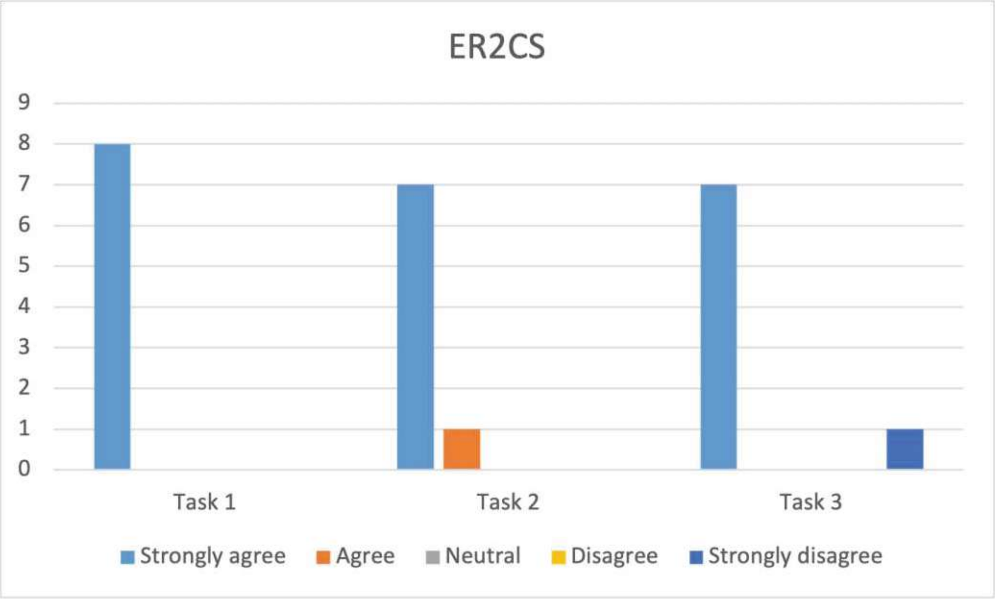


Figure 7.6: I was able to create Task 1/2/3 using ER2CDS.

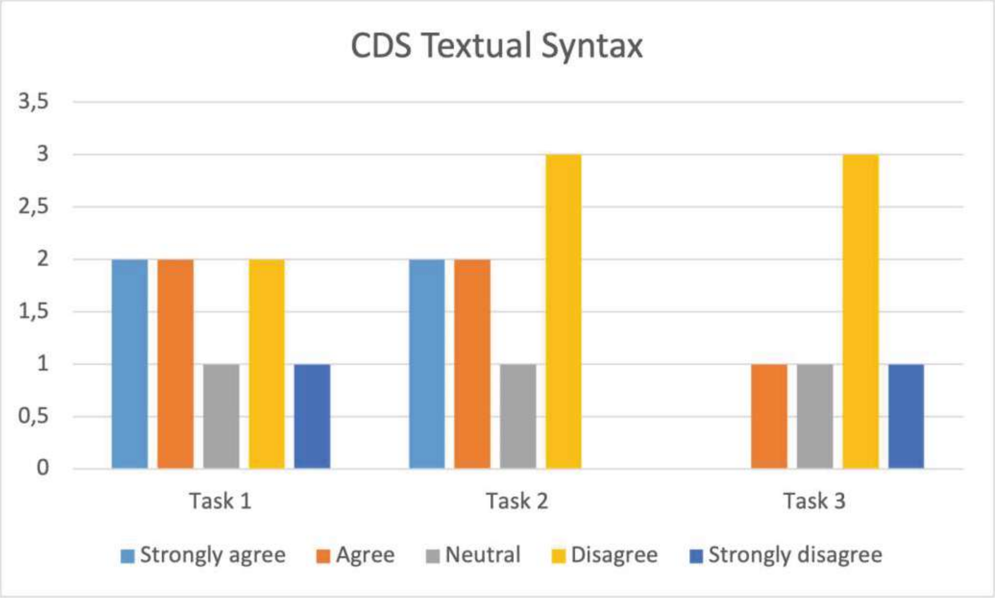


Figure 7.7: I would be able to create Task 1/2/3 by using only the textual syntax of CDS.

The SUS for each participant is presented in Table 7.2. In order to get an overall SUS, we calculate the average over these eight values, resulting in an overall SUS of 86,25. Classifying this score with respect to the studies analyzed in [BKM08], this refers to an above-average SUS.

| Participant | SUS |
|---|---|
| 1 | 95 |
| 2 | 82,5 |
| 3 | 95 |
| 4 | 95 |
| 5 | 87,5 |
| 6 | 100 |
| 7 | 70 |
| 8 | 65 |

Table 7.2: SUS calculated from the responses of the participants

Next, Fig. 7.8 presents the response for the preference between ER2CDS and the textual syntax of CDS. It is significant to note that all of the business experts who participated in the case study prefer to use ER2CDS over the textual syntax of CDS in the future. Also, two of the developers participating are in favor of ER2CDS.

Finally, we will present an unordered list of the responses to the open questions regarding the positive and negative feedback and suggestions for future improvements.

**Positive feedback**

- Simple to use
- Enables the creation of CDS Views without the need for the technical background and coding knowledge
- The offer of a textual and visual editor
- The visual editor with the input fields to define entities/attributes/relationships / ...
- I was able to intuitively create CDS views, without any further instructions.
- Textual and graphical editor. Integration of SAP data model.
- Creation of CDS views on a higher abstraction level. Easy and fast to create simple CDS views.

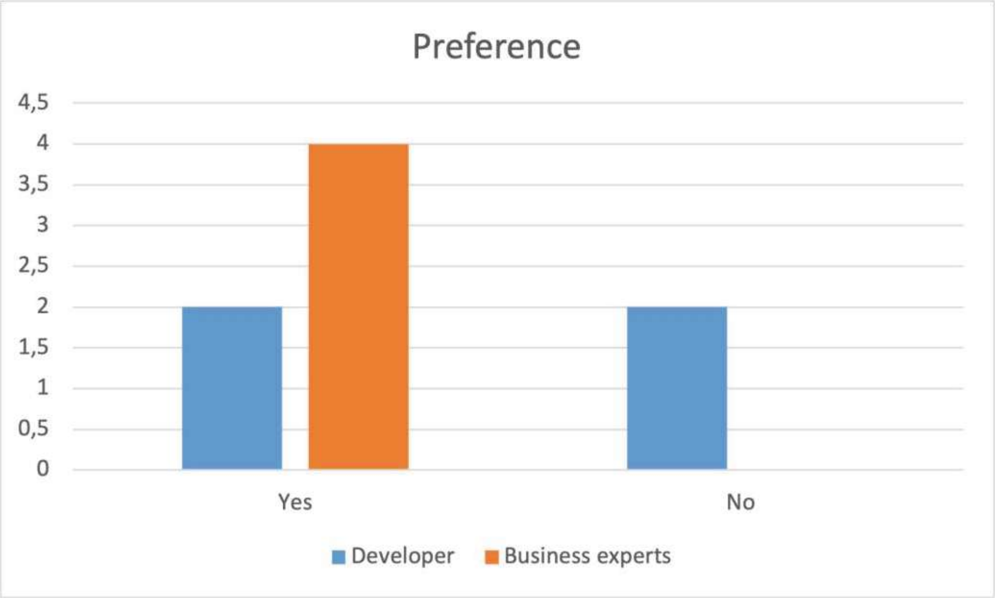


Figure 7.8: In the future I will use ER2CDS over the textual syntax to create CDS view entities.

### Negative feedback

- No explanation for relationship types
- All attributes can only be added once (workaround possible)
- Missing integration in BAS.
- Still technical knowledge is required.

### Suggestions for improvement

- Automatic scrolling when adding and editing attributes
- SAP Tool integration
- List the attributes in the visual editor to select them
- Support for all language features of CDS.
- Add a beginner's guide or documentation.

The full dataset from the conducted survey can be found in Appendix B. In the next section, we will discuss the experiment's findings and the case study further.

## 7.3 Discussion

The evaluation of ER2CDS has shown the effectiveness and usefulness of the tool within the context. However, it also pointed out some of the tool's weaknesses in this early stage of development. The experiment for creating and importing existing CDS view entities presented the capabilities the tool already has but also the ones still missing. However, no findings indicated the unfeasibility of such a modeling tool for CDS view entities.

Furthermore, the case study results support the business need for ER2CDS. One of the significant findings is that ER2CDS allows developers and business experts to create CDS view entities. Also, the SUS indicates high usability of the tool, making it accessible to an even broader audience. Another critical finding is users' preference regarding the use of ER2CDS and the textual syntax of CDS. Especially for business experts, ER2CDS offers a valuable alternative for creating CDS view entities, again indicating the value of such a tool, which is also reflected in the open question concerning positive feedback from the survey. However, developers are also eager to use the tool as an alternative to the textual syntax of CDS. Regarding the feedback, the hybrid modeling of CDS and the higher level of abstraction are the predominant advantages of ER2CDS. The negative suggestions and suggestions for improvement should serve as the basis for further development and future work within the context of low-code business app development and, specifically, ER2CDS. However, not all suggestions are of relevance. "SAP Tool integration" is already supported using BAS, as soon as the extension is published within the VS Code marketplace. A documentation of the tool, addressing the suggestion "Add a beginner's guide or documentation", is currently available via `https://github.com/borkdominik/ER2CDS/wiki`. On the other hand "Automatic scrolling when adding and editing attributes" and "List the attributes in the visual editor to select them" are usability improvements that can be addressed straight forwardly. For the "Support for all language features of CDS" ER2CDS aims to fully supports CDS language features in upcoming versions. The most significant features ER2CDS lacks are discussed in Section 7.1.2 and shall be addressed first. Due to the multitude of CDS features, supporting all may not be feasible. Future work should evaluate the importance of any feature and consequently implement it in ER2CDS.

In the next chapter, we will conclude and summarize the results of this thesis, revisit the research questions, and give an outlook on future work.

# CHAPTER 8

# Conclusion

In this chapter, the thesis is finalized. First, a summary of the contributions is presented. Then, the research questions are revisited and answered within the context of this thesis. Finally, an outlook on future work is presented.

## 8.1 Summary

This thesis contributes to the model-driven engineering process for CDS view entities within the SAP context. On the one hand, a domain-specific language, the ER2CDS modeling language, for textual and graphical modeling acts as a basis for a model-driven creation of CDS view entities. The novel ER2CDS tool presented in this thesis is a modeling tool supporting the ER2CDS language. The hybrid approach, supported by ER2CDS, allows for textual and graphical modeling and the creation of CDS view entities within a modern editor, in this case, VS Code. Furthermore, the import feature of ER2CDS enables users to work with existing CDS view entities. All in all, ER2CDS provides a higher level of abstraction for CDS. This implicitly broadens the target audience for CDS development.

In the following, the research questions presented in Chapter 1 will be answered using the approaches presented by this thesis:

**RQ1: To what extent does a model-driven engineering process improve the development of Core Data Services?**
ER2CDS, as presented in this thesis, is a development tool supporting the model-driven engineering process for CDS. Furthermore, this thesis has shown that business experts especially prefer ER2CDS, therefore a model-driven engineering process, over standard textual CDS development. However, developers are also keen to explore the capabilities of such a tool.

**RQ2: What is an appropriate means to model Core Data Services?**
The novel ER2CDS domain-specific language supports the modeling of CDS. The textual and graphical concrete syntax enable modeling in two different ways. Moreover, a hybrid approach is supported within the ER2CDS tool. This thesis also proved the validity of this approach by implementing existing CDS view entities using the ER2CDS language.

**RQ3: Which extensions to the Language Server Protocol are necessary to support hybrid textual and graphical modeling for Core Data Services?**
This question can be answered using the extension to LSP, as presented in Chapter 5. Combined with the synchronization, also illustrated in Chapter 5, and the implementation of ER2CDS, as presented in Chapter 6, allows for hybrid modeling of CDS.

**RQ4: What are the advantages and disadvantages of the integration of external data sources to a web-based modeling tool?**
First, data sources are integrated within ER2CDS for value-helps. This improves the usability of the tool, as well as the efficiency of modeling CDS. Secondly, the external data sources are used for advanced validation against the existing data model, improving the models' quality. The effectiveness of this integration is reflected in the high SUS, as presented as a part of the case study in Chapter 7.

