Accepted for publication in Software and Systems Modeling (SoSyM)
https://www.springer.com/journal/10270

Model-driven Engineering of SAP Core Data Services - The BIGER2CDS Modeling Tool

Gallus Huber[1] and Dominik Bork[1*]

[1*]Business Informatics Group, TU Wien, Favoritenstrasse 9-11, Vienna, 1040, Austria.

*Corresponding author(s). E-mail(s): dominik.bork@tuwien.ac.at;
Contributing authors: gallus.huber@outlook.com;

Abstract

This paper introduces BIGER2CDS, a novel model-driven engineering approach and tool support for SAP Core Data Services (CDS). BIGER2CDS 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 modeling language. Based on web technologies and the Language Server Protocol (LSP), we realized a modeling tool for SAP CDS. Our 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 enable 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 BIGER2CDS modeling tool. Finally, BIGER2CDS 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. The tool can be freely downloaded from the VS Code marketplace: https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.er2cds.

Keywords: Model-driven engineering, SAP Core Data Services, Domain-specific language, CDS, Modeling tool, LSP, Langium, Sprotty

1 Introduction

Compared to code, models allow different stakeholders to get a technical understanding of an application without prior programming knowledge. Furthermore, models are crucial for managing the complexity of applications [20]. Combined with model-to-text transformations, a model-driven engineering (MDE) process allows the creation of all different kinds of applications [17].

Especially in the context of business informatics, business experts drive the development process. Allowing them to create applications independently by applying an MDE 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 is achieved as they can focus on the development of the remaining applications. Furthermore, maintaining applications with a graphical user interface leads to a better understanding and fewer errors [10, 11].

One of the most used enterprise information systems is the Enterprise Resource Planning (ERP) solution by SAP SE (SAP) [48]. At the core of SAP's ERP solution is its data structure, which forms the basis for domain- and

customer-specific customizations. Developers can create database views of that data structure with enhanced access functions, so-called *Core Data Services* (CDS) [31]. Furthermore, front-end applications for existing CDS Views can be generated [30]. However, CDS are currently developed in a textual editor, requiring heavy skills in SQL and additional CDS-specific syntax [31]. It is, therefore, impossible for users without a programming background to create CDS themselves. Currently, the CDS development process is twofold. First, the business experts state the needs and the data that should be retrieved informally using natural language. Then, developers implement the CDS accordingly. One of the significant problems with this approach is the lack of a shared understanding of the problem domain amongst these stakeholders. This leads to long development times and unsatisfactory results.

This work aims to bridge that gap between business experts and developers by creating a new model-driven engineering approach and a tool based on the Language Server Protocol (LSP) [3, 21]. The approach shall exploit an extended variant of Entity-Relationship (ER) diagrams [9], which allows 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. The tool's evaluation focuses on the tool's usability compared to the current text-based development approach. BIGER2CDS should contribute to embracing MDE processes in the SAP ecosystem. The overall objective of this research is to provide a tool that allows business experts and developers to create CDS more effectively.

To achieve this objective, we first need a hybrid tool that enables modeling of ER diagrams in a textual and graphical manner similar to the one proposed in [14]. This can be achieved by extending the ER modeling language [9] to cover all aspects of CDS and integrating the tool into the SAP ecosystem. Concretely, CDS supports specific features, e.g., *associations*, join relationships executed on demand, which should be represented in the modeling tool as first-class concepts. For the efficient utilization of our tool, it needs to be able to automatically represent the data model of the connected SAP ERP solution and allow users to select existing entities while creating their CDS views. 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 created models. Eventually, the resulting BIGER2CDS tool should come with a model-to-text-transformation that transforms the created ER2CDS models into corresponding textual CDS view specifications that can be used to realize CDS views within SAP. The correctness and quality of the generated code are crucial for user acceptance.

This paper is structured as follows. Section 2 provides all necessary background information that this paper will build upon. Section 3 discusses related works. Section 4 then elaborates on requirements for BIGER2CDS. In Section 5, the main concepts for BIGER2CDS are introduced and their implementation is reported. The results of the empirical evaluation of BIGER2CDS are reported in Section 6. A thorough discussion of the implications and limitations of this research is given in Section 7 before this paper is concluded in Section 8.

2 Background

2.1 SAP HANA

SAP SE was founded in 1972 by five former IBM employees and specialized in developing programs for materials management, financial accounting, and auditing [24]. SAP SE is regarded as one of the inventors of standard software [24] and, in 2024, will be one of the largest software companies, with a turnover of 31.2 billion euros in 2023 [41]. In 2015, SAP Business Suite 4 SAP HANA, or S/4HANA for short, was introduced. The switch to the new software generation includes the mandatory introduction of the HANA database, an in-memory database developed by SAP SE [46]. Similarly to its predecessor, S/4HANA is a standardized enterprise resource planning (ERP) system that allows for customization and extension [42]. Furthermore, it is possible to develop custom applications using the integrated technologies, e.g., *Core Data Services*, which is natively supported by SAP HANA [44].

This paper uses SAP HANA indirectly by creating a novel modeling language for CDS and exposing the data model of an existing S/4HANA system for the ER2CDS modeler to support the efficient creation of valid CDS.

2.2 Core Data Services

A virtual data model (VDM) can provide a semantically rich, reusable, and stable data model while abstracting technical details, thereby reducing complexity and providing a business-oriented view of the underlying database [44]. In the SAP S/4HANA, the VDM is implemented using Core Data Services (CDS) [29]. CDS supports a data-centric approach, delegating computations to the database layer [29]. In detail, CDS consists of the following languages, enabling users to create these semantically enriched data models [44]: **Data Definition Language (DDL)** used to create domain-specific data models, CDS entities; **Query Language (QL)** used to read data from defined models; **Data Control Language (DCL)** used to control access to the data model; and **Data Manipulation Language (DML)** used to write data. Furthermore, the CDS DDL allows for building hierarchies, which serve as a building block of the SAP VDM [40]. The VDM can be divided into three layers (see Fig. 1) [43]:

- **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, they can 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.

Technically, CDS DDL enhances SQL DDL [19] by further supporting 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 [40]. In the remainder of this section, we introduce the CDS DDL, particularly CDS entities and CDS view entities.

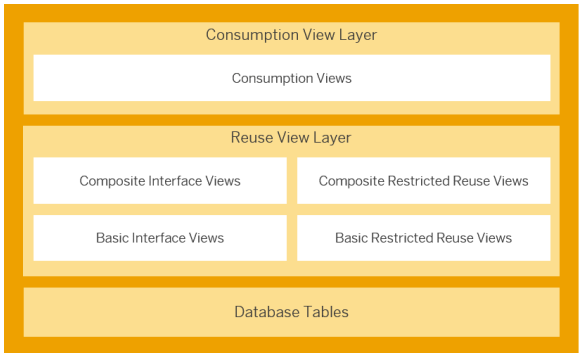

Fig. 1: Layers of the Virtual Data Model [43].

One can use the syntax presented in Listing 1 [39] to define a CDS view entity in CDS DDL. The *DEFINE VIEW ENTITY* keywords are used to specify a new CDS view entity, followed by the name of the view entity. The CDS view entity is eventually implemented by the *select_statement*, of which the syntax will be described afterward.

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

Listing 1: CDS view entity syntax.

As part of a CDS view entity, the select statement used to query a data source follows the syntax shown in Listing 2 [35]. The query begins with the *SELECT* keyword and is performed on the data sources defined in *data_source*, which can be database tables, CDS view entities, CDS table functions, CDS hierarchies, or the obsolete CDS DDIC-based views [35]. *association1* and *association2* further define CDS associations for the current *SELECT* statement. In the *select_list*, the components of the view entity are listed. Components of the *data_source* and the defined associations can be accessed via path expressions [35]. Further, the *clauses* in line 4 allow for defining conditions, groupings, or set operators [35].

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

Listing 2: CDS select syntax.

The syntax of the *data_source* is presented in Listing 3 [37]. It is used to define the data source directly using the entity name or via a SQL path expression. Also, it is possible to define joins (inner join, left outer join, right outer join), to combine multiple data sources [37].

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

Listing 3: CDS data source syntax.

CDS associations define relationships between CDS entities [36]. It can include fields of the association target in the current CDS view entity or expose these fields for reuse in other CDS entities. 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. 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. Furthermore, it allows for reusing these relationships. A particular type of CDS association is the composition, which consists of a *to-child* association and a *to-parent* association. The composition association can state an existential relationship between parent and child.

```
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 4: CDS association syntax.

The syntax of CDS associations is presented in Listing 4 [32–34]. The first two lines present the syntax of a simple association. In line 4, the syntax for a composition *to-child* association is shown. 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. Finally, lines 6 and 7 present the syntax for an *to-parent* association. 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.

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. Listing 5 shows the syntax of the *select_list* in line 1, followed by the syntax of a specific element in lines 3 to 9 [38].

```
1  ... element1, element2, ...
2
3  ... {[@element_annot1]
4       [@element_annot2]
5         ...
6         [KEY] { field [AS alias] } |
7               { expose_assoc    [AS
                    alias] }
8                   ...
9      }
```

Listing 5: CDS select list syntax.

The *clauses* of the CDS DDL closely follow the standard SQL DDL, and we will, therefore, not further detail their usage.

2.3 Core Data Service Example

In Listing 6, 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.

