22

STEP Technology for ECAD Databases

Frank Buijs*
Cadlab
Bahnhofstr. 32
33102 Paderborn
Germany

Wolfgang Käfer†
Daimler-Benz F3P
Postfach 2360
89013 Ulm
Germany

Topics: *Common Databases, Standardization Activities, Exchange Formats*

1 Introduction

Standards are increasingly important for ECAD frameworks. The emerging international standard for the exchange of product model data (CAD data, CAM data, ...) is ISO 10303 "Industrial automation systems and integration - Product data representation and exchange", better known as STEP (STandard for Exchange of Product data). This paper addresses the STEP technology for ECAD databases.

STEP provides standardized data schemas, called *Application Protocols (APs)*, for a number of application domains. E.g., AP210 is a standardized data schema for "Printed circuit assembly product design data". For defining the data schemas, STEP provides the information modeling language EXPRESS. Thus, the APs are annotated EXPRESS schemas.

Part 21 of STEP defines how an ASCII file format is derived from an EXPRESS schema. Thus, an AP together with Part 21 defines a standardized file format for data exchange. E.g., AP210 together with Part 21 provides a functionality similar to EDIF-PCB.

Part 22 of STEP defines the database interface SDAI (Standard Data Access Interface), which provides a standardized access to data that conform to an AP. E.g., AP210 together with Part 22 defines a standardized database interface to PCB data. In general, applications can be implemented on the basis of the SDAI as soon as an EXPRESS schema of the data is available.

Therefore, we believe that SDAI will also be the leading standard interface for ECAD

*E-mail: Buijs@Cadlab.DE
†E-mail: Wolfgang.Kaefer@dbag.ulm.DaimlerBenz.COM

F. J. Rammig et al. (eds.), *Electronic Design Automation Frameworks*

databases. However, the definition of the SDAI is still under development and is not yet published by the ISO. The goal of this paper is to present the basic concepts and features of the SDAI. The new features that will appear in the new SDAI version are also described.

Section 2 provides some STEP background. The subsequent sections present the SDAI concepts. Section 5 gives an example of an SDAI session. Section 6 presents the features that were added to the SDAI recently.

2 STEP Background

For an overview of STEP we refer to [1, 2, 6]. The goal of this section is only to give some background information before explaining the SDAI concepts.

STEP is organized in two main blocks. The first block of documents, i.e., Parts 1 - 10 (fundamentals), Parts 11 - 20 (description methods), Parts 21 - 30 (implementation methods), and Parts 31 - 40 (conformance testing), are not associated with any specific application domain. They describe generic methods and tools which are applicable to any kind of information modeling task. For the scope of this paper Part 11 [3], the EXPRESS Language Reference Manual, and Part 22 [5], the SDAI specification, are of particular interest and discussed below.

The second block of documents employ the methods described in the first block to more and more application specific environments. Parts 200 - 1199 define the Application Protocols, like Part 210 (AP210 "Printed circuit assembly product design data"). and Part 214 (AP214 "Core data for automotive mechanical design processes"). The APs use subschemas, called *resources*. Parts 41 - 100 of STEP define the integrated resources, application-independent subschemas for kinds of data that are frequently used. They are used by several APs. For example, the subschema "Product structure configuration" (Part 44) is used by both AP210 and AP214. Parts 100 - 199 define the application resources, subschemas that are more application specific. Parts 1200 - 2199 define the abstract test suites corresponding to the APs 200 - 1199.

2.1 EXPRESS

Part 11 of STEP defines the information modeling language EXPRESS [3]. With EXPRESS, and its graphical representation EXPRESS-G, data *schemas* can be defined.

An EXPRESS schema basically defines a set of entities. An entity is defined by an identifier and a set of attributes. The data type of an attribute may be a simple type (number, real, integer, binary, boolean, logical, string), an enumeration type, a select type, or an entity type. Aggregate data types (set, bag, list, array) are also allowed. The entity type attributes are used for defining the relationships between the entities.

EXPRESS also allows for the definition of *rules* in order to state consistency conditions or restrictions on the values of attributes.

For example, Figure 1 shows a simple EXPRESS schema with two entities. The attribute `members` of `port_group` defines an $1 : n$ relationship between the entities `port_group` and `port`. The uniqueness rule `UR1` defines that the names of each instance of the entity `port` are unique.

```
SCHEMA example;

ENTITY port
     name :  STRING;
UNIQUE
     UR1:  name;
END_ENTITY;

ENTITY port_group
     members :  SET OF port;
END_ENTITY;

END_SCHEMA;
```

Figure 1: Sample EXPRESS schema

Furthermore, EXPRESS supports inheritance. Entities inherit attributes from other entities that are defined as their supertypes.