**RQ5: To what extent can a Core Data Service be generated from a model which is based on the novel Core Data Service modeling language?**
The ER2CDS tool developed within this thesis uses a template-driven approach to generate a CDS view entity from a given ER2CDS model. The effectiveness and validity of this approach are proven in the experiment, presented in Chapter 7. The experiment conducted in this thesis has shown that 15 of 20 existing CDS view entities can already be implemented using ER2CDS. Moreover, the problems are not related to an inappropriate approach but to the lack of full support for all CDS language features.

## 8.2 Outlook

This section will present an outlook on future work within the context of this thesis. First and foremost, ER2CDS should be extended to support all CDS language features. This includes the missing functionality, as shown in Chapter 7, as well as annotations, a mechanism to add metadata to the source code of any CDS view entity.

Another important step left open by this thesis for future work is publishing the VS Code extension in the VS Code marketplace. This implicitly integrates ER2CDS within the SAP environment since SAP BAS supports the installation of VS Code extensions published in the marketplace. Currently, the tool is available open-source via `https://github.com/borkdominik/ER2CDS/`.

Finally, integrating a beginner's guide could improve CDS modeling and make it more accessible. In a more advanced extension, a modeling guide or modeling assistant using artificial intelligence [Bor21, GBE+23, MCK+20, SOFS23] would also increase users' productivity. All of these would improve the model-driven engineering process for CDS and, furthermore, the development of business applications [BCC22].

# List of Figures

# List of Tables

# Bibliography

[B+96] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[BCC22] Alessio Bucaioni, Antonio Cicchetti, and Federico Ciccozzi. Modelling in low-code development: a multi-vocal systematic review. *Softw. Syst. Model.*, 21(5):1959–1981, 2022.

[BCD10] Jeannette Bennett, Kendra Cooper, and Lirong Dai. Aspect-oriented model-driven skeleton code generation: A graph-based transformation approach. *Science of Computer Programming*, 75(8):689–725, 2010. Designing high quality system/software architectures.

[BCW12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-driven software engineering in practice.* Morgan & Claypool Publishers, 2012.

[BE03] Sikha Bagui and Richard Earp. *Database design using entity-relationship diagrams.* Auerbach Publications, 2003.

[BG01] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. *Automated Software Engineering*, pages 273– 280, 12 2001.

[BKM08] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[BL23] Dominik Bork and Philip Langer. Language server protocol: An introduction to the protocol, its use, and adoption for web modeling tools. *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.*, 18:9:1–16, 2023.

[BLO23] Dominik Bork, Philip Langer, and Tobias Ortmayr. A vision for flexible glsp-based web modeling tools. In João Paulo A. Almeida, Monika Kaczmarek-Heß, Agnes Koschmider, and Henderik A. Proper, editors, *The Practice of Enterprise Modeling - 16th IFIP Working Conference, PoEM 2023, Vienna, Austria, November 28 - December 1, 2023, Proceedings*, volume 497 of *Lecture Notes in Business Information Processing*, pages 109–124. Springer, 2023.

[BLRB24] Chiara Braghin, Mario Lilli, Elvinia Riccobene, and Marian Baba. Kant: A domain-specific language for modeling security protocols. In Francisco José Domínguez Mayo, Luís Ferreira Pires, and Edwin Seidewitz, editors, *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering, MODELSWARD 2024, Rome, Italy, February 21-23, 2024*, pages 62–73. SCITEPRESS, 2024.

[Bor21] Dominik Bork. Conceptual modeling and artificial intelligence: Challenges and opportunities for enterprise engineering - keynote presentation at the 11th enterprise engineering working conference (EEWC 2021). In David Aveiro, Henderik A. Proper, Sérgio Guerreiro, and Marné de Vries, editors, *Advances in Enterprise Engineering XV - 11th Enterprise Engineering Working Conference, EEWC 2021, Revised Selected Papers*, volume 441 of *Lecture Notes in Business Information Processing*, pages 3–9. Springer, 2021.

[CH$^+$03] Krzysztof Czarnecki, Simon Helsen, et al. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, volume 45, pages 1–17. USA, 2003.

[Che76a] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[Che76b] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, mar 1976.

[CTVW19] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. Blended modelling - what, why and how. In Loli Burgueño, Alexander Pretschner, Sebastian Voss, Michel Chaudron, Jörg Kienzle, Markus Völter, Sébastien Gérard, Mansooreh Zahedi, Erwan Bousse, Arend Rensink, Fiona Polack, Gregor Engels, and Gerti Kappel, editors, *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15-20, 2019*, pages 425–430. IEEE, 2019.

[DCLB22] Giuliano De Carlo, Philip Langer, and Dominik Bork. Advanced visualization and interaction in glsp-based web modeling: realizing semantic zoom and off-screen elements. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, MODELS '22, page 221–231, New York, NY, USA, 2022. Association for Computing Machinery.

[DLP$^+$23] Istvan David, Malvina Latifaj, Jakob Pietron, Weixing Zhang, Federico Ciccozzi, Ivano Malavolta, Alexander Raschke, Jan-Philipp Steghöfer, and Regina Hebig. Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. *Softw. Syst. Model.*, 22(1):415–447, 2023.

[EN16] Ramez Elmasri and Shamkant B Navathe. *Fundamentals of Database Systems*. Pearson, 2016.

[FCP+12] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. Sap hana database: data management for modern business applications. *SIGMOD Rec.*, 40(4):45–51, jan 2012.

[FML+12] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The sap hana database–an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[Foua] Eclipse Foundation. Documentation. https://eclipse.dev/glsp/documentation/. Accessed: 2024-6-15.

[Foub] Eclipse Foundation. The eclipse graphical language server platform (glsp). https://eclipse.dev/glsp/. Accessed: 2024-5-5.

[Fouc] Eclipse Foundation. Introduction. https://sprotty.org/docs/introduction/. Accessed: 2024-6-8.

[Foud] Eclipse Foundation. Overview. https://sprotty.org/docs/overview/. Accessed: 2024-6-8.

[fS] Xiao fei Song. An introduction to cds graphical modeler for sap business application studio. https://community.sap.com/t5/technology-blogs-by-sap/an-introduction-to-cds-graphical-modeler-for-sap-business-application/ba-p/13504103. Accessed: 2024-6-15.

[GB21] Philipp-Lorenz Glaser and Dominik Bork. The biger tool - hybrid textual and graphical modeling of entity relationships in vs code. In *2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 337–340, 2021.

[GBE+23] Antonio Garmendia, Dominik Bork, Martin Eisenberg, Thiago do Nascimento Ferreira, Marouane Kessentini, and Manuel Wimmer. Leveraging artificial intelligence for model-based software analysis and design. In José Raúl Romero, Inmaculada Medina-Bulo, and Francisco Chicano, editors, *Optimising the Software Development Process with Artificial Intelligence*, Natural Computing Series, pages 93–117. Springer, 2023.

[GGC22] Joan Giner-Miguelez, Abel Gómez, and Jordi Cabot. Describeml: a tool for describing machine learning datasets. In Thomas Kühn and Vasco Sousa, editors, *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS 2022, Montreal, Quebec, Canada, October 23-28, 2022*, pages 22–26. ACM, 2022.

[GHHB22] Philipp-Lorenz Glaser, Georg Hammerschmied, Vladyslav Hnatiuk, and Dominik Bork. The biger modeling tool. In Sebastian Link, Iris Reinhartz-Berger, Jelena Zdravkovic, Dominik Bork, and Srinath Srinivasa, editors, *Proceedings of the ER Forum and PhD Symposium 2022 co-located with 41st International Conference on Conceptual Modeling (ER 2022), Virtual Event, Hyderabad, India, October 17, 2022*, volume 3211 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[HKGV10] Zef Hemel, Lennart CL Kats, Danny M Groenewegen, and Eelco Visser. Code generation by model transformation: a case study in transformation modularity. *Software & Systems Modeling*, 9:375–402, 2010.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[HRW11] John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 633–642, 2011.

[JKCP15] Ankur Joshi, Saket Kale, Satish Chandel, and D Kumar Pal. Likert scale: Explored and explained. *British journal of applied science & technology*, 7(4):396–403, 2015.

[Kel] Horst Keller. Cds - one concept, two flavors. `https://community.sap.com/t5/technology-blogs-by-sap/cds-one-concept-two-flavors/ba-p/13168795`. Accessed: 2024-6-14.

[KJ+13] Max Kuhn, Kjell Johnson, et al. *Applied predictive modeling*, volume 26. Springer, 2013.

[KWB03] Anneke G Kleppe, Jos B Warmer, and Wim Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[Lan] Philip Langer. Diagram editors with glsp: Why flexibility is key. `https://www.youtube.com/watch?v=mSTXgUZCBVE`. Accessed: 2024-6-15.

[MB23] H. Metin and D. Bork. On developing and operating glsp-based web modeling tools: Lessons learned from biguml. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 129–139, Los Alamitos, CA, USA, oct 2023. IEEE Computer Society.

[MBWM23] Judith Michael, Dominik Bork, Manuel Wimmer, and Heinrich C. Mayr. Quo vadis modeling? *Software and Systems Modeling*, Oct 2023.

[MCK+20] Gunter Mussbacher, Benoît Combemale, Jörg Kienzle, Silvia Abrahão, Hyacinth Ali, Nelly Bencomo, Márton Búr, Loli Burgueño, Gregor Engels, Pierre Jeanjean, Jean-Marc Jézéquel, Thomas Kühn, Sébastien Mosser, Houari A. Sahraoui, Eugene Syriani, Dániel Varró, and Martin Weyssow. Opportunities in intelligent modeling assistance. *Softw. Syst. Model.*, 19(5):1045–1053, 2020.

[Mica] Microsoft. Commands. `https://code.visualstudio.com/api/extension-guides/command`. Accessed: 2024-6-8.

[Micb] Microsoft. Extension api. `https://code.visualstudio.com/api`. Accessed: 2024-6-8.

[Micc] Microsoft. Getting started. `https://code.visualstudio.com/docs`. Accessed: 2024-6-8.

[Micd] Microsoft. Language extensions overview. `https://code.visualstudio.com/api/language-extensions/overview`. Accessed: 2024-6-8.

[Mice] Microsoft. Language server extension guide. `https://code.visualstudio.com/api/language-extensions/language-server-extension-guide`. Accessed: 2024-6-8.

[Micf] Microsoft. Language server protocol specification. `https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/`. Accessed: 2024-5-5.

[Micg] Microsoft. Webview api. `https://code.visualstudio.com/api/extension-guides/webview`. Accessed: 2024-6-7.

[Mich] Microsoft. What is the language server protocol? `https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/`. Accessed: 2024-6-7.

[NN22] Gunasinghe Nadeeshaan and Marcus Nipuna. *Language Server Protocol and Implementation.* Apress, 2022.

[O'R15] Gerard O'Regan. *Pillars of computing.* Springer, 2015.

[Pet22] Jette Petzold. *A Textual Domain Specific Language for System-Theoretic Process Analysis.* PhD thesis, Kiel University, 2022.

[PKvH23] Jette Petzold, Jana Kreiß, and Reinhard von Hanxleden. PASTA: pragmatic automated system-theoretic process analysis. In *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Network, DSN 2023, Porto, Portugal, June 27-30, 2023*, pages 559–567. IEEE, 2023.

[PLV23] Gregory Popov, Joan Lu, and Vladimir Vishnyakov. Toward extensible low-code development platforms. In *International Conference on Innovation of Emerging Information and Communication Technology*, pages 487–497. Springer, 2023.

[RIWC18] Roberto Rodríguez-Echeverría, Javier Luis Cánovas Izquierdo, Manuel Wimmer, and Jordi Cabot. Towards a language server protocol infrastructure for graphical modeling. In Andrzej Wasowski, Richard F. Paige, and Øystein Haugen, editors, *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018*, pages 370–380. ACM, 2018.

[S$^+$06] Douglas C Schmidt et al. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2):25, 2006.

[Sar23] Siar Sarferaz. *Virtuelles Datenmodell*, pages 349–360. Springer Fachmedien Wiesbaden, Wiesbaden, 2023.

[Sch16] Olaf Schulz. *Der SAP-Grundkurs für Einsteiger und Anwender*. Rheinwerk Verlag, 2016.

[SEa] SAP SE. Abap - core data services (abap cds). `https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm?file=abencds.htm`. Accessed: 2024-5-5.