```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'CDS example 2'
define view entity DEMO_SALES_CDS_SO_I_VE
  as select from
    demo_sales_so_i
    association [0..1] to DEMO_SALES_CDS_MATERIAL_VE
        as _Material
        on $projection.material = _Material.material
    association to parent DEMO_SALES_CDS_SO_VE
        as _SalesOrder
        on $projection.parent_key = _SalesOrder.so_key
    composition [0..*] of DEMO_SALES_CDS_SO_I_SL_VE
        as _ScheduleLine
    {
      key so_item_key,
          parent_key,
          posnr,
          _Material.material as mat,
          _SalesOrder,
          _ScheduleLine
    }
```

Listing 6: CDS view entity example.

2.4 Langium-based Language Servers

One of the most crucial building blocks of web-based modeling is the Language Server Protocol (LSP) [3, 21]. 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 [22]. 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 [5, 28].

A *language server* can be defined as a server that provides *language-specific smarts* like code completion. The language server communicates with a *language client* via a standardized and extensible protocol, that is, the LSP. Furthermore, the LSP allows the definition of custom messages, which will be used in this paper to extend the standard protocol to support hybrid modeling—i.e., synchronous textual and graphical modeling—of SAD Core Data Services.

Langium[1] is an open source framework *"with first-class support for the Language Server Protocol, written in Typescript and running in Node.js."* [52] 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, and more. The effectiveness of creating Langium-based modeling tools has been shown in several recent works [6, 13, 25–27].

The most important element of a Langium project is the grammar file that describes the abstract syntax of the language for which the language server should be created. *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* [51]. Langium supports defining these rules using the Extended Backus-Naur Form (EBNF) Expressions and Regular Expressions [51]. These tokens are atomic and

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

have to return a primitive Typescript type such as *string*, *number*, *boolean*, *bigint*, or *Date* [51]. 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 [51].

```
1 Person:
2     'person' name=ID;
3
4 entry Model:
5     (persons+=Person | greetings+=
          Greeting)*;
```

Listing 7: Langium grammar parser rules.

In the next processing step, the *parser* creates the AST using defined sequences of tokens [51]. These valid sequences of tokens are defined using *parser rules*, which are written using EBNF [51]. A simple *parser rule* is shown in lines 1 and 2 of Listing 7. 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* [51].

Furthermore, the langium grammar allows to define an *entry rule*, as shown in lines 4 and 5 of Listing 7, which serves as the starting point for the parser [51]. 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* [51].

In addition, Langium supports defining *cross-referencing* directly in the grammar [51]. In Listing 8, 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 *'!'*.

```
1 Person:
2     'person' name=ID;
3 Greeting:
4     'Hello' person=[Person:ID] '!';
```

Listing 8: Langium grammar cross-referencing.

By definition, the following example in Listing 9 will be parsed successfully:

```
1 person Bob
2 Hello Bob !
```

Listing 9: Example for a valid cross-reference.

While the example in Listing 10 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*.