2.2 Implementation Methods

For the exchange of *entity instances*, two implementation methods are defined. Part 21 of STEP defines the format of STEP Physical Files [4]. A STEP file contains mainly a list of entity instances. Each entity instance is defined by a unique identifier (an integer preceded by a '#'), an entity identifier of the corresponding EXPRESS schema, and a list with the values of the attributes. For example, the entity instances listed in Figure 2 correspond to the EXPRESS schema in Figure 1.

```
#1=port("port1");
#2=port_group((#1,#3));
#3=port("port2");
```

Figure 2: Sample STEP Physical File

Part 22 of STEP defines the database interface SDAI [5]. The functionality of the SDAI is independent of a particular programming language. For implementing the SDAI, Parts 23 - 25 of STEP define *language bindings* (C++, C, and FORTRAN, respectively).

A language binding can be either a *late binding* or an *early binding*. A late binding is independent of a particular EXPRESS schema, an early binding is not. For example, in

order to give an instance p1 of the entity port the name port1, a late binding provides the generic operation PutAttribute(p1,name,"port1"), an early binding provides the schema-specific operation PutAttributePortName(p1,"port1"). In the case of a late binding, the parameters like name are typically evaluated at run-time. In the case of an early binding, such parameters are typically evaluated at application program compile-time. Therefore, an early binding SDAI implementation performs better than a late binding SDAI implementation.

3 Data Organization of the SDAI

The SDAI is independent of the underlying data storage technology, i.e., the data may be stored in a relational database, an object-oriented database, a file system, or a combination of them. The generic name for the data stores is *repository*. A repository may be a single database or a collection of physical files. Thus, from the physical point of view, the entity instances are grouped in repositories.

Apart from the physical view, the SDAI also distinguishes a logical view. Logically, the entity instances are grouped in *schema instances*. Each schema instance has an underlying EXPRESS schema. All entity instances in a schema instance conform to this underlying schema. Schema instances are created, modified, and deleted by the application. Several schema instances with the same underlying schema are allowed.

The schema instance is the domain for referencing. For example, in the case of the three entity instances in Figure 2, entity instance #2 references to the entity instances #1 and #3. This referencing is only allowed if #1 and #3 are in the same schema instance as #2. The schema instance is also the domain for the validation of rules. For example, suppose a schema instance has the schema in Figure 1 as underlying schema. Uniqueness rule UR1 defines that each instance of the entity port must have a unique name. A validation of this uniqueness rule tests if all instances of port in the schema instance have unique names. Note, that the domain for referencing and rule validation is a schema instance and not a repository, i.e. in our example, a validation of rule UR1 does not test if all instances of port in the repository have unique names.

Within the repositories and the schema instances, the entity instances are grouped in *sdai models*. An sdai model may contain some grouping of data which has a meaning as a whole, such as a particular part description along with its geometry. A schema instance can be distributed across several repositories, but all the entity instances in an sdai model are allocated in *one* repository. Thus, each entity instance is in one sdai model, and each sdai model is in one repository.

Figure 3 illustrates the notions of repository, schema instance, and sdai model. This figure also illustrates that an entity instance in one sdai model may refer to an entity instance in another sdai model, even if the sdai models are in different repositories — provided the sdai models belong to the same schema instance.