[SEb] SAP SE. Abap - core data services (abap cds). `https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds.htm`. Accessed: 2024-6-14.

[SEc] SAP SE. Abap core data services | s/4hana - best practice guide. `https://www.sap.com/documents/2019/01/0e6d5904-367d-0010-87a3-c30de2ffd8ff.html`. Accessed: 2024-6-14.

[SEd] SAP SE. Cds ddl - cds view entity, association. `https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_simple_association_v2.htm`. Accessed: 2024-6-14.

[SEe] SAP SE. Cds ddl - cds view entity, association to parent. `https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_to_parent_assoc_v2.htm`. Accessed: 2024-6-14.

[SEf] SAP SE. Cds ddl - cds view entity, associations and joins. `https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_assoc_join_v2.htm`. Accessed: 2024-6-14.

[SEg] SAP SE. Cds ddl - cds view entity, composition. https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_composition_v2.htm. Accessed: 2024-6-14.

[SEh] SAP SE. Cds ddl - cds view entity, select. https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_select_statement_v2.htm. Accessed: 2024-6-14.

[SEi] SAP SE. Cds ddl - cds view entity, select, associations. https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_association_v2.htm. Accessed: 2024-6-14.

[SEj] SAP SE. Cds ddl - cds view entity, select, clauses. https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_select_clauses_v2.htm. Accessed: 2024-6-14.

[SEk] SAP SE. Cds ddl - cds view entity, select, data_source. https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_joined_data_source_v2.htm. Accessed: 2024-6-14.

[SEl] SAP SE. Cds ddl - cds view entity, select, select_list. https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_select_list_entry_v2.htm. Accessed: 2024-6-14.

[SEm] SAP SE. Cds ddl - define view entity. https://help.sap.com/doc/abapdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_define_view_entity.htm. Accessed: 2024-6-14.

[SEn] SAP SE. Definition language (cdl). https://cap.cloud.sap/docs/cds/cdl. Accessed: 2024-6-15.

[SEo] SAP SE. Developing apps with sap fiori elements. https://sapui5.hana.ondemand.com/sdk/#/topic/03265b0408e2432c9571d6b3feb6b1fd. Accessed: 2024-5-5.

[SEp] SAP SE. Introduction to abap core data services (cds). https://www.sap.com/documents/2022/01/96489f20-157e-0010-bca6-c68f7e60039b.html. Accessed: 2024-6-14.

[SEq] SAP SE. Sap announces q4 and fy 2023. https://news.sap.com/2024/01/sap-announces-q4-and-fy-2023-results/. Accessed: 2024-6-9.

[SEr] SAP SE. Sap s/4hana. https://help.sap.com/docs/SAP_S4HANA_ON-PREMISE. Accessed: 2024-6-9.

[SEs] SAP SE. Sap s/4hana 2023. https://help.sap.com/doc/e2048712f0ab45e791e6d15ba5e20c68/2023/en-US/FSD_OP2023_latest.pdf. Accessed: 2024-6-9.

[SEt] SAP SE. Vdm layers and view types. https://help.sap.com/docs/SAP_S4HANA_ON-PREMISE/ee6ff9b281d8448f96b4fe6c89f2bdc8/0a875bc7a005465aad92c08becc11776.html. Accessed: 2024-6-14.

[SOFS23] Nikolay Shilov, Walaa Othman, Michael Fellmann, and Kurt Sandkuhl. Machine learning for enterprise modeling assistance: an investigation of the potential and proof of concept. *Softw. Syst. Model.*, 22(2):619–646, 2023.

[Sta] Statista. Top ERP software market share by company 2023. https://www.statista.com/statistics/249637/erp-software-market-share-by-company/. Accessed: 2024-5-5.

[Typa] TypeFox. Document lifecycle. https://langium.org/docs/reference/document-lifecycle/. Accessed: 2024-6-8.

[Typb] TypeFox. Features. https://langium.org/docs/features/. Accessed: 2024-6-8.

[Typc] TypeFox. Grammar language. https://langium.org/docs/reference/grammar-language/. Accessed: 2024-6-8.

[Typd] TypeFox. What is langium? https://langium.org/docs/introduction/. Accessed: 2024-6-8.

[WHR14] Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79–85, 2014.

# APPENDIX A

# Implementation

## A.1 CDS view entity generation

```typescript
import { URI, expandToString } from 'langium';
import { createER2CDSServices } from '../er2cds-module.js';
import { Attribute, ER2CDS, Entity, Relationship, RelationshipJoinClause } from '../generated/ast.js';
import { ER2CDSFileSystem } from '../er2cds-file-system-provider.js';
import { MessageType } from 'vscode-languageserver-protocol';
import { connection } from '../server.js';

export async function generateCDS(fileName: string | undefined): Promise<void> {
    if (!fileName) {
        connection.sendNotification('window/showMessage', {
            type: MessageType.Error,
            message: 'No␣valid␣ER2CDS␣file␣opened.'
        });
        return;
    }

    const services = createER2CDSServices(ER2CDSFileSystem).ER2CDS;

    const fileUri = URI.parse(fileName);
    const document = services.shared.workspace.LangiumDocumentFactory.create(fileUri);
    await services.shared.workspace.DocumentBuilder.build([
       document], { validation: true })

    const parseResult = document.parseResult;
    if ((parseResult.parserErrors && parseResult.parserErrors.
       length > 0) || (parseResult.lexerErrors && parseResult.
       lexerErrors.length > 0)) {
        connection.sendNotification('window/showMessage', {
            type: MessageType.Error,
            message: 'Model␣contains␣errors.␣CDS␣cannot␣be␣
               generated.'
        });
        return;
    }

    const diagnostics = await services.validation.
       DocumentValidator.validateDocument(document);
    if (diagnostics.some(d => d.severity === 1)) {
        connection.sendNotification('window/showMessage', {
            type: MessageType.Error,
            message: 'Model␣contains␣errors.␣CDS␣cannot␣be␣
               generated.'
        });
        return;
    }

    const sourceCode = generateSourceCode(parseResult.value as
       ER2CDS);
    if (!sourceCode) {
        connection.sendNotification('window/showMessage', {
            type: MessageType.Error,
            message: 'CDS-Generation␣failed.'
        });
        return;
    }

    const fileNameWithExtension = fileUri.fsPath.substring(
       fileUri.fsPath.lastIndexOf('/') + 1, fileUri.fsPath.
       length);
    const fileNameWithoutExtension = fileNameWithExtension.
       substring(0, fileNameWithExtension.lastIndexOf('.'));

    const generatedFileName = fileNameWithoutExtension + '-
```

```
       generated.abapcds';
    const generatedFilePath = fileUri.fsPath.substring(0,
       fileUri.fsPath.lastIndexOf('/')) + '/' +
       generatedFileName;

    ER2CDSFileSystem.fileSystemProvider().writeFile(URI.parse(
       generatedFilePath), sourceCode);

    return Promise.resolve();
}

function generateSourceCode(model: ER2CDS): string | undefined
   {
    model.relationships.sort((r1, r2) => {
        if (r1.joinOrder && r2.joinOrder)
            return r1.joinOrder - r2.joinOrder;

        if (r1.joinOrder)
            return -1;

        if (r2.joinOrder)
            return 1;

        return 0;
    });

    return expandToString`
        ${generateHeaderAnnotations(model)}
        ${generateHeader(model)}
            ${generateFromClause(model)}
            ${generateJoins(model)}
            ${generateAssociations(model)}
            ${generateAssociationsToParent(model)}
            ${generateCompositions(model)}
        {
            ${generateKeyAttributes(model)}${model.entities.
               find(e => e.attributes.find(a => a.type === 'key
               ')) && model.entities.find(e => e.attributes.
               find(a => a.type !== 'key')) ? ',' : ''}
            ${generateAttributes(model)}${model.entities.find(e
                => e.attributes.find(a => a.type !== 'key')) &&
                model.relationships.find(r => r.type === '
               association' || r.type === 'association-to-
               parent' || r.type === 'composition') ? ',' : ''}
            ${generateAssociationAttributes(model)}
        }
    `;
}

function generateHeaderAnnotations(model: ER2CDS): string |
   undefined {
    return expandToString`
        @AccessControl.authorizationCheck: #CHECK
        @Metadata.ignorePropagatedAnnotations: true
        @EndUserText.label: 'Generated␣by␣ER2CDS'
    `;
}

function generateHeader(model: ER2CDS): string | undefined {
    return expandToString`
        define view entity ${model.name} as select
    `;
}

function generateFromClause(model: ER2CDS): string | undefined
   {
    if (model.relationships && model.relationships.length > 0)
       {
        return expandToString`
            from ${model.relationships[0].source?.target.ref?.
               name} ${model.relationships[0].source?.target.
               ref?.alias ? `as ${model.relationships[0].source
               ?.target.ref?.alias}` : undefined}
        `;
    } else {
        return expandToString`
            from ${model.entities[0].name} ${model.entities[0].
               alias ? `as ${model.entities[0].alias}` :
               undefined}
        `;
    }
}

function generateJoins(model: ER2CDS): string | undefined {
    if (model.relationships) {
        return model.relationships.filter(r => !r.type).map(r
           => {
            let join;

            if (r.source?.cardinality === '1' && r.target?.
               cardinality === '1') {
                join = generateInnerJoin(model, r);

            } else if (r.source?.cardinality === '1' && r.
               target?.cardinality === '0..N') {
                join = generateLeftJoin(model, r);

            } else if (r.source?.cardinality === '0..N' && r.
               target?.cardinality === '1') {
                join = generateRightJoin(model, r);

            } else if (r.source?.cardinality === '0..N' && r.
               target?.cardinality === '0..N') {
                join = generateLeftJoin(model, r);

            } else {
                join = generateInnerJoin(model, r);

            }

            return join;
        }).filter(Boolean).join('\n');
    }

    return undefined;
}

function generateInnerJoin(model: ER2CDS, relationship:
   Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            inner join ${relationship.target?.target.ref?.name}
               ${relationship.target?.target.ref?.alias ? ` as
               ${relationship.target?.target.ref?.alias}` :
               undefined} on ${generateJoinClause(relationship,
                relationship.joinClauses)}
        `;
    }

    return undefined;
}

function generateLeftJoin(model: ER2CDS, relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            left outer join ${relationship.target?.target.ref?.name}${relationship.target?.target.ref?.alias ? ` as ${relationship.target?.target.ref?.alias}` : undefined} on ${generateJoinClause(relationship, relationship.joinClauses)}
        `;
    }

    return undefined;
}

function generateRightJoin(model: ER2CDS, relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            right outer join ${relationship.target?.target.ref?.name}${relationship.target?.target.ref?.alias ? ` as ${relationship.target?.target.ref?.alias}` : undefined} on ${generateJoinClause(relationship, relationship.joinClauses)}
        `;
    }

    return undefined;
}

function generateJoinClause(relationship: Relationship, joinClauses: RelationshipJoinClause[]): string | undefined {
    let joinClause;

    joinClause = joinClauses.map(jc => {
        return expandToString`
            ${relationship.source?.target.ref?.alias ? relationship.source?.target.ref?.alias : relationship.source?.target.ref?.name}.${jc.firstAttribute.ref?.name} = ${relationship.
```

```
               target?.target.ref?.alias ? relationship.target
               ?.target.ref?.alias : relationship.target?.
               target.ref?.name}.${jc.secondAttribute.ref?.name
               }
        `
    }).filter(Boolean).join('␣and␣');

    return joinClause;
}

function generateAssociations(model: ER2CDS): string |
   undefined {
    if (model.relationships) {
        return model.relationships.filter(r => r.type === '
           association').map(r => {
            let association;

            if (r.source?.cardinality === '1' && r.target?.
               cardinality === '1') {
                association = generateOneOneAssociation(model,
                   r);

            } else if (r.source?.cardinality === '1' && r.
               target?.cardinality === '0..N') {
                association = generateOneManyAssociation(model,
                    r);

            } else if (r.source?.cardinality === '0..N' && r.
               target?.cardinality === '1') {
                association = generateZeroOneAssociation(model,
                    r);

            } else if (r.source?.cardinality === '0..N' && r.
               target?.cardinality === '0..N') {
                association = generateZeroManyAssociation(model
                   , r);

            } else {
                association = generateZeroOneAssociation(model,
                    r);

            }

            return association;
        }).filter(Boolean).join('\n');
    }

    return undefined;
}

function generateOneOneAssociation(model: ER2CDS, relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            association[1..1] to ${relationship.target?.target.ref?.name}${relationship.target?.target.ref?.alias ? ` as ${relationship.target?.target.ref?.alias}` : undefined} on ${generateAssociationClause(relationship, relationship.joinClauses)}
        `;
    }

    return undefined;
}

function generateOneManyAssociation(model: ER2CDS, relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            association[1..*] to ${relationship.target?.target.ref?.name}${relationship.target?.target.ref?.alias ? ` as ${relationship.target?.target.ref?.alias}` : undefined} on ${generateAssociationClause(relationship, relationship.joinClauses)}
        `;
    }

    return undefined;
}

function generateZeroOneAssociation(model: ER2CDS, relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
```

```
            association[0..1] to ${relationship.target?.target.
               ref?.name}${relationship.target?.target.ref?.
               alias ? ` as ${relationship.target?.target.ref?.
               alias}` : undefined} on ${
               generateAssociationClause(relationship,
               relationship.joinClauses)}
        `;
    }

    return undefined;
}

function generateZeroManyAssociation(model: ER2CDS,
   relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            association[0..*] to ${relationship.target?.target.
               ref?.name}${relationship.target?.target.ref?.
               alias ? ` as ${relationship.target?.target.ref?.
               alias}` : undefined} on ${
               generateAssociationClause(relationship,
               relationship.joinClauses)}
        `;
    }

    return undefined;
}

function generateAssociationClause(relationship: Relationship,
   joinClauses: RelationshipJoinClause[]): string | undefined {
    let associationClause;

    associationClause = joinClauses.map(jc => {
        return expandToString`
            $projection.${jc.firstAttribute.ref?.alias ? jc.
               firstAttribute.ref?.alias : jc.firstAttribute.
               ref?.name} = ${relationship.target?.target.ref?.
               alias ? relationship.target?.target.ref?.alias :
                relationship.target?.target.ref?.name}.${jc.
               secondAttribute.ref?.name}
        `
    }).filter(Boolean).join('␣and␣');

    return associationClause;
}

function generateAssociationsToParent(model: ER2CDS): string | undefined {
    if (model.relationships) {
        return model.relationships.filter(r => r.type === ’association-to-parent’).map(r => generateAssociationToParent(model, r)).filter(Boolean).join(’\n’);
    }

    return undefined;
}

function generateAssociationToParent(model: ER2CDS, relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            association to parent ${relationship.target?.target.ref?.name}${relationship.target?.target.ref?.alias ? ` as ${relationship.target?.target.ref?.alias}` : undefined} on ${generateAssociationClause(relationship, relationship.joinClauses)}
        `;
    }

    return undefined;
}

function generateCompositions(model: ER2CDS): string | undefined {
    if (model.relationships) {
        return model.relationships.filter(r => r.type === ’composition’).map(r => {
            let association;

            if (r.source?.cardinality === ’1’ && r.target?.cardinality === ’1’) {
                association = generateOneOneComposition(model, r);

            } else if (r.source?.cardinality === '1' && r.target?.cardinality === '0..N') {
                association = generateOneManyComposition(model, r);

            } else if (r.source?.cardinality === '0..N' && r.target?.cardinality === '1') {
                association = generateZeroOneComposition(model, r);

            } else if (r.source?.cardinality === '0..N' && r.target?.cardinality === '0..N') {
                association = generateZeroManyComposition(model, r);

            } else {
                association = generateZeroManyComposition(model, r);

            }

            return association;
        }).filter(Boolean).join('\n');
    }

    return undefined;
}

function generateOneOneComposition(model: ER2CDS, relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            composition[1..1] of ${relationship.target?.target.ref?.name}${relationship.target?.target.ref?.alias ? ` as ${relationship.target?.target.ref?.alias}` : undefined}
        `;
    }

    return undefined;
}

function generateOneManyComposition(model: ER2CDS, relationship
   : Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            composition[0..*] of ${relationship.target?.target.
               ref?.name}${relationship.target?.target.ref?.
               alias ? ` as ${relationship.target?.target.ref?.
               alias}` : undefined}
        `;
    }

    return undefined;
}

function generateZeroOneComposition(model: ER2CDS, relationship
   : Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            composition[0..1] of ${relationship.target?.target.
               ref?.name}${relationship.target?.target.ref?.
               alias ? ` as ${relationship.target?.target.ref?.
               alias}` : undefined}
        `;
    }

    return undefined;
}

function generateZeroManyComposition(model: ER2CDS,
   relationship: Relationship): string | undefined {
    if (model.entities && model.entities.length > 0) {
        return expandToString`
            composition[0..*] of ${relationship.target?.target.
               ref?.name}${relationship.target?.target.ref?.
               alias ? ` as ${relationship.target?.target.ref?.
               alias}` : undefined}
        `;
    }

    return undefined;
}

function generateKeyAttributes(model: ER2CDS): string |
   undefined {
    if (model.entities) {
        let attributes: string[] = [];

        model.entities.forEach(e => {
            const keyFields = e.attributes.filter(a => a.type === ’key’);

            if (keyFields.length > 0)
                attributes.push(keyFields.map(a => generateAttribute(e, a)).filter(Boolean).join(’,\n’));
        });

        return attributes.filter(Boolean).join(’,\n’);
    }

    return undefined;
}

function generateAttributes(model: ER2CDS): string | undefined {
    if (model.entities) {
        let attributes: string[] = [];

        model.entities.forEach(e => {
            const nonKeyFields = e.attributes.filter(a => a.type !== ’key’);

            if (nonKeyFields.length > 0) {
                attributes.push(nonKeyFields.map(a => generateAttribute(e, a)).filter(Boolean).join(’,\n’));
            }
        });

        return attributes.filter(Boolean).join(’,\n’);
    }

    return undefined;
}

function generateAttribute(entity: Entity, attribute: Attribute): string | undefined {
    if (attribute.type !== 'no-out') {
        return expandToString`
            ${attribute.type === 'key' ? 'key' : ''} ${entity.alias ? entity.alias : entity.name}.${attribute.name} ${generateAttributeLabel(entity, attribute)}
        `;
    }

    return undefined;
}

function generateAttributeLabel(entity: Entity, attribute: Attribute): string | undefined {
    if (attribute.alias) {
        return expandToString`
            as ${attribute.alias}
        `;
    }

    return undefined;
}

function generateAssociationAttributes(model: ER2CDS): string | undefined {
    if (model.relationships) {
        let associationAttributes: string[] = [];

        model.relationships.filter(r => r.type === 'association' || r.type === 'association-to-parent' || r.type === 'composition').forEach(r => {
            associationAttributes.push(expandToString`${r.target?.target.ref?.alias ? r.target?.target.ref?.alias : r.target?.target.ref?.name}`);
        });

        return associationAttributes.filter(Boolean).join(',\n');
    }

    return undefined;
}
```