```
1 person Bob
2 Hello Laura !
```

Listing 10: Example for an invalid cross-reference.

Another feature of Langium is the scoping mechanism [50]. Scoping eases the process of *linking*, which is used to resolve references between elements within the language [50]. One only needs to define the scoping behavior of the defined language, and the linking process will be done by the framework [50].

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 [49]. However, the *LangiumDocument* has to be built further after the input is parsed, which happens depending on the state of the *LangiumDocument* [49]. The possible states are listed 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 [49]:

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, are furthermore illustrated in Fig. 2.

Finally, Langium has first-class support for LSP [50]. This allows us to easily create a language

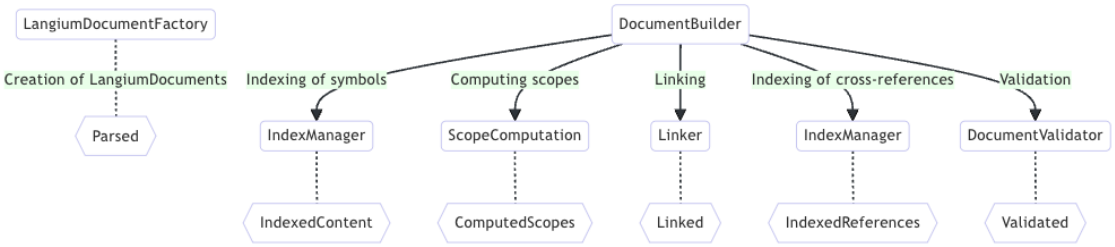

Fig. 2: Build stages of a *LangiumDocument* [49].

server that can then be used by multiple development tools [50]. 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. An exhaustive list of features supported by the Langium grammar can be found in [51].

3 Related Works

Many modeling tools for data modeling exist, including tools that support Entity Relationship modeling. One recently introduced ER modeling tool is BIGER [15, 16]. The tool allows hybrid modeling of ER in a textual or graphical manner within VS Code[2]. The implementation is based on LSP with a client-server architecture. For the client side, Sprotty[3] is used for the graphical interface, while VS Code provides the textual interface. BIGER introduced a domain-specific language implemented in Xtext[4] that also serves as the basis for the Java-based language server. Currently, BIGER only supports conventional data modeling with ER and code generation for SQL databases. There is no means of creating and representing SAP CDS views; neither is support provided to connect the tool with an existing SAP S/4HANA database schema.

ABAP Development Tools (ADT) for Eclipse is the recommended development tool for CDS. 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. However, the textual editor does not support a model-driven engineering process, and users are forced to implement CDS entities at the textual level. This establishes a high entry barrier for domain experts, rendering it unfeasible for them to understand and maintain the CDS directly and without the support of developers. Fig. 7 illustrates the usage of ADT within the Eclipse IDE. It becomes clear that domain experts are not ready to use such a notation.

Several VS Code extensions for CDS development exist. On the one hand, the ABAP CDS Language Support[5] 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[6] and core data services graphical modeler for VS Code[7] provide full language support, as well as graphical modeling support, but for SAP Cloud Application Programming (CAP), therefore not supporting SAP S/4HANA CDS.

Synopsis. The overview of related work shows that, as of now, there exists no open source solution that enables textual and graphical modeling of CDS. Existing solutions are either not fully compliant with the latest CDS version, lack graphical modeling support, or lack advanced developer support like validation and auto-completion. Consequently, we see a need for a fully operational hybrid CDS modeling editor that tightly integrates with a locally running SAP system.

[2] https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.erdiagram, last accessed: 10.07.2025
[3] https://sprotty.org/, last accessed: 10.07.2025
[4] https://eclipse.dev/Xtext/, last accessed: 11.07.2025
[5] https://marketplace.visualstudio.com/items?itemName=hudakf.cds
[6] https://marketplace.visualstudio.com/items?itemName=SAPSE.vscode-cds
[7] https://marketplace.visualstudio.com/items?itemName=SAPSE.vscode-wing-cds-editor-vsc

4 Requirements Elicitation

Based on our experience with existing tools and informal discussions with experts and practitioners, bigER2CDS is designed to serve as the primary tool for the model-driven engineering of CDS. The tool shall support graphical and textual (i.e., hybrid) modeling of a domain-specific modeling language designed specifically for the construction of CDS (**Req-1**). The tool shall further enable the transformation of created models into CDS view entities using a model-to-text transformation (**Req-2**). To further ensure high usability, bigER2CDS should allow users to connect to existing SAP S/4HANA systems (**Req-3**).

bigER2CDS is intended to be used within VS Code and Business Application Studio (BAS), a browser-based development environment in the professional SAP environment (**Req-4**). No further dependencies should be required to run bigER2CDS (**Req-5**). Therefore, a pure extension of the existing ER modeling tool bigER is not feasible as it requires a Java runtime environment.

The ER2CDS modeling language must support the following CDS concepts (**Req-6**): *Root element definition*, *Entity definition*, *Attribute definition*, *Relationship definition*, and *Join clause definition*. bigER2CDS's textual and graphical representation of models should support all functionality to create and modify ER2CDS models (**Req-7**). Furthermore, bigER2CDS should enable truly *hybrid modeling* by offering background mechanisms that keep the graphical and textual representations permanently synchronized. To ensure high-quality models, an *extended validation* shall be implemented (**Req-8**). 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 entities and attributes used. A *model-to-text transformation* shall be implemented to create a CDS view entity for an ER2CDS model. Furthermore, to *import* existing CDS view entities, a parser that extracts the entity's information from an SAP S/4HANA system's database must be implemented. Finally, to ensure usability, bigER2CDS should also implement features that are widely common in textual and graphical modeling. These features include syntax support, renaming, layout support, and auto-completion/value help (**Req-9**).

5 Realization of bigER2CDS

In the following, we report on the conceptual design of the bigER2CDS tool. We detail the realization of the textual and graphical concrete syntax, the model editing support, the integration with SAP S/4HANA, and eventually show the realized bigER2CDS tool.

5.1 Architecture

Fig. 3 illustrates a high-level view of the solution architecture. On the one hand, VS Code and the extension host, which includes the ER2CDS language client, as well as the webview used for diagramming, and, on the other hand, the ER2CDS language server. The language server and the client run in separate processes and communicate via LSP actions. In general, the language client orchestrates all communication with the language server. This includes the Sprotty integration, which sends and receives messages through the language client.

The VS Code extension serves as the entry point of bigER2CDS. It is responsible for instantiating the language client and hosting the webview. Furthermore, the language and grammar for the textual editor, as well as VS Code native commands are defined within the extension. The creation of the language client is implemented straightforwardly using the *vscode-languageclient* npm module that allows VSCode extensions to easily integrate LSP-based language servers[8]. bigER2CDS uses the proxy pattern to handle and forward commands to the language server. This allows the extension of the payload with information stored within the extension. One use case is to handle secrets, e.g., SAP user credentials, within the *vscode.ExtensionContext.secrets.store* and provide them to the language server when needed.

The bigER2CDS webservice itself is implemented as a CDS view entity. It exposes 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, like the set of entities and attributes present in the connected SAP system.

[8] https://www.npmjs.com/package/vscode-languageclient

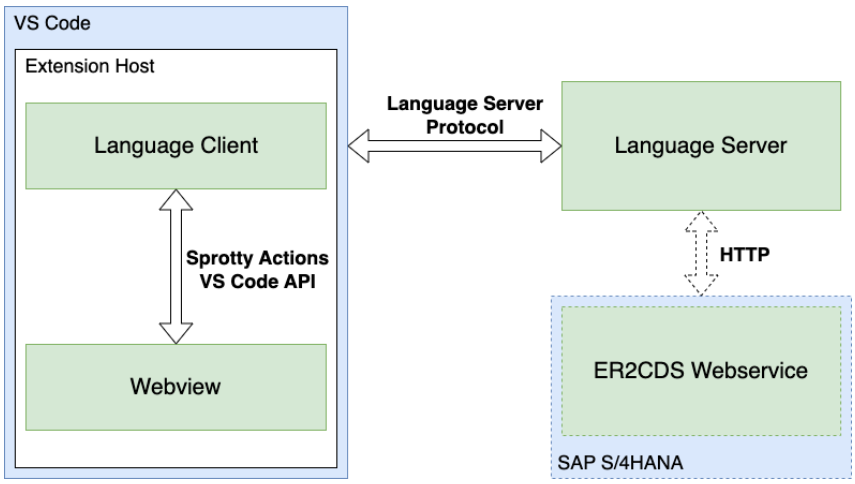

Fig. 3: Architecture of BIGER2CDS with all three main components.

The Langium-based ER2CDS language server implements custom extensions to the standard protocol to, e.g., handle commands sent by the VS Code extension, generate the ER2CDS diagram, validate the ER2CDS model, and synchronize the textual and the graphical model representations.

Furthermore, the language server implements two key functionalities, the **generateCDS** for generating a CDS view entity from a given ER2CDS model and **importCDS** for importing existing CDS view entities. The **generateCDS** feature first extracts the model of the Langium document, then validates it, generates the CDS view entity source code using the model-to-text transformation, and finally serializes the code into the file system. The **importCDS** feature first requests the CDS view entity information from the webservice, then transforms the response into a valid ER2CDS model, serializes this model into its textual representation, and, finally, writes the ER2CDS source into the file system which triggers the generation of the Langium model.

The webview is implemented using Sprotty. The views themselves are implemented using Scalable Vector Graphics (SVG). Each view class implements a *render* method for displaying the element. Furthermore, 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. 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. These central services, especially the *DiagramEditorService*, are available throughout the module using dependency injection.

5.2 ER2CDS Syntax

Fig. 4 visualizes the abstract syntax of the ER2CDS grammar using plantUML. The abstract syntax represents edges as object types following the abstract syntax specification technique introduced in [4]. A valid *ER2CDS* model is composed of an *ER2CDS* object, which composes *Entities* and *Relationships*, and a *RelationshipEntity*. An entity in ER2CDS is composed of *Associations*, *EntityWhereClauses*, and *Attributes*. *Relationships* in ER2CDS compose a *RelationshipJoinClause* which itself composes two *Attributes*. Finally, an *Attribute* has a *DataType*.

Next, we will briefly explain the concepts present in the abstract syntax of ER2CDS. ER2CDS: Das ist die Root Entity des Models. Die sollte nur einfach vorhanden sein. **Entity** Represents a table/view/CDS on which the new CDS is based. One can specify whether the table is relevant for the generated CDS and also aliases

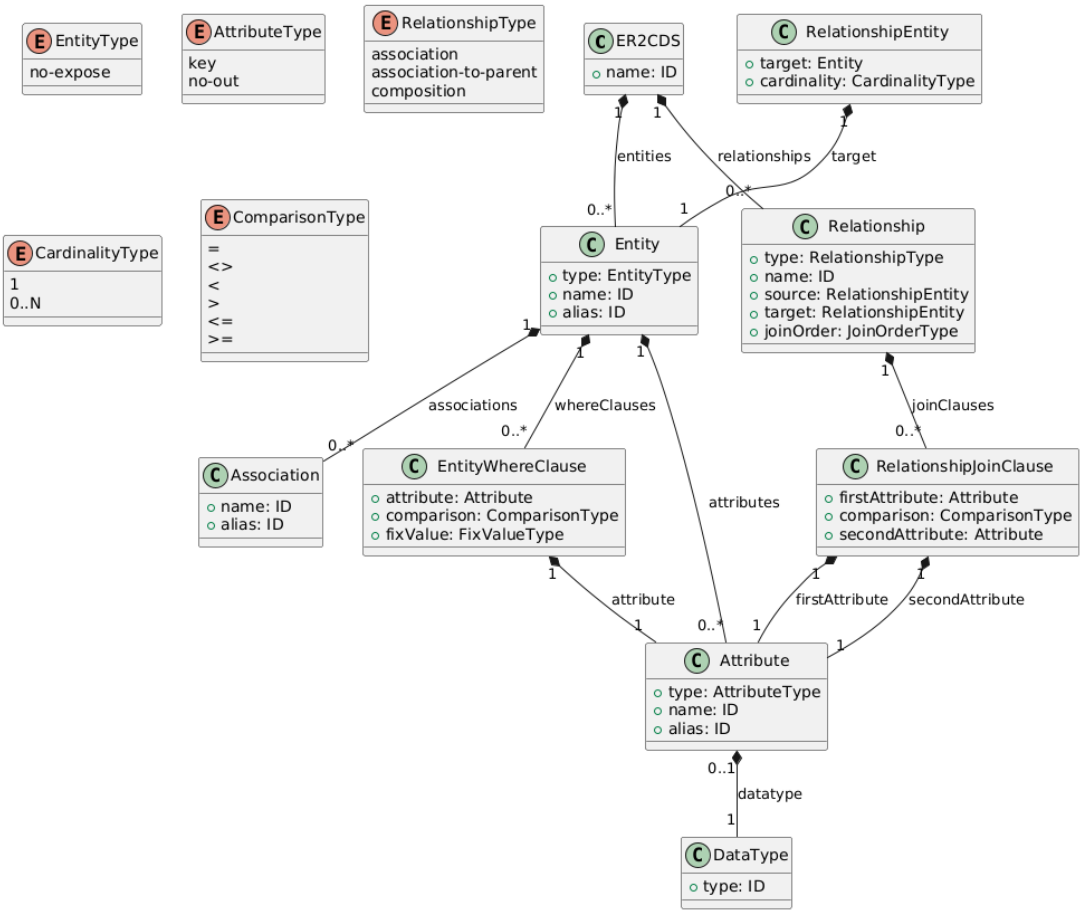

Fig. 4: Abstract syntax of ER2CDS as UML class diagram

for referencing the entity. **EntityWhereClause** An additional condition can be specified for each entity. In the generated view, the individual conditions are summarized as a single WHERE clause. **Relationship** Holds a reference to the source/target (RelationshipEntity). Also contains the type and the conditions (RelationshipJoinClause) of the relationship. **RelationshipEntity** This concept contains a reference to the entity plus the cardinality of the relationship, which is being used in the Langium grammar. **RelationshipJoinClause** Contains the two attributes and the comparison operator for joining. **Association** Associations can be used in CDS to express relationships to other tables. Unlike "normal" JOIN relationships, associations are only evaluated at runtime depending on whether a field in the association was requested or not. The JOIN is only executed on the table if necessary, as the views are not persisted and represent a virtual data model. These associations can also be "exposed", i.e., one can access the relationships and use fields from them. **Attribute** Attribute definition. **DataType** Datatype definition.

In the following, we will describe how we realized the textual (Section 5.2.1) and the graphical concrete syntax (Section 5.2.2) of ER2CDS.

5.2.1 Textual Concrete Syntax

Listing 11 presents the core of the ER2CDS DSL Langium grammar specification (the complete grammar specification is provided in Appendix A). A valid model starts with the keyword `er2cds` followed by the `name` of the model, zero-to-many `entities`, and zero-to-many `relationships`. Entities can have `attributes` which conform to a `datatype`. Each relationship connects a `source` and a `target entity`.

```
grammar ER2CDS
entry ER2CDS:
  'er2cds' name=ID
  (entities+=Entity | relationships+=Relationship)*;
Entity:
  (type=EntityType)? 'entity' name=ID '{'
    ('alias' alias=ID)?
    (attributes+=Attribute)*
    ('expose' (associations+=Association)*)?
    ('where' (whereClauses+=EntityWhereClause)*)?
  '}';
Attribute:
  (type=AttributeType)? name=ID (':' datatype=DataType)? ('as' alias=ID)?;
Association:
  name=ID ('as' alias=ID)?;
EntityWhereClause:
  (attribute=[Attribute:ID] comparison=ComparisonType fixValue=FixValueType);
Relationship:
  (type=RelationshipType)? 'relationship' name=ID '{'
    ((source=RelationshipEntity)? (('->' target=RelationshipEntity))?)?
    ('join' 'order' joinOrder=JoinOrderType)?
    (joinClauses+=RelationshipJoinClause)*
  '}';
RelationshipEntity:
  target=[Entity:ID] ('['
    cardinality=CardinalityType
  ']')?;
RelationshipJoinClause:
  (firstAttribute=[Attribute:ID] comparison=ComparisonType secondAttribute=[
      Attribute:ID]);
...
EntityType returns EntityType:
  NO_EXPOSE
;
...
AttributeType returns AttributeType:
  KEY | NO_OUT ;
...
RelationshipType returns RelationshipType:
  ASSOCIATION | ASSOCIATION_TO_PARENT | COMPOSITION ;
...
CardinalityType returns CardinalityType:
  ONE | ZERO_MANY ;
...
FixValueType returns FixValueType:
  CHAR | INT | ONE
;
...
ComparisonType returns ComparisonType:
  EQUAL | NOT_EQUAL | LOWER_THAN | GREATER_THAN | LOWER_EQUAL | GREATER_EQUAL
;
...
```

Listing 11: ER2CDS DSL grammar implemented using the Langium grammar.

In Listing 12, 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[9] and the use of an alias and

[9] `no-out` attributes are present in the model, e.g., to specify join conditions but do not form part of the generated CDS view entities.