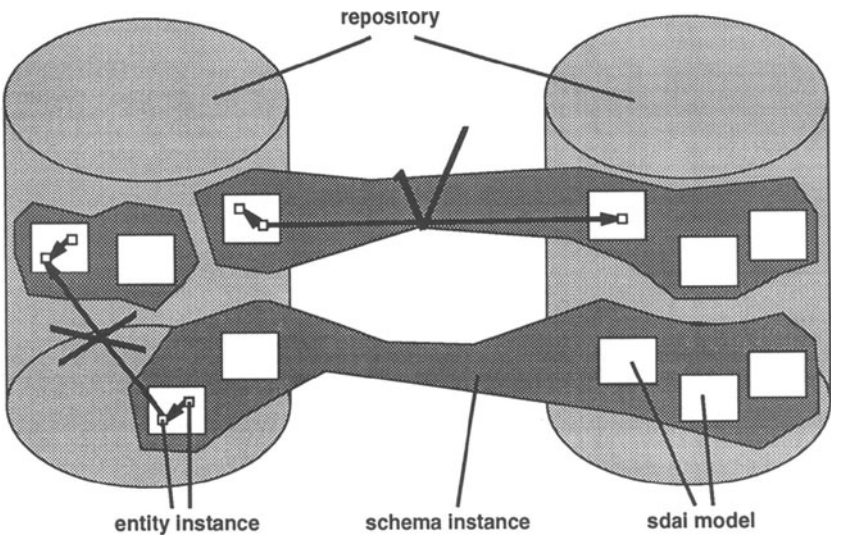

Figure 3: Logical and physical data organization

Apart from the sdai models containing application data, the SDAI also has some special sdai models containing the so-called *meta data*. These sdai models contain the session data, such as the error log and information on the data organization, and the dictionary data. The dictionary data describe the EXPRESS schemas supported by the SDAI implementation. In the case of a late binding SDAI implementation, the dictionary data are used for mapping the SDAI operations onto database operations. Since the meta data are stored in sdai models, just like the application data, the meta data can be accessed using the same SDAI operations as used for retrieving application data. The applications have read-only access to the meta data.

4 SDAI Operations

The SDAI provides a large number of operations to access the data. This section gives an introduction to the SDAI operations.

The *schema instance operations* create and delete schema instances, and validate rules. E.g., the operation `ValidateUniquenessRule(SI1,port,UR1)` tests if all instances of `port` in schema instance `SI1` have unique names. Note, that validations are not performed automatically by the SDAI, but only on demand of the application. Even if an application adds an element to a set, the SDAI does not test whether this element is already a member of the set. The SDAI merely provides an operation that tests whether the members of a

set are unique.

The *sdai model operations* create and delete sdai models, and initiate the access to an sdai model. E.g., the operation `StartReadWriteAccessMode(M1)` opens the sdai model `M1` for read-write access.

The *entity instance operations* are used for creating, copying, modifying, and deleting entity instances. Typical entity instance operations are CreateEntityInstance, PutAttribute, GetAttribute, etc.

For the handling of aggregates, like sets and lists, iterators are used. An *iterator* is a mechanism that allows for traversing the contents of an instance of an aggregate. E.g., in the case of an instance of the entity `port_group`, the operation CreateIterator can be applied to its attribute `members` to obtain an iterator on this aggregate instance. Subsequent operations such as Next, GetCurrentMember, and RemoveCurrentMember are applied to this iterator for traversing and accessing the aggregate instance. An application may use multiple iterators on an aggregate instance. However, if the aggregate instance is modified via one iterator, the use of the other iterators may produce undefined results afterwards.

Further SDAI operations are presented in Section 6.

5 An Example Session

Each SDAI operation belongs to a session. A *session* is the set of operations and implementation-specific activities that occur when one application uses an SDAI implementation for an unbroken period of time. Each session is initiated by an OpenSession operation and terminated by a CloseSession operation.

Figure 4 shows the first part of an example SDAI session. First the session is initiated, repository `R1` is opened, and sdai model `M1` in this repository is opened. `M1` is part of a schema instance that has the schema `example` in Figure 1 as underlying schema. Subsequently, the entity instances listed in Figure 2 are entered in the sdai model `M1`. After that, some operations are performed to find out the name of a port in the port_group. Since the attribute `members` of `port_group` is an aggregate (`SET OF port`), an iterator is used for this purpose.

6 New Features

The definition of the SDAI is still under development. In 1993 an ISO Committee Draft was sent out for ballot. After the ballot, a number of new features were added to the SDAI in order to meet the user requirements. The new SDAI version provides a basic transaction mechanism, supports queries, and enables data sharing.

```
Initiate session, open repository R1 and sdai model M1
session <- OpenSession()
OpenRepository(session,R1)
StartReadWriteAccessMode(M1)
Enter the entity instances listed in Figure 2
p1 <- CreateEntityInstance(port,M1)
PutAttribute(p1,name,"port1")
p2 <- CreateEntityInstance(port,M1)
PutAttribute(p2,name,"port2")
pg1 <- CreateEntityInstance(port_group,M1)
m1 <- CreateAggregateInstance(pg1,members)
Add(m1,p1)
Add(m1,p2)
Find out the name of a port in the port_group pg1
m1 <- GetAttribute(pg1,members)
i1 <- CreateIterator(m1)
Next(i1)
p1 <- GetCurrentMember(i1)
GetAttribute(p1,name) returns the name of the port
```

Figure 4: Sample SDAI session

6.1 Transactions

The SDAI transaction concept is designed to support an application programmer performing complex data transformations on highly interconnected data. As such, an SDAI *transaction* is defined as a sequence of operations working on entity instances. Which entity instances are available for inspection or modification is defined by the session in which the transaction is started.