Listing A.1: Generation of a CDS view entity.

## A.2 Import CDS view entity

```
import { Agent } from 'https';
import { ER2CDSGlobal } from '../er2cds-module.js';
import fetch from 'node-fetch';
import { connection } from '../server.js';
import { MessageType } from 'vscode-languageserver-protocol';
import { Attribute, ER2CDS, Entity, Relationship, RelationshipJoinClause } from '../generated/ast.js';
import { serialize } from '../serializer/serializer.js';
import { URI } from 'langium';
import { ER2CDSFileSystem } from '../er2cds-file-system-provider.js';

export async function importCds(entityName: string, fileName: string): Promise<void> {
    const agent = new Agent({ rejectUnauthorized: false });

    entityName = entityName.toUpperCase();

    const urlFromClause = encodeURI(ER2CDSGlobal.sapUrl + "sap/opu/odata/sap/ZER2CDS/ImportFromClause?$filter=EntityName␣eq␣'" + entityName + "'&$format=json&$top=9999&sap-client=" + ER2CDSGlobal.sapClient);
    const urlSelectList = encodeURI(ER2CDSGlobal.sapUrl + "sap/opu/odata/sap/ZER2CDS/ImportSelectList?$filter=EntityName␣eq␣'" + entityName + "'&$format=json&$top=9999&sap-client=" + ER2CDSGlobal.sapClient);
    const urlCondition = encodeURI(ER2CDSGlobal.sapUrl + "sap/opu/odata/sap/ZER2CDS/ImportCondition?$filter=EntityName␣eq'" + entityName + "'&$format=json&$top=9999&sap-client=" + ER2CDSGlobal.sapClient);
    const urlAssocDef = encodeURI(ER2CDSGlobal.sapUrl + "sap/opu/odata/sap/ZER2CDS/ImportAssocDef?$filter=EntityName␣eq'" + entityName + "'&$format=json&$top=9999&sap-client=" + ER2CDSGlobal.sapClient);

    const fromClause: IImportFromClause[] = await fetch(
        urlFromClause,
        {
            agent: agent,
            method: 'GET',
            headers: {
                'Authorization': 'Basic␣' + btoa(ER2CDSGlobal.
                   sapUsername + ':' + ER2CDSGlobal.sapPassword
                   )
            }
        }
    ).then(
        (response: any) => response.json()
    ).then(
        (response: any) => {
            return response.d.results.map((r: any) => r as
               IImportFromClause);
        }
    ).catch(
        (error: any) => {
            connection.sendNotification('window/showMessage', {
                type: MessageType.Error,
                message: `CDS View Entity cannot be imported. $
                   {error}`
            });

            return Promise.resolve();
        }
    )

    const selectionList: IImportSelectionList[] = await fetch(
        urlSelectList,
        {
            agent: agent,
            method: 'GET',
            headers: {
                'Authorization': 'Basic␣' + btoa(ER2CDSGlobal.
                   sapUsername + ':' + ER2CDSGlobal.sapPassword
                   )
            }
        }
    ).then(
        (response: any) => response.json()
    ).then(
        (response: any) => {
            return response.d.results.map((r: any) => r as
               IImportSelectionList)
                .sort((r1: IImportSelectionList, r2:
                   IImportSelectionList) => r1.ElementPos - r2.
                   ElementPos);
        }
    ).catch(
        (error: any) => {
            connection.sendNotification(’window/showMessage’, {
                type: MessageType.Error,
                message: `CDS View Entity cannot be imported. $
                   {error}`
            });

            return Promise.resolve();
        }
    )

    const condition: IImportCondition[] = await fetch(
        urlCondition,
        {
            agent: agent,
            method: ’GET’,
            headers: {
                ’Authorization’: ’Basic␣’ + btoa(ER2CDSGlobal.
                   sapUsername + ’:’ + ER2CDSGlobal.sapPassword
                   )
            }
        }
    ).then(
        (response: any) => response.json()
    ).then(
        (response: any) => {
            return response.d.results.map((r: any) => r as
               IImportCondition)
                .sort((r1: IImportCondition, r2:
                   IImportCondition) => r1.Pos !== r2.Pos ? r1.
                   Pos - r2.Pos : r1.StackLine - r1.StackLine);
        }
    ).catch(
        (error: any) => {
            connection.sendNotification(’window/showMessage’, {
                type: MessageType.Error,
                message: `CDS View Entity cannot be imported. $
                   {error}`
            });

            return Promise.resolve();
        }
    )

    const assocDef: IImportAssocDef[] = await fetch(
        urlAssocDef,
        {
            agent: agent,
            method: ’GET’,
            headers: {
                ’Authorization’: ’Basic␣’ + btoa(ER2CDSGlobal.sapUsername + ’:’ + ER2CDSGlobal.sapPassword)
            }
        }
    ).then(
        (response: any) => response.json()
    ).then(
        (response: any) => {
            return response.d.results.map((r: any) => r as IImportAssocDef);
        }
    ).catch(
        (error: any) => {
            connection.sendNotification(’window/showMessage’, {
                type: MessageType.Error,
                message: `CDS View Entity cannot be imported. ${error}`
            });

            return Promise.resolve();
        }
    )

    if (!condition || condition.length <= 0) {
        connection.sendNotification(’window/showMessage’, {
            type: MessageType.Error,
            message: `CDS View Entity cannot be imported. CDS View Entity not found.`
        });

        return Promise.resolve();
    }

    const er2cds = convertToER2CDS(entityName, fromClause, selectionList, condition, assocDef);
    const source = serialize(er2cds);

    let cleanedEntityName = entityName.replace('/', '');
    cleanedEntityName = cleanedEntityName.replaceAll('/', '_');

    const fileUri = URI.parse(fileName);
    const generatedFileName = cleanedEntityName + '-imported.er2cds';
    const generatedFilePath = fileUri.fsPath.substring(0, fileUri.fsPath.lastIndexOf('/')) + '/' + generatedFileName;

    ER2CDSFileSystem.fileSystemProvider().writeFile(URI.parse(generatedFilePath), source);

    connection.sendNotification('window/showMessage', {
        type: MessageType.Info,
        message: `CDS View Entity imported successfully.`
    });
}

export function convertToER2CDS(entityName: string, fromClause: IImportFromClause[], selectionList: IImportSelectionList[], condition: IImportCondition[], assocDef: IImportAssocDef[]) : ER2CDS {
    let er2cds: ER2CDS = {
        $type: 'ER2CDS',
        name: entityName,
        entities: [],
        relationships: []
    };

    convertFromToER2CDS(er2cds, fromClause, selectionList, assocDef, condition);
    convertAssociationToER2CDS(er2cds, selectionList, assocDef, condition);

    er2cds.entities.map(e => e.attributes = e.attributes.filter(a => a.name !== 'MANDT' && a.name !== 'CLIENT'));

    return er2cds;
}

export function convertFromToER2CDS(er2cds: ER2CDS, fromClause:
    IImportFromClause[], selectionList: IImportSelectionList[],
    assocDef: IImportAssocDef[], condition: IImportCondition[])
   : void {
    convertFromToER2CDSEntity(er2cds, selectionList, assocDef,
       condition);
    convertFromToER2CDSRelationship(er2cds, fromClause,
       condition);
}

export function convertFromToER2CDSEntity(er2cds: ER2CDS,
   selectionList: IImportSelectionList[], assocDef:
   IImportAssocDef[], condition: IImportCondition[]): void {
    condition.filter(c => c.ConditionType === 'FROM' && c.
       ExprType === 'TABLE_DATASOURCE').forEach(c => {
        let entityName = c.BaseobjName;
        let entityAlias = c.BaseobjAlias;

        if (c.BaseobjAlias) {
            const assocAlias = condition.find(sc => sc.
               ConditionType === 'ASSOC_DEFINITION' && sc.
               ExprType === 'TABLE_DATASOURCE' && sc.
               BaseobjAlias === c.BaseobjAlias)
            const association = assocDef.find(a => a.AssocName
               === assocAlias?.AssocName);
            if (assocAlias && association) {
                entityName = assocAlias.BaseobjName;
                entityAlias = association.AssocNameRaw;
            }
        }

        let entity: Entity = {
            $type: 'Entity',
            $container: er2cds,
            name: entityName,
            alias: entityAlias,
            attributes: []
        };

        let attributes = selectionList.filter(s => s.
           BaseObjName === c.BaseobjName && s.ExprType === 'ATOMIC').map(s => {
            const ss = selectionList.find(ss => ss.ElementPos === s.ElementPos && ss.ExprType === 'STDSELECTLIST_ENTRY');
            return <Attribute>{
                $type: 'Attribute',
                $container: entity,
                name: s.BaseElementName,
                type: ss?.Keyflag ? 'key' : '',
                alias: ss?.FieldNameRaw,
                datatype: {
                    $type: 'DataType',
                    $container: null!,
                    type: s.Datatype ? s.Datatype : ss?.Datatype
                }
            }
        });
        entity.attributes = attributes;
        er2cds.entities.push(entity);
    });
}

export function convertFromToER2CDSRelationship(er2cds: ER2CDS, fromClause: IImportFromClause[], condition: IImportCondition[]): void {
    let relationship: Relationship = {
        $type: 'Relationship',
        $container: er2cds,
        name: null!,
        joinClauses: []
    };

    let joinClause: RelationshipJoinClause = {
        $type: 'RelationshipJoinClause',
        $container: null!,
        firstAttribute: null!,
        secondAttribute: null!
    };

    condition.filter(c => c.ConditionType === 'FROM').forEach(c => {
        if (c.ExprType === 'TABLE_DATASOURCE') {
            if (!relationship?.source) {
                relationship.source = {
                    $type: 'RelationshipEntity',
                    $container: relationship,
                    target: {
                        ref: er2cds.entities.find(e => e.name
                           === c.BaseobjName && e.alias === c.
                           BaseobjAlias)!,
                        $refText: c.BaseobjName
                    }
                }
            } else if (!relationship.target) {
                relationship.target = {
                    $type: 'RelationshipEntity',
                    $container: relationship,
                    target: {
                        ref: er2cds.entities.find(e => e.name
                           === c.BaseobjName && e.alias === c.
                           BaseobjAlias)!,
                        $refText: c.BaseobjName
                    }
                }
            }
        }

        if (c.ExprType === 'ATOMIC') {
            if (!joinClause.firstAttribute) {
                if ((relationship.source?.target.ref?.name !==
                   c.BaseobjName || relationship.source.target.
                   ref.alias !== c.BaseobjAlias) && (!
                   relationship.joinClauses || relationship.
                   joinClauses.length <= 0)) {
                    if (er2cds.entities.some(e => e.name === c.
                       BaseobjName && e.alias === c.
                       BaseobjAlias)) {
                        relationship.source = {
                            $type: 'RelationshipEntity',
                            $container: relationship,
                            target: {
                                ref: er2cds.entities.find(e =>
                                   e.name === c.BaseobjName &&
                                   e.alias === c.BaseobjAlias)
                                   !,
                                $refText: c.BaseobjName
                            }
                        }
                    }
                }

                joinClause.firstAttribute = {
                    ref: er2cds.entities.find(e => e.name === c
                       .BaseobjName && e.alias === c.
                       BaseobjAlias)?.attributes.find(a => a.
                       name === c.FieldName),
                    $refText: c.FieldName
                };
            } else if (!joinClause.secondAttribute) {
                if ((relationship.target?.target.ref?.name !==
                   c.BaseobjName || relationship.target.target.
                   ref.alias !== c.BaseobjAlias) && (!
                   relationship.joinClauses || relationship.
                   joinClauses.length <= 0)) {
                    if (er2cds.entities.some(e => e.name === c.
                       BaseobjName && e.alias === c.
                       BaseobjAlias)) {
                        relationship.target = {
                            $type: 'RelationshipEntity',
                            $container: relationship,
                            target: {