```
er2cds EmployeeManagesDepartment

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
}
```

Listing 12: An example for the usage of the textual ER2CDS DSL, the graphical example is shown in Fig. 5

cardinalities. Additionally, the `manages` relationship is defined as the first relationship, using the `join order` keyword.

5.2.2 Graphical Concrete Syntax

The graphical concrete syntax of ER2CDS is based on the Chen-Notation of entity relationship diagrams [9]. Entities are represented by their name in the component's header separated by a line from their attributes. Furthermore, key attributes are underlined, and no-out attributes are displayed and crossed out. The graphical syntax in combination with the *tool palette* and the *property palette*, presented in more detail in the next section support the same modeling features as the textual concrete syntax.

The example presented in Fig. 5 illustrates the equivalent graphical representation of the model textually represented by Listing 12. The model features an entity named *Employee*, with the field *PERNR* as key. The attributes *FNAME* and *LNAME* show the use of an alias. In contrast, *DEPARTMENT_ID* of the *Department* entity 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

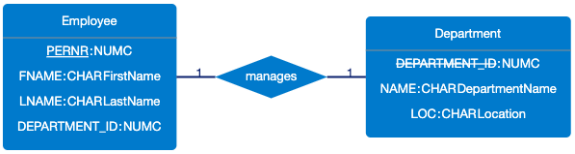

Fig. 5: Illustrative example of the graphical concrete syntax of the BIGER2CDS tool consistent with the textual example shown in Listing 12.

by their names only. 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, for this *Employee* acts as the source entity, which is related to *Department* via the *manages* relationship. The cardinalities are represented as the labels above the relationship. In this case, both are one and can be interpreted as exactly one employee managing one department.

5.3 Model Graphical Editing

We will now focus on graphical modeling interactions since the textual modeling uses the native functionality of VS Code text editing. To implement the CRUD model editing operations, the BIGER2CDS 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 and showing the current state of the selected model element.

To support hybrid graphical and textual modeling, we have to extend LSP. More specifically, additional LSP actions for graphical modeling must be defined. For the textual part, BIGER2CDS is based on the standard LSP actions. Table 1 describes the custom actions defined by BIGER2CDS.

To synchronize the textual and the graphical model representation, we must define a leading

Table 1: Custom LSP actions defined by BIGER2CDS for graphical modeling.

Action Name *[Parameter list]*	**Description**
CreateElement [$elementType : string$]	Used to create a specific element type. The *elementType* parameter contains the information on which kind of element should be created.
CreateEdge [$sourceElementId : string$, $targetElementId : string$]	Used to create an edge from the source element, identified by *sourceElementId*, and the target element, identified by the *targetElementId*. One of the elements has to be an entity, and the other one is a relationship.
CreateAttribute [$elementId : string$]	Creates an attribute for an entity, defined by the *elementId*. This action can only be successfully executed if the *elementId* corresponds to an entity.
CreateJoinClause [$elementId : string$]	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.
UpdateElementProperty [$sourceElementId : string$, $propertyId : string$, $value : string$]	Used to update a property of an element, corresponding to the *elementId*. The property is identified by the *propertyId*. Furthermore, the *value* parameter defines the new value of the property.
DeleteElement [$elementIds : string[]$]	Deletes all elements identified by *elementIds*. It allows to delete multiple elements at the same time.
RequestAutoComplete [$elementId : string$, $type : string$, $search : string$]	Allows to retrieve value suggestions from the backend. The *elementId* identifies the selected element. The *type* defines the type of the autocomplete request. 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. Secondly, it is used 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. The server responds with a SetAutoCompleteAction.
SetAutoComplete [$elementId : string$, $values : AutoCompleteValue[]$]	Sent in response to RequestAutoComplete. The *elementId* identifies the element to which it corresponds. The values are defined as an array of *AutoCompleteValue*, a type that contains a label.
RequestPopupConfirmModel [$elementId : string$, $bounds : Bounds$]	Sent to request a popup from the server. Used to confirm the creation of an entity from the external data source, identified by *elementId*. The bounds correspond to a default LSP type, defining the location of the popup. The server responds with a SetPopupModelAction predefined in LSP.
CreateElementExternal [$elementId : string$]	Is sent after the manual popup confirmation, as a result of RequestPopupConfirmModelAction, to load an entity from the external data source.
RequestMarkers	Sent to validate the model. The server responds with a SetMarkersAction.
SetMarkers [$markers : Marker[]$]	Sent as a response to the RequestMarkersAction. It triggers a validation of the model and contains the validation result in the form of markers. These are defined with a *elementId* to identify the element, *kind*, defining the severity (info, warning, error), and a *description*, a textual explanation.

system that holds the only truth in case of inconsistency. We chose the Langium internal model to act as the single source of truth. Therefore, all updates should be committed to this model. We distinguish between two cases: *i*) updates through the textual representation and *ii*) updates through the graphical representation. While the first case is well supported by Langium natively, the latter case relies on the custom-defined LSP actions for the BIGER2CDS elements' CRUD operators, model validation, and auto-complete.

All custom LSP actions with their respective usage are presented in Table 1. The language server handles the create, update, and delete actions by modifying the current Langium model. It then serializes the model to the file system. The subsequent save triggers the framework to start the same process as modifying the textual representation (see the Langium document lifecycle illustrated in Fig. 2). 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.

5.4 Integration with SAP S/4HANA

An SAP S/4HANA system can be connected to BIGER2CDS. The system needs to expose a custom webservice that can be used to access the data model. This service is implemented as a CDS view entity, which is then exposed as a webservice using the SAP S/4HANA infrastructure. 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 an attribute (see presented suggestions in the property palette of BIGER2CDS in Fig. 6). For example, when modeling 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.

Another use case for connecting to an SAP S/4HANA system is for model validation. Access to an external data model allows the validation of the entities and attributes against it. The ER2CDS webservice 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 with all defined associations.
- **ImportFromClause**: Exposes the *DDCDS_FROMCLAUSE* table with information about which properties the CDS uses in the from clause.

These entity sets are offered by the SAP system and exposed through the custom webservice to provide additional information to BIGER2CDS when generating an ER2CDS model from a connected SAP database. Implementing the webservice 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.

5.4.1 ER2CDS Model-to-Text Transformation

The model-to-text transformation is one of the central features of BIGER2CDS. It allows the transformation of an ER2CDS model into a valid CDS view entity. We will use a template-based approach to generate the textual output. The generation of a CDS view entity can be divided into five steps, which are directly derived from the syntax of CDS.

1. **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.
2. **Generation of header**: The name of the ER2CDS model serves as the name of the generated CDS view entity.
3. **Generation of from clause**: We analyze the model for relationships with no defined association type. Since the join order is

Table 2: Mapping of cardinalities to the respective join/association type.

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]

important, the relationships must be sorted according to the *join order* in the ER2CDS model. Furthermore, the join type is defined based on the specified cardinalities of the relationship.

4. **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. The concrete mapping is illustrated in Table 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.
5. **Generation of attributes**: First, the key attributes are generated with the appropriate *key* keyword of CDS, then the remaining attributes, considering the *no-out* attributes which are not added to the CDS view entity.

The resulting template is illustrated in Listing 13, with <...> denoting markers. Using this template, the generated CDS view entity for the example presented in Listing 12 and Fig. 5 is displayed in Listing 14.

5.4.2 CDS View Import

The import of existing CDS view entities into bigER2CDS is implemented indirectly, using the serialized data of the view. Similarly to the value-help and validation, the webservice defines and exposes a CDS view entity to access the required data. When importing an existing CDS view entity, the data of all associated entities is requested and used to create an ER2CDS model. After converting the SAP representation to an ER2CDS model, the resulting model is serialized to the file system, triggering the creation of the Langium model.

5.5 bigER2CDS Modeling Tool

Fig. 6 shows a screenshot of the LSP-based realization of the bigER2CDS modeling tool in the VS Code extension. The language server is based on Langium. The textual domain-specific language is defined in the Langium grammar, while the graphical modeling part is implemented using Sprotty[10]. It is well integrated with VS Code and also supports LSP.

The user interface of the tool, the *Editor-Panel* webview, serves as the main container for the ER2CDS diagram. It is divided into the *main modeling container* and the *property panel* (see Fig. 6). 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. A custom implementation for all different element types exposes input elements within the property palette.

The *tool palette* is another user interface element that hosts the model editing tools for bigER2CDS, like selection, deletion, validation, and search. It is separated into two parts. 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 enabling the *MarqueeMouseTool* using 'Shift', the latter allows deleting elements using the 'Delete' or 'Backspace' key.
- **DeleteMouseTool** Deletion of an element on mouse click.
- **MarqueeMouseTool** Selection of multiple elements by dragging the mouse.

[10] https://sprotty.org/, last accessed: 17.07.2025

```
@AccessControl.authorizationCheck: #CHECK
@Metadata.ignorePropagatedAnnotations: true
@EndUserText.label: 'Generated by ER2CDS'
define view entity <name> as select from <entity>
([inner join | left outer join | right outer join] <entity> (as <entityAlias>)? on
    ([<entity> | <entityAlias>].<firstAttribute> = [<entity> | <entityAlias>].
    <secondAttribute> (and)?)*)*
([association[1..1] | association[1..*] | association[0..1] | association[0..*]]
    to <entity> (as <entityAlias>)? on ($projection.<firstAttribute> = [<entity> |
     <entityAlias>].<secondAttribute> (and)?)*)*
([composition[1..1] | composition[1..*] | composition[0..1] | composition[0..*]]
    of <entity> (as <entityAlias>)?)*
(association to parent <entity> (as <entityAlias>)? on ($projection.
    <firstAttribute> = [<entity> | <entityAlias>].<secondAttribute> (and)?)*)*
{
        ((key)? [<entity> | <entityAlias>].<attribute> (as <attributeAlias>))*
        (<entity>)*
}
where
[<entity> | <entityAlias>].<attribute> [ = | <> | < | > | <= | >= ] <fixValue>
```

Listing 13: The template used by ER2CDS to generate CDS view entities.

```
@AccessControl.authorizationCheck: #CHECK
@Metadata.ignorePropagatedAnnotations: true
@EndUserText.label: 'Generated by ER2CDS'
define view entity EmployeeManagesDepartment as select
    from Employee
    inner join Department on Employee.DEPARTMENT_ID = Department.DEPARTMENT_ID
{
    key Employee.PERNR,
    Employee.FNAME as FirstName,
    Employee.LNAME as LastName,
    Employee.DEPARTMENT_ID,
    Department.NAME as DepartmentName,
    Department.LOC as Location
}
```

Listing 14: An example of an generated CDS view entity using ERCDS, this CDS view entity has been created from the textual specification in Listing 12 and the graphical specification in Fig. 5

- **Validation** Triggers a validation request to the language server.
- **Search** Allows searching within the tool palette.

The palette further offers the creation tools for all ER2CDS elements like *Add Entity* and *Add Edge*.

Fig. 6 presents a complete overview of the BIGER2CDS modeling tool, integrated into VS Code with the textual and graphical editor, and highlights the value help functionality in the properties palette, where, while the modeler is defining the name of an entity, the tool suggests valid names based on the connected SAP S/4HANA database.

Regarding the deployment and distribution of BIGER2CDS, both VS Code and Business Application Studio (BAS) allow for the installation of

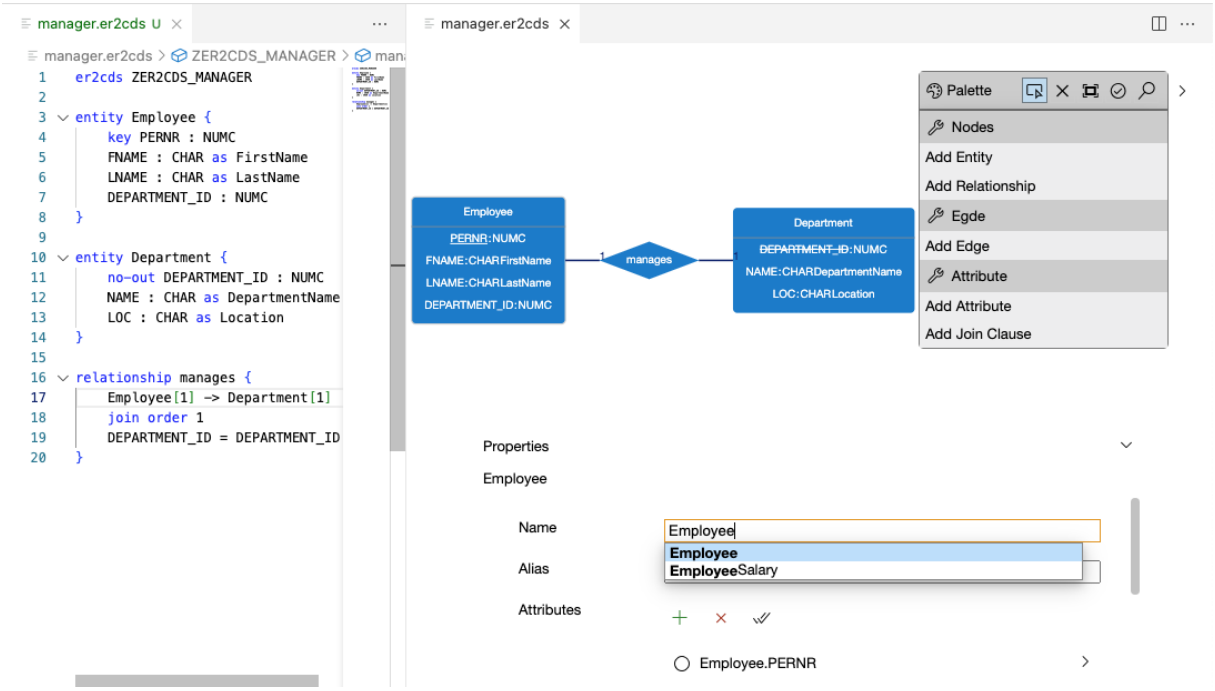

Fig. 6: The BIGER2CDS modeling tool integrated in VS Code.

extensions through the VS Code Marketplace[11], making it the platform of choice for the tool. Users can search, install, and use the extension directly within the development environment[12] without any further dependencies on the local runtime environment.

6 Evaluation

Our evaluation is twofold and composes a controlled experiment—focusing on the expressiveness of the CDS grammar (Req-1 and Req-6), import/export features (Req-2) and the SAP integration (Req-3)—and a case study with practitioners—focusing on the usability of the BIGER2CDS tool (Req-5, Req-7, Req-8, and Req-9). Note that all data belonging to the experiments, including the task descriptions with sample solutions, the imported and exported models can be found in the BIGER2CDS repository[13].

Our evaluation responds to the following research questions.

- **RQ1**: To what extent is BIGER2CDS able to model, import, and export CDS?
- **RQ2**: What is the usability of BIGER2CDS for business experts and developers?

6.1 Experimental Evaluation

The experimental evaluation focused on the expressiveness of the grammar to represent modeled and imported CDS views. We first modeled 20 existing CDS view entities using BIGER2CDS (see Table 3 for descriptive statistics of the dataset). These CDS views were derived from the company for which one of the paper's authors works. He was himself responsible for developing these CDS views using the textual language and the Eclipse ADT tool. The selection was aimed to cover a diversity of CDS view concepts and also a diversity of CDS view sizes, ranging from 0 to 23 relations, three to 128 elements, and 16 to 221 lines of code, respectively. Afterward, we used the model to generate a CDS view entity and compared it to the original textual view description. Furthermore, the resulting output is imported into the SAP S/4HANA system of the original view to

[11] VS Code Marketplace: https://marketplace.visualstudio.com/VSCode
[12] BIGER2CDS tool in the VS Code marketplace: https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.er2cds
[13] BIGER2CDS repository: https://github.com/borkdominik/ER2CDS/tree/main/evaluation

Table 3: Descriptive statistics of the models in our evaluations

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

validate its correctness. The dataset consisted of ten standard CDS view entities created by SAP and ten custom CDS view entities implemented by one co-author working in a company using a SAP S/4HANA system.

All modeled CDS view entities output a syntactically correct CDS view entity. We tested this by importing the generated CDS view code into the running SAP system without receiving any errors. Furthermore, when comparing the generated view to the original view, 20 out of 20 were evaluated positively regarding the correctness of the output itself by the authors of this paper by using basic text comparison techniques. BIGER2CDS can model and generate all the customer-specific view entities and standard view entities.

Next, we evaluated the import of existing CDS view entities into BIGER2CDS. 100 randomly selected standard CDS view entities were imported to BIGER2CDS and validated concerning the tool's capability to i) import and ii) display the CDS view entity (see Table 3 for descriptive statistics of the dataset). Furthermore, the errors and warnings issued by BIGER2CDS were documented.

Regarding the import of existing CDS view entities, BIGER2CDS 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 one out of the 100 CDS view entities. The reason for this was the UNION operator, which is not supported by BIGER2CDS since a UNION of two queries would result in two separate ER2CDS models with the same structure (i.e., the same fields) in the result set.

In a final step, we aimed to contrast the definition of a concrete CDS with BIGER2CDS to its implementation in Eclipse using ADT. Fig. 7 shows the textual specification of CDS in Eclipse ADT while Fig. 8 presents the corresponding CDS implementation using the graphical concrete syntax in BIGER2CDS. While this comparison obviously only shows a single instance, it serves the purpose of illustrating the complexity of specifying valid CDS using the textual syntax only compared to the more intuitively comprehensible representation in a graphical manner (more on this in the case study evaluation). Admittedly, not all users prefer a graphical representation, but having the graphical and the textual representation available in BIGER2CDS in a synchronized manner is clearly an asset as it offers a more accessible way to specify CDS.

Table 4 summarizes the extent to which our implemented BIGER2CDS tool fulfills the individual requirements introduced in Section 4. As can be seen, eight out of nine requirements are completely fulfilled. The one partially fulfilled requirement relates to the lacking ER2CDS language support for the 'UNION' CDS operator. As explained previously, this operator does not make sense in the context of BIGER2CDS as the UNION operator would require establishing cross-references between two CDS views in the tool. An obvious workaround is to create two individual CDS view models in BIGER2CDS, trigger the code generator on both, and then only hand-write the UNION code to connect the two CDS.

6.2 Case Study

For the case study, the participants are presented and asked to model three tasks with increasing difficulty with BIGER2CDS. The description of