                                ref: er2cds.entities.find(e =>
                                   e.name === c.BaseobjName &&
                                   e.alias === c.BaseobjAlias)
                                   !,
                                $refText: c.BaseobjName
                            }
                        }
                    }
                }

                joinClause.secondAttribute = {
                    ref: er2cds.entities.find(e => e.name === c
                       .BaseobjName && e.alias === c.
                       BaseobjAlias)?.attributes.find(a => a.
                       name === c.FieldName),
                    $refText: c.FieldName
                };
            }
        }

        if (c.ExprType === 'COMPARISON') {
            if (joinClause.firstAttribute && joinClause.
               secondAttribute) {
                const sourceEntity = er2cds.entities.find(e =>
                   e.name === relationship.source?.target.ref?.
                   name && e.alias === relationship.source?.
                   target.ref?.alias);
                if (sourceEntity && !sourceEntity.attributes.
                   some(a => a.name === joinClause.
                   firstAttribute.$refText)) {
                    const attribute = condition.find(sc => sc.
                       ConditionType === 'FROM' && sc.ExprType
                       === 'ATOMIC' &&
                        sc.BaseobjName === sourceEntity.name &&
                            sc.FieldName === joinClause.
                           firstAttribute.$refText);

                    if (attribute) {
                        sourceEntity.attributes.push({
                            $type: 'Attribute',
                            $container: sourceEntity,
                            name: attribute?.FieldName,
                            type: 'no-out',
                            datatype: {
                                $type: 'DataType',
                                $container: null!,
                                type: attribute?.Datatype
                            }
                        });
                    } else {
                        joinClause.firstAttribute = undefined!;
                    }
                }

                const targetEntity = er2cds.entities.find(e =>
                   e.name === relationship.target?.target.ref?.
                   name && e.alias === relationship.target?.
                   target.ref?.alias);
                if (targetEntity && !targetEntity.attributes.
                   some(a => a.name === joinClause.
                   secondAttribute.$refText)) {
                    const attribute = condition.find(sc => sc.
                       ConditionType === 'FROM' && sc.ExprType
                       === 'ATOMIC' &&
                        sc.BaseobjName === targetEntity.name &&
                            sc.FieldName === joinClause.
                           secondAttribute.$refText);

                    if (attribute) {
                        targetEntity.attributes.push({
                            $type: 'Attribute',
                            $container: targetEntity,
                            name: attribute.FieldName,
                            type: 'no-out',
                            datatype: {
                                $type: 'DataType',
                                $container: null!,
                                type: attribute.Datatype
                            }
                        });
                    } else {
                        joinClause.secondAttribute = undefined
                           !;
                    }
                }

                if (joinClause.firstAttribute && joinClause.
                   secondAttribute) {
                    relationship.joinClauses.push(joinClause);
                }
            }

            joinClause = {
                $type: 'RelationshipJoinClause',
                $container: null!,
                firstAttribute: undefined!,
                secondAttribute: undefined!
            };
        }

        if (c.ExprType === 'JOIN_DATASOURCE') {
            if (relationship.source && relationship.target &&
                relationship.joinClauses && relationship.joinClauses.length > 0) {
                if (c.JoinOperation === 'INNER') {
                    relationship.source.cardinality = '1';
                    relationship.target.cardinality = '1';
                } else if (c.JoinOperation === 'LEFT') {
                    relationship.source.cardinality = '1';
                    relationship.target.cardinality = '0..N';
                } else if (c.JoinOperation === 'RIGHT') {
                    relationship.source.cardinality = '0..N';
                    relationship.target.cardinality = '1';
                }

                const fc = fromClause.find(f => f.BaseObjName === relationship.target?.target.$refText);
                if (fc && fc.BaseObjPos - 1 > 0) {
                    relationship.joinOrder = fc.BaseObjPos - 1;
                } else {
                    relationship.joinOrder = 1;
                }

                er2cds.relationships.push(relationship);

                const nextSourceEntity = relationship.source?.target.ref;

                relationship = {
                    $type: 'Relationship',
                    $container: er2cds,
                    name: null!,
                    joinClauses: []
                };

                if (nextSourceEntity) {
                    relationship.source = {
                        $type: 'RelationshipEntity',
                        $container: relationship,
                        target: {
                            ref: nextSourceEntity,
                            $refText: nextSourceEntity.name
                        }
                    }
                }
```

```
            }
        }
    });
}

export function convertAssociationToER2CDS(er2cds: ER2CDS, selectionList: IImportSelectionList[], assocDef: IImportAssocDef[], condition: IImportCondition[]): void {
    convertAssociationToER2CDSEntity(er2cds, assocDef, condition);
    convertAssociationToER2CDSRelationship(er2cds, selectionList, assocDef, condition);
}

export function convertAssociationToER2CDSEntity(er2cds: ER2CDS, assocDef: IImportAssocDef[], condition: IImportCondition[]): void {
    condition.filter(c => c.ConditionType === ’ASSOC_DEFINITION’ && c.ExprType === ’TABLE_DATASOURCE’).forEach(c => {
        if (!condition.some(sc => sc.ConditionType === ’FROM’ && sc.ExprType === ’TABLE_DATASOURCE’ && sc.BaseobjName === c.BaseobjName) &&
            !er2cds.entities.some(e => e.name === c.BaseobjName && e.alias === c.AssocName)) {
            const association = assocDef.find(a => a.AssocName === c.AssocName);

            er2cds.entities.push({
                $type: ’Entity’,
                $container: er2cds,
                name: c.BaseobjName,
                alias: association?.AssocNameRaw ? association.AssocNameRaw : c.BaseobjAlias,
                attributes: []
            });
        }
    });
}

export function convertAssociationToER2CDSRelationship(er2cds: ER2CDS, selectionList: IImportSelectionList[], assocDef: IImportAssocDef[], condition: IImportCondition[]): void {
    let relationship: Relationship = {
        $type: 'Relationship',
        $container: er2cds,
        name: null!,
        joinClauses: []
    };

    let joinClause: RelationshipJoinClause = {
        $type: 'RelationshipJoinClause',
        $container: null!,
        firstAttribute: null!,
        secondAttribute: null!
    };

    condition.filter(c => c.ConditionType === 'ASSOC_DEFINITION').forEach(c => {
        if (c.ExprType === 'ASSOC_ELEMENT') {
            if (!relationship?.source) {
                const association = assocDef.find(a => a.AssocName === c.AssocName);

                if (association) {
                    relationship.source = {
                        $type: 'RelationshipEntity',
                        $container: relationship,
                        target: {
                            ref: er2cds.entities.find(e => e.name === c.BaseobjName && e.alias === association.AssocNameRaw)!,
                            $refText: c.BaseobjName
                        }
                    }
                } else {
                    relationship.source = {
                        $type: 'RelationshipEntity',
                        $container: relationship,
                        target: {
                            ref: er2cds.entities.find(e => e.name === c.BaseobjName && e.alias === c.BaseobjAlias)!,
                            $refText: c.BaseobjName
                        }
                    }
                }
            } else if (!relationship.target) {
                const association = assocDef.find(a => a.AssocName === c.AssocName);

                if (association) {
                    relationship.target = {
                        $type: 'RelationshipEntity',
                        $container: relationship,
                        target: {
                            ref: er2cds.entities.find(e => e.name === c.BaseobjName && e.alias === association?.AssocNameRaw)!,
                            $refText: c.BaseobjName
                        }
                    }
                } else {
                    relationship.target = {
                        $type: 'RelationshipEntity',
                        $container: relationship,
                        target: {
                            ref: er2cds.entities.find(e => e.name === c.BaseobjName && e.alias === c.BaseobjAlias)!,
                            $refText: c.BaseobjName
                        }
                    }
                }
            }

            if (c.BaseobjName === er2cds.name) {
                let sourceEntity: Entity | undefined;
                let sourceAttribute: Attribute | undefined;

                let ss = selectionList.find(s => s.ElementAlias === c.FieldName);
                if (ss) {
                    const sss = selectionList.find(s => s.ElementPos === ss?.ElementPos && s.ExprType === 'ATOMIC');
                    const association = assocDef.find(a => a.AssocName === sss?.AssocName);

                    if (association) {
                        sourceEntity = er2cds.entities.find(e
                           => e.name === sss?.BaseObjName && e.
                           alias === association?.AssocNameRaw)
                           ;
                    } else {
                        sourceEntity = er2cds.entities.find(e
                           => e.name === sss?.BaseObjName && e.
                           alias === sss?.BaseObjAlias);
                    }

                    sourceAttribute = sourceEntity?.attributes.
                       find(a => a.name === sss?.
                       BaseElementName);
                } else {
                    ss = selectionList.find(s => (s.ExprType
                       === 'ATOMIC' || s.ExprType === '
                       ATOMIC_VIA_PATH') && s.BaseElementName
                       === c.FieldName);
                    const association = assocDef.find(a => a.
                       AssocTarget === ss?.BaseObjName);

                    if (association) {
                        sourceEntity = er2cds.entities.find(e
                           => e.name === ss?.BaseObjName && e.
                           alias === association?.AssocNameRaw)
                           ;
                    } else {
                        sourceEntity = er2cds.entities.find(e
                           => e.name === ss?.BaseObjName && e.
                           alias === ss?.BaseObjAlias);
                    }

                    sourceAttribute = sourceEntity?.attributes.
                       find(a => a.name === ss?.BaseElementName
                       );

                    if (!sourceAttribute) {
                        let attribute: IImportCondition |
                           undefined;
                        const association = assocDef.find(a =>
                           a.AssocNameRaw === sourceEntity?.
                           alias);

                        if (association) {
                            attribute = condition.find(sc => sc
                               .ConditionType === '
                               ASSOC_DEFINITION' && sc.ExprType
                                === 'ASSOC_ELEMENT' &&
                                sc.BaseobjName === sourceEntity
                                   ?.name && sc.AssocName ===
                                   association?.AssocName && sc
                                   .FieldName === c.FieldName);