```
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Delivery Schedule for Customers'
@Metadata.ignorePropagatedAnnotations: true
define view entity Z_I_MVLCONFIRMED
  as select distinct from I_SalesDocumentItem
    inner join            I_SalesDocument            as _SalesDocument            on I_SalesDocumentItem.SalesDocument = _SalesDocument.SalesDocument
    inner join            I_SalesDocumentPartner     as _SalesDocumentPartner     on  I_SalesDocumentItem.SalesDocument     = _SalesDocumentPartner.SalesDocument
                                                                                  and _SalesDocumentPartner.PartnerFunction = 'AG'

    left outer join       I_SalesDocumentScheduleLine as _SalesDocumentScheduleLine on  I_SalesDocumentItem.SalesDocument                    = _SalesDocumentScheduleLine.SalesDocument
                                                                                  and I_SalesDocumentItem.SalesDocumentItem                = _SalesDocumentScheduleLine.SalesDocumentItem
                                                                                  and _SalesDocumentScheduleLine.IsConfirmedDelivSchedLine = 'X'

    left outer join       I_DeliveryDocumentItem     as _DeliveryDocumentItem     on  I_SalesDocumentItem.SalesDocument     = _DeliveryDocumentItem.ReferenceSDDocument
                                                                                  and I_SalesDocumentItem.SalesDocumentItem = _DeliveryDocumentItem.ReferenceSDDocumentItem

  association [1..1] to I_CalendarDate as _ScheduleDate on _SalesDocumentScheduleLine.ConfirmedDeliveryDate = _ScheduleDate.CalendarDate

{
  key I_SalesDocumentItem.SalesDocument,
  key I_SalesDocumentItem.SalesDocumentItem,
      I_SalesDocumentItem.PurchaseOrderByCustomer,
      I_SalesDocumentItem.ProductConfiguration,

      _SalesDocument.IncotermsClassification,
      _SalesDocument.IncotermsTransferLocation,

      _SalesDocumentPartner.Customer,
      _SalesDocumentPartner.Supplier,

      _SalesDocumentScheduleLine.ConfirmedDeliveryDate,
      _SalesDocumentScheduleLine.OrderQuantityUnit,

      _DeliveryDocumentItem.GoodsMovementStatus,

      _ScheduleDate.CalendarDate,
      _ScheduleDate.CalendarMonth,
      _ScheduleDate.CalendarYear,