                        } else {
                            attribute = condition.find(sc => sc
                               .ConditionType === 'FROM' && sc.
                               ExprType === 'ATOMIC' &&
                                sc.BaseobjName === sourceEntity
                                   ?.name && sc.BaseobjAlias
                                   === sourceEntity?.alias &&
                                   sc.FieldName === c.FieldName
                                   );
                        }

                        if (attribute) {
                            sourceEntity?.attributes.push({
                                $type: 'Attribute',
                                $container: sourceEntity,
                                name: attribute?.FieldName,
                                type: 'key',
                                datatype: {
                                    $type: 'DataType',
                                    $container: null!,
                                    type: attribute.Datatype
                                }
                            });
                        }

                        sourceAttribute = sourceEntity?.
                           attributes.find(a => a.name === ss?.
                           BaseElementName);
                    }
                }

                if (sourceEntity && sourceAttribute) {
                    if (relationship.source?.target.$refText === er2cds.name || relationship.source?.target.$refText === relationship.target?.target.$refText) {
                        relationship.source = {
                            $type: 'RelationshipEntity',
                            $container: relationship,
                            target: {
                                ref: sourceEntity,
                                $refText: sourceEntity.name
                            }
                        };
                    }

                    joinClause.firstAttribute = {
                        ref: sourceAttribute,
                        $refText: sourceAttribute.name
                    };
                }
            } else {
                let targetEntity: Entity | undefined;
                const association = assocDef.find(a => a.AssocName === c?.AssocName);

                if (association) {
                    targetEntity = er2cds.entities.find(e => e.name === c.BaseobjName && e.alias === association?.AssocNameRaw);
                } else {
                    targetEntity = er2cds.entities.find(e => e.name === c.BaseobjName && e.alias === c.BaseobjAlias);
                }

                const targetAttribute = targetEntity?.attributes.find(a => a.name === c.FieldName);

                if (!targetAttribute) {
                    let attribute: IImportCondition | undefined;
                    const association = assocDef.find(a => a.AssocNameRaw === targetEntity?.alias);

                    if (association) {
                        attribute = condition.find(sc => sc.ConditionType === 'ASSOC_DEFINITION' && sc.ExprType === 'ASSOC_ELEMENT' &&
                            sc.BaseobjName === targetEntity?.name && sc.AssocName === association?.AssocName && sc.FieldName === c.FieldName);
                    } else {
                        attribute = condition.find(sc => sc.ConditionType === 'FROM' && sc.ExprType === 'ATOMIC' &&
                            sc.BaseobjName === targetEntity?.name && sc.BaseobjAlias === targetEntity?.alias && sc.FieldName === c.FieldName);
                    }

                    if (attribute) {
                        targetEntity?.attributes.push({
                            $type: 'Attribute',
                            $container: targetEntity,
                            name: attribute?.FieldName,
                            type: 'no-out',
                            datatype: {
                                $type: 'DataType',
                                $container: null!,
                                type: attribute.Datatype
                            }
                        });
                    }
                }

                relationship.target = {
                    $type: 'RelationshipEntity',
                    $container: relationship,
                    target: {
                        ref: targetEntity,
                        $refText: c.BaseobjName
                    }
                };

                joinClause.secondAttribute = {
                    ref: targetAttribute,
                    $refText: c.FieldName
                };
            }
        }

        if (c.ExprType === 'COMPARISON' && joinClause.
           firstAttribute && joinClause.secondAttribute) {
            const sourceEntity = er2cds.entities.find(e => e.
               name === relationship.source?.target.ref?.name
               && e.alias === relationship.source?.target.ref?.
               alias);
            if (sourceEntity && !sourceEntity.attributes.some(a
                => a.name === joinClause.firstAttribute.
               $refText)) {
                const attribute = condition.find(sc => sc.
                   ConditionType === 'ASSOC_DEFINITION' && sc.
                   ExprType === 'ASSOC_ELEMENT' && sc.
                   BaseobjName === sourceEntity.name && sc.
                   FieldName === joinClause.firstAttribute.
                   $refText);

                if (attribute) {
                    sourceEntity.attributes.push({
                        $type: 'Attribute',
                        $container: sourceEntity,
                        name: joinClause.firstAttribute.
                           $refText,
                        type: 'no-out',
                        datatype: {
                            $type: 'DataType',
                            $container: null!,
                            type: attribute.Datatype
                        }
                    });
                }
            }

            const targetEntity = er2cds.entities.find(e => e.
               name === relationship.target?.target.ref?.name
               && e.alias === relationship.source?.target.ref?.
               alias);
            if (targetEntity && !targetEntity.attributes.some(a
                => a.name === joinClause.secondAttribute.
               $refText)) {
                const attribute = condition.find(sc => sc.
                   ConditionType === 'ASSOC_DEFINITION' && sc.
                   ExprType === 'ASSOC_ELEMENT' &&
                    sc.BaseobjName === targetEntity.name && sc.
                       FieldName === joinClause.secondAttribute
                       .$refText);

                if (attribute) {
                    targetEntity.attributes.push({
                        $type: 'Attribute',
                        $container: targetEntity,
                        name: attribute?.FieldName,
                        type: 'no-out',
                        datatype: {
                            $type: 'DataType',
                            $container: null!,
                            type: attribute.Datatype
                        }
                    });
                }
            }

            relationship.joinClauses.push(joinClause);
            joinClause = {
                $type: 'RelationshipJoinClause',
                $container: null!,
                firstAttribute: null!,
                secondAttribute: null!
            };
        }

        if (c.ExprType === 'ASSOCIATION' && relationship.source
            && relationship.target) {
            const association = assocDef.find(a => a.AssocName
               === c.AssocName);

            if (association) {
                if (association?.CardMin === 1) {
                    relationship.source.cardinality = '1';
                } else {
                    relationship.source.cardinality = '0..N';
                }

                if (association?.CardMax === 1) {
                    relationship.target.cardinality = '1';
                } else {
                    relationship.target.cardinality = '0..N';
                }

                relationship.type = 'association';
                er2cds.relationships.push(relationship);
            }

            relationship = {
                $type: 'Relationship',
                $container: er2cds,
                name: null!,
                joinClauses: []
            };
        }

        if (c.ExprType === 'TO_PARENT_ASSO' && relationship.
           source && relationship.target) {
            const associationToParent = assocDef.find(a => a.
               AssocName === c.AssocName);
            if (associationToParent?.CardMin === 1) {
                relationship.source.cardinality = '1';
            } else {
                relationship.source.cardinality = '0..N';
            }

            if (associationToParent?.CardMax === 1) {
                relationship.target.cardinality = '1';
            } else {
                relationship.target.cardinality = '0..N';
            }

            relationship.type = 'association-to-parent';
            er2cds.relationships.push(relationship);

            relationship = {
                $type: 'Relationship',
                $container: er2cds,
                name: null!,
                joinClauses: []
            };
        }

        if (c.ExprType === ’COMPOSITION’ && relationship.source
            && relationship.target) {
            const composition = assocDef.find(a => a.AssocName
               === c.AssocName);
            if (composition?.CardMin === 1) {
                relationship.source.cardinality = ’1’;
            } else {
                relationship.source.cardinality = ’0..N’;
            }

            if (composition?.CardMax === 1) {
                relationship.target.cardinality = ’1’;
            } else {
                relationship.target.cardinality = ’0..N’;
            }

            relationship.type = ’composition’;
            relationship.joinClauses = [];
            er2cds.relationships.push(relationship);

            relationship = {
                $type: ’Relationship’,
                $container: er2cds,
                name: null!,
                joinClauses: []
            };
        }
    });
}

interface IImportFromClause {
    EntityName: string,
    UnionNumber: string,
    BaseObjAlias: string,
    BaseObjPos: number,
    ParameterName: string,
    ParameterComponent: string,
    As4local: string,
    ParameterValue: string,
    BaseObjName: string,
    BaseObjType: string,
    ParameterComponentName: string
}

interface IImportSelectionList {
    EntityName: string,
    UnionNumber: string,
    ElementAlias: string,
    ElementPos: number,
    StackLine: string,
    As4local: string,
    ExprType: string,
    ExprSubtype: string,
    ExprParamCount: string,
    BaseObjAlias: string,
    BaseObjName: string,
    BaseObjType: string,
    BaseElementName: string,
    BaseElementNameRaw: string,
    FunctionName: string,
    IsTypePreserving: string,
    Operator: string,
    LiteralValue: string,
    ParameterName: string,
    SessionVarExpr: string,
    DataElement: string,
    Keyflag: string,
    IsVirtual: string,
    IsGenerated: string,
    IsGeneratedLocalized: string,
    AssocName: string,
    AssocBaseType: string,
    Datatype: string,
    Leng: string,
    Decimals: string,
    AnnoName: string,
    AnnoQualifier: string,
    AnnoValue: string,
    AnnoValueUnescaped: string,
    AnnoType: string,
    IsPathField: string,
    FieldNameRaw: string
}

interface IImportCondition {
    EntityName: string,
    UnionNumber: string,
    Pos: number,
    AssocName: string,
    StackLine: number,
    ConditionType: string,
    As4local: string,
    ExprType: string,
    ExprParacount: string,
    BaseobjAlias: string,
    BaseobjName: string,
    BaseobjType: string,
    FieldName: string,
    FieldIndex: string,
    Function: string,
    Operator: string,
    Literal: string,
    ParameterName: string,
    AssocBaseType: string,
    JoinOperation: string,
    SessionVarExpr: string,
    Rollname: string,
    Datatype: string,
    Leng: string,
    Decimals: string,
    AssocIsDefinedLocally: string,
    AssocIsDerivedLocally: string,
    AssocIsExposed: string,
    IsGenerated: string,
    AssocIsManage: string
}

interface IImportAssocDef {
    EntityName: string,
    UnionNumber: string,
    AssocName: string,
    As4local: string,
    AssocType: string,
    AssocTarget: string,
    AssocTargetType: string,
    CardMin: number,
    CardMax: number,
    ExtendName: string,
    AssocNameRaw: string,
    RedefinedFromEntity: string,
    RedefinedFromAssocNam: string
}
```