      //Associations
      _ScheduleDate._CalendarMonth

}
where
      I_SalesDocumentItem.HigherLevelItem         <> '000000'
  and I_SalesDocumentItem.SalesDocumentRjcnReason =  ''

```

Fig. 7: Example of a CDS view entity using Eclipse ADT, equivalent to the graphical variant in Fig. 8

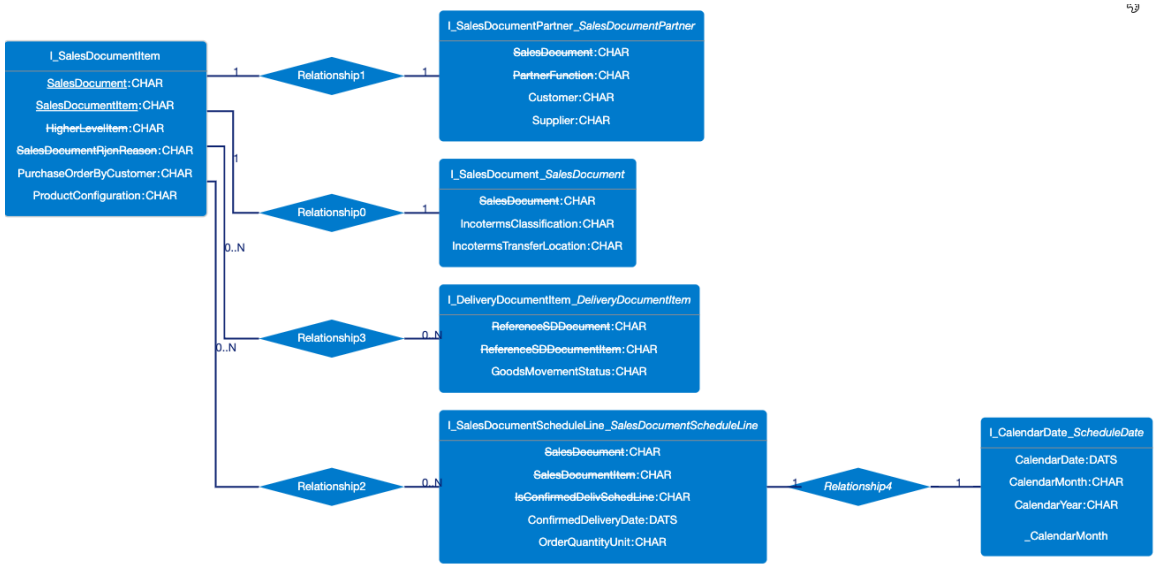

Fig. 8: Example of a CDS view entity using ER2CDS, equivalent to the textual variant in Fig. 7

the three tasks[14], as well as a reference implementation of all three[15] can be found in the BIGER2CDS repository.

After executing the three modeling tasks we surveyed the participants to gather feedback and collect data. The survey can be divided into three parts: six questions regarding the three tasks of the case study, for each task we asked: *i*) if the participant could implement the given task using BIGER2CDS, and *ii*) if the participant could implement the given task using

[14] BIGER2CDS repository case study: https://github.com/borkdominik/ER2CDS/blob/main/evaluation/case-study/case-study.pdf

[15] BIGER2CDS repository case study examples: https://github.com/borkdominik/ER2CDS/tree/main/evaluation/case-study/examples

Table 4: Requirements Fulfillment

Requirement	Fulfillment
Req-1: DSL for CDS	We developed a comprehensive domain-specific language to represent CDS (cf. Fig. 4 and Listing 11).
Req-2: CDS code generation	We realized a template-based model-to-text transformation that uses ER2CDS models as input and automatically generates valid SAP code from it. The evaluation yielded zero errors for the code generator.
Req-3: SAP S/4Hana connection	We realized a webservice that enables the efficient integration between BIGER2CDS and SAP S/4Hana. The tool can query the existing views in the SAP database and can automatically generate the corresponding models.
Req-4: VS Code & BAS integration	We successfully deployed BIGER2CDS as an extension in the VS Code marektplace from which it can be directly installed and used in VS Code and in BAS.
Req-5: Runtime dependencies	No additional runtime dependencies are introduced through BIGER2CDS. The tool can be executed directly within VS Code or BAS.
Req-6: CDS language support	Except for the UNION operator, BIGER2CDS has full CDS language support. The experiments with 100 randomly sampled CDS yielded only a single error, which was related to the use of the UNION operator.
Req-7: CDS CRUD support	We realized full CRUD support for the textual and the graphical concrete syntax. Users can equally edit the model in whatever presentation they prefer.
Req-8: CDS model validation	We realized custom validations for created and imported CDS. Entire CDS models are validated against the CDS grammar while all property values are further validated against a connected SAP S/4Hana system.
Req-9: Editing support	We realized syntax highlighting, code completion, and cross-referencing for the textual concrete syntax. Moreover, we realized the property palette, as well as default, delete, and marquee tools, to support efficient model editing in the graphical view.

the standard textual syntax of CDS. Each question provides possible answers using the Likert scale [18]. Questions of the standard system usability score (SUS) [7] followed, quantifying the usability of BIGER2CDS. The SUS questionnaire is widely adopted with more than 20.000 citations in GScholar. It offers a standardized score for usability that allows a comparative assessment. Subsequently, the participants were asked whether they prefer using BIGER2CDS or the standard textual syntax of CDS in the future. Finally, our survey concluded with three open questions for positive or negative feedback and suggestions for future improvements of BIGER2CDS.

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). All participants were able to create the CDS view entity for the first two tasks using BIGER2CDS, while only five thought they could implement the same tasks using only the textual syntax of CDS. Even fewer participants (two) assessed themselves as being able to implement the third task of the case study using only the textual syntax of CDS. In contrast, seven out of eight participants were able to implement it using BIGER2CDS. Regarding the traditional approach, a lack of programming skills regarding CDS development was the main issue, while when using ER2CDS, the only issue was linked to a lack of knowledge in CDS features (association to parent, composition).

The SUS for each participant ranged from 65 to 100 (out of 100), with an average value of 86.25, a median value of 91.25, and a standard deviation of 12.82. Classifying these scores concerning the studies analyzed in [1], renders the BIGER2CDS tool usability *above-average*.

Finally, it is significant to note that all business experts who participated in the case study prefer to use BIGER2CDS over the textual syntax of CDS in the future. Further, two participating developers favor using BIGER2CDS over their experienced use of the textual syntax.

This finding underpins the increased accessibility and effectiveness of our model-driven approach and the developed tool support compared to the traditional, code-only approach, especially, but not exclusively, for business-oriented users or domain experts lacking a strong programming background.

7 Discussion

The evaluation of BIGER2CDS has shown the effectiveness and usefulness of the tool with real-world examples and real-world CDS developers and business stakeholders. The experiment for creating and importing existing CDS view entities presented the strong capabilities the tool already has. No findings indicated the unfeasibility of such a modeling tool for CDS view entities.

Furthermore, the results of the case support the business need for BIGER2CDS. One of the significant findings is that BIGER2CDS allows developers and business experts to create CDS view entities. Also, the SUS indicates high usability of the tool, making it accessible to a broader audience [45]. Another critical finding is the users' preference to use BIGER2CDS over the textual CDS syntax. Especially for business experts, BIGER2CDS offers a valuable alternative for creating CDS view entities, again indicating the value of such a tool.

The open questions showed rich feedback and further shed light on the fact that developers are also eager to use the tool. Regarding this open feedback, the hybrid modeling of CDS and the higher level of abstraction are denoted as advantages of BIGER2CDS. The suggestions for improvement are related primarily to limitations of the ER2CDS grammar, in more detail, the UNION operator, which is currently not supported, and serve as the basis for further development and future work within the context of low-code business app development and, specifically, BIGER2CDS.

7.1 Implications

Next, we discuss implications for the broader MDE community that can be derived from our research and the lessons we learned.

From an **MDE community perspective**, we showed that it is feasible to realize hybrid modeling support for industrially used languages like CDS. We showed that new technology stacks are capable of realizing more accessible solutions able to bridge the gap between different types of users with varying backgrounds and preferences, and the highly complex and technical domains they are working in, in our case, SAP with the CDS.

By providing a higher level of abstraction, the BIGER2CDS enables business experts who may lack extensive programming skills to participate in the CDS development process, bridging the gap between domain experts and developers. We showed how an academic tool prototype can seamlessly integrate with SAP S/4HANA systems, offering features such as data model validation, value help, and the import of existing CDS view entities.

From a **technical perspective**, this research contributes a novel hybrid modeling tool that is realized on the latest technology stacks. The underlying architectural principles and the means to integrate the widely used frameworks can inform others who are interested in developing hybrid modeling editors [11] with web technologies. BIGER2CDS effectively employs Langium for language server capabilities and Sprotty for graphical rendering, demonstrating the synergies between these frameworks in creating a robust and user-friendly hybrid modeling environment. Langium allows for the creation of domain-specific languages with relative ease, thanks to its comprehensive grammar and parser support. We showed how the built-in functionality of Langium can be extended to support custom LSP actions for model editing, validation, and code generation. Moreover, we showed how Sprotty can be integrated with Langium to seamlessly support hybrid modeling. We further show how such a developed tool can be successfully deployed as an extension to the VS Code Marketplace and integrated into an industrial web application.

Our research further shows how custom domain-specific modeling editors can exploit the rich functionality and services offered by platforms like VS Code. We show how the rich language supporting functionality of VS Code and LSP can be facilitated to efficiently realize a feature-rich, cross-platform, modern web-based hybrid modeling editor.

BIGER2CDS implements a template-based approach for generating CDS view entities from ER models, ensuring the correctness and quality of the generated CDS views. This transformation process is crucial for maintaining consistency and accuracy in the CDS development lifecycle and can be a blueprint for other MDE tool developers.

By releasing BIGER2CDS on the VS Code Marketplace[16] and open source repositories[17], we enable efficient further development, customization, and adoption of our solutions.

From a **domain expert perspective** we have implications based on our thorough evaluation and the controlled experiments. Our research demonstrates that MDE approaches can be developed that are well-received by both developers and business experts for supporting highly technical and complex tasks like the development of CDS. This should raise interest in exploring more means to equip (or even replace) highly technical industrial solutions with MDE approaches. A need that was only recently raised in an industrial modeling survey [20].

7.2 Challenges

The realization of BIGER2CDS was not free from challenges. To enable the community to learn from our research, we share our reflections on the challenges and limitations we faced in the following.

Managing the complexity of hybrid graphical and textual representations in ER2CDS was a significant hurdle, particularly in ensuring consistency and accuracy across model transformations. The decision to use the Langium model as the single truth and transforming graphical model edits into the textual representation, which then triggers the standardized and mature Langium document builder lifecycle, has proved useful in our context. Designing an intuitive yet powerful UI that caters to both novice and expert users proved challenging, often resulting in a trade-off between accessibility and functionality. Future work can clearly further improve the UI-end of BIGER2CDS.

Next to the conceptual challenges described before, we also faced limitations in the tools and frameworks used. BIGER2CDS is not fully integrated with SAP Business Application Studio (BAS), although integration is feasible. We decided to build a webservice as a wrapper for the tight integration, which would require more development. Despite its intended accessibility, BIGER2CDS still demands some technical expertise, restricting its use for entirely non-technical stakeholders. Especially when aiming for creating complex CDS views, business stakeholders might require further assistance, e.g., in the sense of an intelligent modeling assistant [23].

The simplified approach of hybrid textual and graphical modeling sometimes overlooks deeper complexities present in CDS view entities. The hybrid modeling of CDS views highlighted a significant challenge in maintaining synchronization between these representations. The hybrid approach sometimes led to inconsistencies when models became overly complex.

From a wider adoption perspective, we see broader barriers primarily in the sometimes insufficient documentation and inadequate beginner resources for the used technologies. As e.g., Langium and Sprotty are under continuous development and maturation, we believe this barrier will be removed soon. A final challenge we face is the need to actively involve domain experts throughout the development, as their early feedback can be crucial for the eventual usability and intention to use of the tool. What helped here is that one of the co-authors was working at the company that used SAP and implemented custom views by themselves. Consequently, it was unusually easy to efficiently receive feedback and guidance throughout the requirements engineering, conceptualization, and implementation phases.

7.3 Threats to Validity

This research is not exempt from limitations and threats to validity [53]. From an internal and concept validity point of view, we made sure that all experiments were conducted in a controlled and equal environment. Participation was voluntary, and there is no reliability relationship between the participants and the researchers. Notably, one of the authors of this paper is a co-worker of

[16] BIGER2CDS VS Code extension https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.er2cds

[17] BIGER2CDS Github repository https://github.com/borkdominik/ER2CDS

the company in which we conducted the empirical evaluation. We did not measure the time and did not make a comparative efficiency evaluation of our tool; these are part of our future research agenda. Notably, the effectiveness is highly subjective and based on the expertise, not only the experience, of the participants. We did not measure the expertise of the individual participants in our questionnaire.

A particular threat we could not mitigate is the limited sample size, which hampers the conclusion validity and the external validity. We tried to mitigate this by using three representative case study tasks with a diverging complexity and by randomly sampling 100 existing CDS models for testing the importing functionality and the language expressiveness of BIGER2CDS. Still, we cannot generalize from these insights, but we believe having eight real practitioners and a number of real CDS models for testing our approach yields very promising and reliable insights. Other modeling cases conducted with other participants might lead to different observations and findings. Part of our future research, and based on the open release of the modeling tool, will focus on gathering more feedback.

The testing protocol itself is also not exempt from threats to validity. We used a standardized system usability questionnaire. A different questionnaire might lead to different conclusions. Moreover, we can state that the questionnaire was intentionally compact as we wanted to collect responses to all questions and did not want to risk that participants stop the survey before concluding. More specific questionnaires testing, e.g., the perceived usefulness, the ease of use, and the intention of use, will lead to even richer insights.

Concluding the threats to validity, we can state that the presented results are limited to the scope defined by the empirical evaluation. Considering the fact that we used standardized questionnaires, real domain experts, and real CDS models, we believe our implications hold and that this research can make meaningful contributions, especially, but not limited to, researchers interested in developing novel web-based MDE tools and practitioners in the context of SAP Core Data Services.

8 Conclusion

This paper contributes a novel model-driven engineering approach for SAP Core Data Services (CDS) view entities. We introduced a domain-specific language, the ER2CDS modeling language, for hybrid textual and graphical modeling of CDS view entities. The novel BIGER2CDS tool presented in this paper supports the hybrid modeling of CDS views and the model-driven generation of their implementation in SAP. Furthermore, its import feature enables the integration of the tool with running SAP systems and the import of existing CDS view entities.

Our multi-faceted evaluation proved the effectiveness of the tool in modeling, importing, and generating CDS view entities. An empirical user study further showed that business users and CDS developers felt comfortable using the tool, many of whom even reported that they would like to use the tool in their professional work. This underpins our assumption that a model-driven approach with a hybrid modeling tool fosters accessibility and eases the development of SAP S/4Hana CDS views.

In our future work, we aim to extend the empirical evaluation of our tool to gain further experience reports. The current version of BIGER2CDS is publicly accessible in the VS Code marketplace[12]. Furthermore, the tool is available open source via[13]. In a future extension of the approach, a modeling guide or modeling assistant using artificial intelligence [2, 12, 23, 47] is aimed at further increasing users' productivity. All of these would improve the model-driven engineering process for CDS and, furthermore, the low-code development of business applications [8].

References

[1] Bangor A, Kortum PT, Miller JT (2008) An empirical evaluation of the system usability scale. Intl Journal of Human–Computer Interaction 24(6):574–594

[2] Bork D (2021) Conceptual modeling and artificial intelligence: Challenges and opportunities for enterprise engineering - keynote presentation at the 11th enterprise engineering working conference (EEWC 2021). In: Aveiro D, Proper HA, Guerreiro S, et al (eds)

Advances in Enterprise Engineering XV - 11th Enterprise Engineering Working Conference, EEWC 2021, Revised Selected Papers, Lecture Notes in Business Information Processing, vol 441. Springer, pp 3–9, https://doi.org/10.1007/978-3-031-11520-2_1

[3] Bork D, Langer P (2023) 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. https://doi.org/10.18417/EMISA.18.9

[4] Bork D, Karagiannis D, Pittl B (2020) A survey of modeling language specification techniques. Inf Syst 87. https://doi.org/10.1016/J.IS.2019.101425

[5] Bork D, Langer P, Ortmayr T (2023) A vision for flexible glsp-based web modeling tools. In: Almeida JPA, Kaczmarek-Heß M, Koschmider A, et al (eds) The Practice of Enterprise Modeling - 16th IFIP Working Conference, PoEM 2023, Vienna, Austria, November 28 - December 1, 2023, Proceedings, Lecture Notes in Business Information Processing, vol 497. Springer, pp 109–124, https://doi.org/10.1007/978-3-031-48583-1_7

[6] Braghin C, Lilli M, Riccobene E, et al (2024) Kant: A domain-specific language for modeling security protocols. In: Mayo FJD, Pires LF, Seidewitz E (eds) Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering, MODELSWARD 2024, Rome, Italy, February 21-23, 2024. SCITEPRESS, pp 62–73

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

[8] Bucaioni A, Cicchetti A, Ciccozzi F (2022) Modelling in low-code development: a multi-vocal systematic review. Softw Syst Model 21(5):1959–1981. https://doi.org/10.1007/S10270-021-00964-0

[9] Chen PP (1976) The entity-relationship model - toward a unified view of data. ACM Trans Database Syst 1(1):9–36. https://doi.org/10.1145/320434.320440

[10] Ciccozzi F, Tichy M, Vangheluwe H, et al (2019) Blended modelling - what, why and how. In: Burgueño L, Pretschner A, Voss S, et al (eds) 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15-20, 2019. IEEE, pp 425–430, https://doi.org/10.1109/MODELS-C.2019.00068

[11] David I, Latifaj M, Pietron J, et al (2023) Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. Softw Syst Model 22(1):415–447. https://doi.org/10.1007/S10270-022-01010-3

[12] Garmendia A, Bork D, Eisenberg M, et al (2023) Leveraging artificial intelligence for model-based software analysis and design. In: Romero JR, Medina-Bulo I, Chicano F (eds) Optimising the Software Development Process with Artificial Intelligence. Natural Computing Series, Springer, p 93–117, https://doi.org/10.1007/978-981-19-9948-2_4

[13] Giner-Miguelez J, Gómez A, Cabot J (2022) Describeml: a tool for describing machine learning datasets. In: Kühn T, Sousa V (eds) Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS 2022, Montreal, Quebec, Canada, October 23-28, 2022. ACM, pp 22–26, https://doi.org/10.1145/3550356.3559087

[14] Glaser P, Bork D (2021) The biger tool - hybrid textual and graphical modeling of entity relationships in VS code. In: 25th International Enterprise Distributed Object Computing Workshop, EDOC Workshop 2021, Gold Coast, Australia, October 25-29, 2021. IEEE, pp 337–340, https://doi.org/10.1109/EDOCW52865.2021.00066

[15] Glaser P, Hammerschmied G, Hnatiuk V, et al (2022) The biger modeling tool. In: Link S, Reinhartz-Berger I, Zdravkovic J, et al

(eds) 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, CEUR Workshop Proceedings, vol 3211. CEUR-WS.org, URL https://ceur-ws.org/Vol-3211/CR_120.pdf

[16] Glaser PL, Bork D (2021) 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), pp 337–340, https://doi.org/10.1109/EDOCW52865.2021.00066

[17] Hutchinson J, Rouncefield M, Whittle J (2011) Model-driven engineering practices in industry. In: Proceedings of the 33rd International Conference on Software Engineering, pp 633–642

[18] Joshi A, Kale S, Chandel S, et al (2015) Likert scale: Explored and explained. British journal of applied science & technology 7(4):396–403

[19] Keller H (2015) 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

[20] Michael J, Bork D, Wimmer M, et al (2024) Quo vadis modeling? Softw Syst Model 23(1):7–28. https://doi.org/10.1007/S10270-023-01128-Y

[21] Microsoft (2024) Language server protocol specification. https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/, accessed: 2024-5-5

[22] Microsoft (2024) What is the language server protocol? https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/, accessed: 2024-6-7

[23] Mussbacher G, Combemale B, Kienzle J, et al (2020) Opportunities in intelligent modeling assistance. Softw Syst Model 19(5):1045–1053. https://doi.org/10.1007/S10270-020-00814-5

[24] O'Regan G (2015) Pillars of computing. Springer

[25] Petzold J (2022) A textual domain specific language for system-theoretic process analysis. PhD thesis, Kiel University

[26] Petzold J, Kreiß J, von Hanxleden R (2023) 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. IEEE, pp 559–567, https://doi.org/10.1109/DSN58367.2023.00058, URL https://doi.org/10.1109/DSN58367.2023.00058

[27] Popov G, Lu J, Vishnyakov V (2023) Toward extensible low-code development platforms. In: International Conference on Innovation of Emerging Information and Communication Technology, Springer, pp 487–497

[28] Rodríguez-Echeverría R, Izquierdo JLC, Wimmer M, et al (2018) Towards a language server protocol infrastructure for graphical modeling. In: Wasowski A, Paige RF, Haugen Ø (eds) Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018. ACM, pp 370–380, https://doi.org/10.1145/3239372.3239383

[29] SAP SE (2020) 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

[30] SAP SE (2020) Developing apps with sap fiori elements. https://sapui5.hana.ondemand.com/sdk/#/topic/03265b0408e2432c9571d6b3feb6b1fd, accessed: 2024-5-5

[31] SAP SE (2024) 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

[32] SAP SE (2024) Cds ddl - cds view entity, association. https://help.sap.com/doc/aba

pdocu_cp_index_htm/CLOUD/en-US/index.htm?file=abencds_simple_association_v2.htm, accessed: 2024-6-14

[33] SAP SE (2024) 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

[34] SAP SE (2024) 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

[35] SAP SE (2024) 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

[36] SAP SE (2024) 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

[37] SAP SE (2024) 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

[38] SAP SE (2024) 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

[39] SAP SE (2024) 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

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

[41] SAP SE (2024) Sap announces q4 and fy 2023. https://news.sap.com/2024/01/sap-announces-q4-and-fy-2023-results/, accessed: 2024-6-9

[42] SAP SE (2024) Sap s/4hana 2023. https://help.sap.com/doc/e2048712f0ab45e791e6d15ba5e20c68/2023/en-US/FSD_OP2023_latest.pdf, accessed: 2024-6-9

[43] SAP SE (2024) Vdm layers and view types. https://help.sap.com/docs/SAP_S4HANA_ON-PREMISE/ee6ff9b281d8448f96b4fe6c89f2bdc8/0a875bc7a005465aad92c08becc11776.html, accessed: 2024-6-14

[44] Sarferaz S (2023) Virtuelles Datenmodell, Springer Fachmedien Wiesbaden, Wiesbaden, pp 349–360. https://doi.org/10.1007/978-3-658-40499-4_20, URL https://doi.org/10.1007/978-3-658-40499-4_20

[45] Sarioglu A, Metin H, Bork D (2025) Accessibility in conceptual modeling - A systematic literature review, a keyboard-only UML modeling tool, and a research roadmap. Data Knowl Eng 158:102423. https://doi.org/10.1016/J.DATAK.2025.102423

[46] Schulz O (2016) Der SAP-Grundkurs für Einsteiger und Anwender. Rheinwerk Verlag

[47] Shilov N, Othman W, Fellmann M, et al (2023) Machine learning for enterprise modeling assistance: an investigation of the potential and proof of concept. Softw Syst Model 22(2):619–646. https://doi.org/10.1007/S10270-022-01077-Y

[48] Statista (2024) Top ERP software market share by company 2023. https://www.statista.com/statistics/249637/erp-software-market-share-by-company/, accessed: 2024-5-5

[49] TypeFox (2025) Document lifecycle. https://langium.org/docs/reference/document-lifecycle/, accessed: 2025-07-17

[50] TypeFox (2025) Features. https://langium.org/docs/features/, accessed: 2025-07-17

[51] TypeFox (2025) Grammar language. https://langium.org/docs/reference/grammar-language/, accessed: 2025-07-17

[52] TypeFox GmbH (2024) Langium. https://langium.org, accessed: 01.02.2023

[53] Wohlin C, Runeson P, Höst M, et al (2012) Experimentation in software engineering. Springer Science & Business Media

Appendix A Complete Langium Grammar Definition