Listing A.2: Import of an exisitng CDS view entity.

# APPENDIX B

# Evaluation

## B.1 Experiment Results - Create

| Name | Relations | Syntax | Output | Text |
|---|---|---|---|---|
| Z_I_MKPF | 0 | X | X | |
| Z_I_BESTELLANFORDERUNG | 1 | X | X | Inner join instead of Join (equivalent). Join Clause different order (equivalent). |
| Z_I_SERIALNMBRDELIVERY | 2 | X | X | Table-prefix on attributes (equivalent). |
| Z_I_CHANGEDOCUMENTITEM | 3 | X | X | |
| Z_I_DELIVANALYSIS | 4 | X | X | |
| Z_I_PRODUCTION | 4 | X | X | Inner join instead of Join (equivalent). |
| Z_I_PURCHDOCBASE | 4 | X | X | Association with explicit min, max and $projection (equivalent). |
| Z_I_AUSSENLAGER | 8 | X | X | Inner join instead of Join (equivalent). Join Clause in different order (equivalent). |
| Z_I_SDSERIALNUMBER | 12 | X | X | Inner join instead of Join (equivalent). Association with min, max and $projection (equivalent). |
| Z_I_Notification | 19 | X | X | |
| I_ACMSettlmtCtnHdrCostRevCkpt | 1 | X | X | Join Clause in different order (equivalent). |
| I_MobileUserSession | 1 | X | X | Join Clause in different order (equivalent). Tableprefix on attributes (equivalent). |
| I_MDOFIELD | 2 | X | X | Association in different order (equivalent). Table-prefix on attributes (equivalent). |
| I_FinSubstitutionRuleSubstnTP | 2 | X | X | Table-prefix on attributes (equivalent). |
| I_FieldLogisticsSupplier | 3 | X | | Attributes of subassociation missing. Where-Clause missing. |
| I_BkPOABankAccount | 4 | X | X | Table-prefix on attributes (equivalent). |
| I_BusPartMobileNumberGovTP | 4 | X | | Table-prefix on attributes (equivalent). Reexport of subassociation missing. Where-Clause missing. |
| I_BPRelshpCntctPersnEmlAddrTP | 4 | X | | Attributes of subassociation missing. Reexport of subassociation missing. |
| I_ProdnRtgOperationTP | 10 | X | | Join Clause only equals. Where-Clause missing. Join Clause on fix values not supported. |
| I_SourcingProject | 23 | X | | Multiple association to same table not working. Cast/Concat not supported. Specific export of associations. |

## B.2 Experiment Results - Import

| Name | Relations | Elements | Imported | Display | Text | Warnings | Errors |
|---|---|---|---|---|---|---|---|
| /DMO/I_Agency | 1 | 10 | X | X | | | |
| /DMO/I_Airport | 1 | 5 | X | X | | | |
| /DMO/I_Booking_D | 6 | 18 | X | X | | | |
| /DMO/I_Booking_Status_VH | 1 | 2 | X | X | | | |
| /DMO/I_Booking_Status_VH_Text | 1 | 4 | X | X | | | |
| /DMO/I_BookingSupplement_D | 4 | 12 | X | X | | | |
| /DMO/I_Carrier | 1 | 9 | X | X | | | |
| /DMO/I_Connection | 1 | 9 | X | X | | | |
| /DMO/I_Customer | 1 | 11 | X | X | | | |
| /DMO/I_Flight | 3 | 11 | X | X | | | |
| /DMO/I_Overall_Status_VH | 1 | 2 | X | X | | | |
| /DMO/I_Overall_Status_VH_Text | 1 | 4 | X | X | | | |
| /DMO/I_SupplementCategory_VH | 1 | 2 | X | X | | | |
| /DMO/I_SupplementCategory_VH_T | 1 | 4 | X | X | | | |
| /DMO/I_SupplementText | 2 | 6 | X | X | | | |
| /DMO/I_Travel_D | 5 | 21 | X | X | | | |
| /DMO/I_Travel_Status_VH | 1 | 2 | X | X | | | |
| /DMO/I_Travel_Status_VH_Text | 1 | 4 | X | X | | | |
| A_ACMAPPDOCOVERVIEW | 0 | 53 | X | X | | | |
| A_ACMSETTLEMENTUNITDETAILS | 0 | 40 | X | X | | | |
| A_ACMTRDGCONTRITMDET | 0 | 22 | X | X | | | |
| A_BATCH | 0 | 25 | X | X | | | |
| A_CABUSPARTINVOICEITEM | 2 | 66 | X | X | | | |
| A_CABUSPARTPAYMENTITEM | 2 | 29 | X | X | | | |
| A_CADOCUMENTBPITEM | 3 | 143 | X | X | | | |
| A_CFINRPLDPURCHASEORDERITEM | 0 | 49 | X | X | | | |
| A_CFINRPLDSALESDOCUMENTITEM | 4 | 34 | X | X | | | |
| A_CFINRPLDSUPLRINVCITMPOREF | 3 | 22 | X | X | | | |
| A_CHANGEMASTER_1 | 12 | 28 | X | X | Join Clause with fix value not supported. | | Relationship has no join clauses. |
| A_CHANGERECORD | 13 | 26 | X | X | | | |
| A_CHANGERECORDREFERENCEDOC | 3 | 15 | X | X | | | |
| A_CNDNCONTRCNDNRECORD | 0 | 53 | X | X | | | |
| C_DFS_RELOCPROJWBSELMNTHIERTP | 2 | 20 | X | X | | | |
| C_FLDLOGSOVRDITEMSBYITEMTYPE | 2 | 12 | X | X | | Datatype of attributes FLDLOGSOVERDUEBYVALUE and FLDLOGSOVERDUEBYVALUE incompatible | |
| C_FLDLOGSRCPTSRETSATREMOTE | 2 | 10 | X | X | | | |
| C_FLDLOGSRECEIPTSBYSTATUSTP | 2 | 15 | X | X | Join Clause on Association attribute not supported | | Relationship has no join clauses. |
| C_INSPLOTOPCHARCRSLTRECG | 19 | 140 | X | X | Upper/Lowercase of alias in from clause. | | |
| C_INSPLOTRESULTVALUERSLTRECG | 6 | 30 | X | X | | | |
| C_INSURPLCYCONTRACT | 8 | 79 | X | X | | | |
| C_INSURPLCYPOLICY | 9 | 42 | X | X | | | |
| C_INTCOVALUECHAINDOCUMENTITMTP | 5 | 21 | X | X | Hierarchical associations not supported. | | Relationship has no join clauses. |
| C_MAINTJOBWRKITMCOMPHISTPRODVH | 6 | 22 | X | X | | | |
| C_MSPRMOVEMENT | 2 | 11 | X | X | | | |
| C_PROJECTBILLINGELEMENTTP | 2 | 66 | X | X | | | |
| C_SOURCINGPROJECTCOMPARE | 6 | 32 | X | X | | | |
| C_SRCGPROJDMNDDISTRSUMMARY | 4 | 16 | X | X | | | |
| C_SRCGPROJQTNITEMCOMPARE | 8 | 36 | X | X | Join Clause with fix value not supported. | | Relationship has no join clauses. |
| C_SRCGPROJQUOTATIONCOMPARE | 6 | 38 | X | X | Join Clause with fix value not supported. | | Relationship has no join clauses. |
| C_VARCONFIGNASSIGNEDVALUE | 2 | 16 | X | X | | | |
| C_VARCONFIGNCHARACTERISTIC | 4 | 14 | X | X | | | |
| C_VARCONFIGNINCONSTCYCHARC | 2 | 8 | X | X | | | |
| C_VARCONFIGNINSTCECHARCGROUP | 2 | 9 | X | X | | | |
| C_VARCONFIGNPOSSIBLEVALUE | 2 | 17 | X | X | | | |
| C_VARCONFIGNVARIANTCONDITION | 2 | 8 | X | X | | | |
| DEMO_CDS_CURRENCY | 0 | 4 | X | X | | | |
| DEMO_CDS_FLIGHTS | 2 | 11 | X | X | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| I_ABAPPACKAGE | 3 | 14 | X | X | | | |
| I_ABAPPACKAGETEXT | 2 | 5 | X | X | | | |
| I_BPRELSHPCNTCTPERSNADDRESSTP | 8 | 38 | X | X | | | |
| I_BPRELSHPCNTCTPERSNEMLADDRTP | 4 | 17 | X | X | | | |
| I_BUSINESSPARTNERADDRESSPROCTP | 8 | 70 | X | X | | | |
| I_BUSINESSPARTNERGOVTP | 30 | 122 | X | X | Multiple Relations to same entity name. Alias not resolved correctly. | Datatype of attributes BUSINESSPARTNERCATEGORY and BUSINESSPARTNERCATEGORY incompatible | |
| I_EWM_HANDLINGUNITHDRLOGVH | 1 | 5 | X | X | | | |
| I_EWM_INBOUNDDELIVERYTYPE_2 | 0 | 4 | X | X | | | |
| I_EWM_OUTBDELIVDOCTYPEVH_2 | 0 | 4 | X | X | | | |
| I_EWM_OUTBOUNDDELIVORDERTYPE_2 | 0 | 4 | X | X | | | |
| I_EWM_STORAGEBINVH | 3 | 12 | X | X | | | |
| I_EWM_STORAGETYPEVH | 0 | 3 | X | X | | | |
| I_EWM_WAREHOUSENUMBERVH | 0 | 5 | X | X | | | |
| I_EWM_WAREHOUSEORDERT_2 | 1 | 37 | X | X | | | |
| I_FLDLOGSEWMHNDLGUNITHDR | 1 | 8 | X | X | | | |
| I_FLDLOGSSHPTCTNCERTEXPIRYDATE | 0 | 2 | X | X | | | |
| I_INSURCLMCLSREASON | 2 | 10 | X | X | | | |
| I_INSURCLMFNOLDAMAGEDOBJECTTP | 12 | 89 | X | X | | | |
| I_INSURCLMFNOLTP | 15 | 52 | X | X | | | |
| I_INSURPLCYINQRYOPNAPPLCNT | 0 | 2 | X | X | | | |
| I_PRODNRTGOPERATIONTP | 10 | 132 | X | X | | | |
| I_PRODUCTSPECIFICATIONTP | 10 | 29 | X | X | Only Equal-Sign is supported in Join Clause. | | |
| P_EWM_INBOUNDDLVHEADER01_2 | 2 | 13 | X | X | Join Clause on Association attribute not supported | | Relationship null has no join clauses |
| P_EWM_OUTBDLVORDDOCADDR01_2 | 2 | 20 | X | X | | | |
| P_EWM_WAREHOUSETASKITEM_2 | 9 | 167 | X | X | Multiple Relations to same entityname. Alias not resolved correctly. | | |
| P_EWM_WHSEDOCUMENTITEM | 26 | 92 | X | X | Multiple Relations to same entity name. Alias not resolved correctly. | | |
| P_INSUROPENAPPLCONTRACT | 1 | 6 | X | X | Union is not supported. | | |
| P_MAINTENANCEORDEROBJECTLIST | 6 | 12 | X | X | | | |
| P_MAINTORDERPURORDLINK | 2 | 10 | X | X | | | |
| P_MSTRRCPOPDOCPRTASSGMT | 1 | 29 | X | X | | | |
| P_RAANLYTSSSPRICETESTDISTANCE | 1 | 19 | X | X | | | |
| P_RAANLYTSUNITSSPRICEBYNAME | 1 | 5 | X | X | | | |
| P_REINTEGOBJMSMTHIERCALC5 | 2 | 7 | X | X | | | |
| R_BILLINGDOCUMENTITEMTP | 3 | 223 | X | X | | | |
| R_SRCGPROJQTNITMCMMDTYTERMTP | 4 | 94 | X | X | | | |
| R_TRDGDOC | 3 | 325 | X | X | | | |
| R_WASTESTREAMDSPLCHNLTP | 3 | 26 | X | X | | | |
| R_WASTESTREAMTP | 3 | 21 | X | X | | | |
| R_WBSELEMENTTP_2 | 7 | 121 | X | X | Self-associations not supported. | | Relationship has no join clauses. |
| R_WORKCENTERCOSTCENTERTP | 1 | 21 | X | X | | | |
| R_WORKCENTERTP | 5 | 60 | X | X | | | |
| R_WORKCTRPOOLEDCAPACITYTEXTTP | 2 | 5 | X | X | | | |
| R_WRNTYCLAIMITEMTP | 8 | 44 | X | X | | | |
| R_WRNTYCLAIMVERSIONTP | 7 | 36 | X | X | | | |