```
grammar ER2CDS

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

Entity:
        (type=EntityType)? 'entity' name=ID '{'
                ('alias' alias=ID)?
                (attributes+=Attribute)*
                ('expose' (associations+=Association)*)?
                ('where' (whereClauses+=EntityWhereClause)*)?
        '}';

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

Association:
        name=ID ('as' alias=ID)?;

EntityWhereClause:
        (attribute=[Attribute:ID] comparison=ComparisonType fixValue=FixValueType);

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

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

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

DataType:
        type=ID;

type EntityType = 'no-expose';
EntityType returns EntityType:
        NO_EXPOSE
;
NO_EXPOSE returns string:
        'no-expose';

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';

type JoinOrderType = number;
JoinOrderType returns number:
        INT | ONE
;

type FixValueType = string | number;
FixValueType returns FixValueType:
        CHAR | INT | ONE
;

type ComparisonType = '=' | '<>' | '<' | '>' | '<=' | '>=';
ComparisonType returns ComparisonType:
        EQUAL | NOT_EQUAL | LOWER_THAN | GREATER_THAN | LOWER_EQUAL | GREATER_EQUAL
;
EQUAL returns string:
        '=';
NOT_EQUAL returns string:
        '<>';
LOWER_THAN returns string:
        '<';
GREATER_THAN returns string:
        '>';
LOWER_EQUAL returns string:
        '<=';
GREATER_EQUAL returns string:
        '>=';

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

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


Listing 15: ER2CDS DSL grammar implemented using the Langium grammar.