## B.3 Case Study - Task Description

**ZER2CDS_AIRPORT** Create a CDS view entity with the name **ZER2CDS_AIRPORT**. The view should contain all attributes of **/DMO/I_AIRPORT**. For better readability the following aliases shall be used:

- /DMO/I_AIRPORT as Airport
- AIRPORTID as AirportId (keyfield)
- CITY as City
- COUNTRYCODE as CountryCode
- NAME as Name

**ZER2CDS_FLIGHT** Create a CDS view entity with the name **ZER2CDS_FLIGHT**. The view should join **/DMO/FLIGHT** (alias Flight) with **/DMO/I_CARRIER** (alias Carrier) and **/DMO/I_CONNECTION** (alias Connection). The following attributes shall be included in the resulting view:

- /DMO/FLIGHT-CARRIER_ID as CarrierId (keyfield)
- /DMO/FLIGHT-CONNECTION_ID as ConnectionId (keyfield)
- /DMO/FLIGHT-FLIGHT_DATE as FlightDate (keyfield)
- /DMO/FLIGHT-SEATS_OCCUPIED as SeatsOccupied
- /DMO/I_CARRIER-NAME as AirlineName
- /DMO/I_CONNECTION-DEPARTUREAIRPORT as DepatureAirport
- /DMO/I_CONNECTION-DESTINATIONAIRPORT as DestinationAirport

The relationship between /DMO/FLIGHT and /DMO/I_CARRIER shall be represented as an inner join on CARRIER_ID = AIRLINEID. For the relationship between /DMO/FLIGHT and /DMO/I_CONNECTION a left outer join on CARRIER_ID = AIRLINEID and CONNECTION_ID = CONNECTIONID shall be used. Keep in mind, that the no more attributes than stated above should appear in the resulting view.

**ZER2CDS_BOOKING** Create a CDS view entity with the name **ZER2CDS_BOOKING**. The view should be based on **/DMO/A_BOOKING_D** and contain the following attributes:

- /DMO/A_BOOKING_D-BOOKING_UUID as BookingUUID (keyfield)

- /DMO/A_BOOKING_D-PARENT_UUID as TravelUUID
- /DMO/A_BOOKING_D-BOOKING_DATE as BookingDate
- /DMO/A_BOOKING_D-CUSTOMER_ID as CustomerId
- /DMO/A_BOOKING_D-CARRIER_ID as AirlineId
- /DMO/A_BOOKING_D-CONNECTION_ID as ConnectionId
- /DMO/A_BOOKING_D-BOOKING_STATUS as BookingStatus

Furthermore relationships to the following entities shall be realized. They should be accessed only on demand:

- /DMO/I_CUSTOMER on CUSTOMER_ID = CUSTOMERID
  - Alias _Customer
  - Cardinality [1..1]
- /DMO/I_CARRIER on CARRIER_ID = AIRLINEID
  - Alias _Carrier
  - Cardinality [1..1]
- /DMO/I_CONNECTION on CARRIER_ID = AIRLINEID and CONNECTION_ID = CONNECTIONID
  - Alias _Connection
  - Cardinality [1..1]
- /DMO/I_BOOKING_STATUS_VH on BOOKING_STATUS = BOOKINGSTATUS
  - Alias _BookingStatus
  - Cardinality [1..1]
- /DMO/I_TRAVEL_D on PARENT_UUID = TRAVELUUID
  - Alias _Travel
  - Parent relationship
- /DMO/I_BOOKINGSUPPLEMENT_D
  - Alias _BookingSupplement
  - Composition relationship
  - Cardinality [0..*]

## B.4 Case Study - Survey

1. Consent to participate in the study to evaluate ER2CDS. All information we collect in this study will be treated confidentially and used for scientific purposes only. Any information you provide will be evaluated anonymously, i.e. will not be linked to you personally. I agree that my data will be stored and analyzed exclusively for scientific purposes at the Institute for Information Systems. After completion of the research project, all data that can be linked to my person will be deleted. I can cancel my participation in this study at any time without giving reasons and withdraw my consent. This will not result in any disadvantages for me. I agree to participate in the study. Furthermore, I authorize the processing of the collected data for scientific purposes.

   a) Yes

   b) No

2. I am a

   a) Business expert

   b) Developer

3. I was able to create ZER2CDS_AIRPORT using ER2CDS.

   a) Strongly disagree

   b) Disagree

   c) Neutral

   d) Agree

   e) Strongly agree

4. I would be able to create ZER2CDS_AIRPORT by using only the textual syntax of CDS.

   a) Strongly disagree

   b) Disagree

   c) Neutral

   d) Agree

   e) Strongly agree

5. I was able to create ZER2CDS_FLIGHT using ER2CDS.

   a) Strongly disagree

   b) Disagree

c) Neutral
d) Agree
e) Strongly agree

6. I would be able to create ZER2CDS_FLIGHT by using only the textual syntax of CDS.

    a) Strongly disagree
    b) Disagree
    c) Neutral
    d) Agree
    e) Strongly agree

7. I was able to create ZER2CDS_BOOKING using ER2CDS.

    a) Strongly disagree
    b) Disagree
    c) Neutral
    d) Agree
    e) Strongly agree

8. I would be able to create ZER2CDS_BOOKING by using only the textual syntax of CDS.

    a) Strongly disagree
    b) Disagree
    c) Neutral
    d) Agree
    e) Strongly agree

9. I think that I would like to use ER2CDS frequently.

    a) Strongly disagree
    b) Disagree
    c) Neutral
    d) Agree
    e) Strongly agree

10. I found ER2CDS unnecessarily complex.

    a) Strongly disagree

   b) Disagree
   c) Neutral
   d) Agree
   e) Strongly agree

11. I thought ER2CDS was easy to use.

   a) Strongly disagree
   b) Disagree
   c) Neutral
   d) Agree
   e) Strongly agree

12. I think that I would need the support of a technical person to be able to use ER2CDS.

   a) Strongly disagree
   b) Disagree
   c) Neutral
   d) Agree
   e) Strongly agree

13. I found the various functions in ER2CDS were well integrated.

   a) Strongly disagree
   b) Disagree
   c) Neutral
   d) Agree
   e) Strongly agree

14. I thought there was too much inconsistency in ER2CDS.

   a) Strongly disagree
   b) Disagree
   c) Neutral
   d) Agree
   e) Strongly agree

15. I would imagine that most people would learn to use ER2CDS very quickly.

   a) Strongly disagree

b) Disagree
c) Neutral
d) Agree
e) Strongly agree

16. I found ER2CDS very cumbersome to use.

    a) Strongly disagree
    b) Disagree
    c) Neutral
    d) Agree
    e) Strongly agree

17. I felt very confident using ER2CDS.

    a) Strongly disagree
    b) Disagree
    c) Neutral
    d) Agree
    e) Strongly agree

18. I needed to learn a lot of things before I could get going with ER2CDS.

    a) Strongly disagree
    b) Disagree
    c) Neutral
    d) Agree
    e) Strongly agree

19. In the future I will use ER2CDS over the textual syntax to create CDS view entities.

    a) Yes
    b) No

20. What do you like most about ER2CDS?

21. What do you like least about ER2CDS?

22. What suggestions do you have for improving ER2CDS?

## B.5 Case Study - Responses

| ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Yes | Developer | 5 Strongly agree | 5 Strongly agree | 5 Strongly agree | 5 Strongly agree | 5 Strongly agree | 5 Strongly agree | 3 Neutral | 1 Strongly disagree | 5 Strongly agree |
| 2 | Yes | Business expert | 5 Strongly agree | 3 Neutral | 5 Strongly agree | 3 Neutral | 5 Strongly agree | 2 Disagree | 4 Agree | 1 Strongly disagree | 5 Strongly agree |
| 3 | Yes | Developer | 5 Strongly agree | 4 Agree | 5 Strongly agree | 4 Agree | 5 Strongly agree | 4 Agree | 4 Agree | 1 Strongly disagree | 5 Strongly agree |
| 4 | Yes | Business expert | 5 Strongly agree | 2 Disagree | 5 Strongly agree | 2 Disagree | 5 Strongly agree | 2 Disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree |
| 5 | Yes | Business expert | 5 Strongly agree | 2 Disagree | 5 Strongly agree | 2 Disagree | 5 Strongly agree | 2 Disagree | 5 Strongly agree | 1 Strongly disagree | 4 Agree |
| 6 | Yes | Developer | 5 Strongly agree | 4 Agree | 5 Strongly agree | 4 Agree | 5 Strongly agree | 3 Neutral | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree |
| 7 | Yes | Developer | 5 Strongly agree | 5 Strongly agree | 5 Strongly agree | 5 Strongly agree | 5 Strongly agree | 5 Strongly agree | 2 Disagree | 2 Disagree | 5 Strongly agree |
| 8 | Yes | Business expert | 5 Strongly agree | 1 Strongly disagree | 4 Agree | 2 Disagree | 1 Strongly disagree | 1 Strongly disagree | 4 Agree | 3 Neutral | 3 Neutral |

| 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|
| 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | No |
| 4 Agree | 5 Strongly agree | 1 Strongly disagree | 4 Agree | 1 Strongly disagree | 5 Strongly agree | 3 Neutral | Yes |
| 2 Disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | Yes |
| 2 Disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 2 Disagree | Yes |
| 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | 1 Strongly disagree | 1 Strongly disagree | Yes |
| 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | 5 Strongly agree | 1 Strongly disagree | Yes |
| 1 Strongly disagree | 3 Neutral | 2 Disagree | 3 Neutral | 4 Agree | 5 Strongly agree | 1 Strongly disagree | No |
| 4 Agree | 5 Strongly agree | 2 Disagree | 4 Agree | 1 Strongly disagree | 3 Neutral | 3 Neutral | Yes |

| 20 | 21 | 22 |
|---|---|---|
| Simple to use | - | Automatic scrolling when adding and editing attributes |
| Enables the creation of CDS Views without the need for the technical background and coding knowledge | No explanation for relationship types | SAP Tool integration |
| The offer of a textual and visual editor | All attributes can only be added once (workaround possible) | List the attributes in the visual editor to select them |
| The visual editor with the input fields to define entities / attributes / relationships / ... | - | - |
| I was able to intuitively create CDS views, without any further instructions. | - | - |
| - | - | - |
| Textual and graphical editor. Integration of SAP data model. | Missing integration in BAS. | Support for all language features of CDS. |
| Creation of CDS views on a higher abstraction level. Easy and fast to create simple CDS views. | Still technical knowledge is required. | Add a beginners guide or documentation. |