The SDAI provides four transaction operations: StartTransaction, Abort, Commit, and EndTransaction. The StartTransaction operation initiates a transaction either in read-write mode or in read-only mode. After starting the transaction, entity instances might be accessed or modified. The Abort operation discards all changes which happened since the last StartTransaction operation or Commit operation. The Commit operation makes all changes persistent which happened since the last StartTransaction operation, Abort operation, or Commit operation. Neither the Commit operation nor the Abort operation terminates a transaction. Only the EndTransaction operation ends the sequence of operations started with the StartTransaction operation. The EndTransaction operation comes in two flavors: either EndTransaction with commit or EndTransaction with abort.

If an SDAI implementation supports transactions, all the operations on entity instances can only be performed within a transaction. Figure 5 shows the sample SDAI session of Figure 4 with transaction operations. In fact, the SDAI specification defines three levels of SDAI implementations: no transactions supported (level 1), transactions supported at sdai-model level (level 2), and full transaction support (level 3). In the case of

```
Initiate session, open repository R1 and sdai model M1
session <- OpenSession()
OpenRepository(session,R1)
StartReadWriteAccessMode(M1)
Enter the entity instances listed in Figure 2
t_id <- StartTransactionReadWriteAccessMode(session)
...
EndTransactionAccessAndCommit(t_id)
Find out the name of a port in the port_group pg1
t_id <- StartTransactionReadOnlyAccessMode(session)
...
```

Figure 5: Sample SDAI session with transaction operations

level 3, the operations StartTransaction, Abort, Commit, and EndTransaction are provided. In the case of level 2, the operations SaveChanges and UndoChanges are provided. In this case, `StartReadWriteAccessMode(M1)` starts a transaction on the sdai model `M1`, and the operations `UndoChanges(M1)` and `SaveChanges(M1)` can be used to abort, and commit, respectively. The transaction on the sdai model `M1` is ended automatically by `EndReadWriteAccessMode(M1)`.

6.2 Query Support

The query operation of the SDAI queries a set of entity instances and returns those entity instances meeting the specified criteria. For example, `SdaiQuery(s,"('b' <= port.name) AND (port.name < 'c')",l)` queries the set `s` and adds all the instances of `port` whose name starts with 'b' to the list `l`.

All the members of the set `s` must be instances of the entity `port` or one of the subtypes of this entity. For example, `s` might be the set of all the instances of `port` in some sdai model. For each sdai model and each entity of the underlying schema, the SDAI provides a variable, called *folder*, that represents the set of all the instances of the entity in the sdai model.

The operation SdaiQuery does not create a list of the entity instances meeting the criteria, but adds those entity instances to an existing non-persistent list. This list can be used for further SdaiQuery operations (either as set that is queried or as list to which the entity instances meeting the criteria are added) and other SDAI operations.

6.3 Data Sharing

Furthermore, the SDAI now allows that schema instances share data. I.e., an sdai model can be in more than one schema instance. Figure 6 illustrates a situation where schema instances share an sdai model.

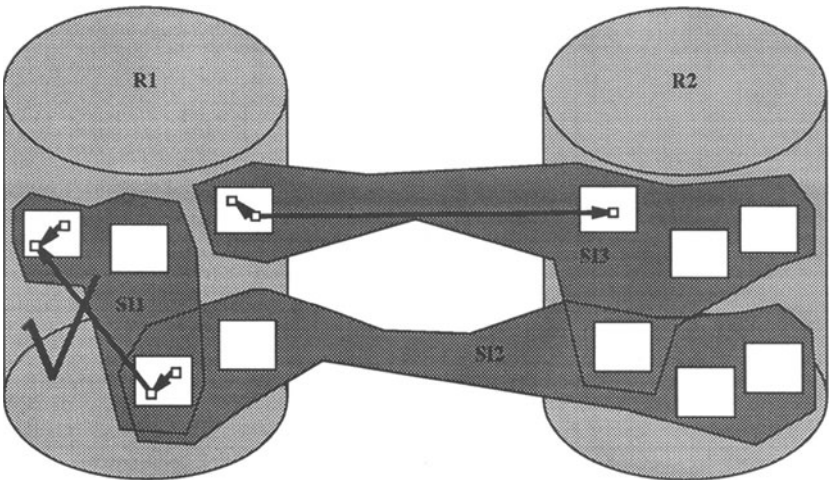

Figure 6: Data sharing

Schema instances can also share an sdai model if the schema instances have different underlying schemas. For example, suppose schema instance `SI1` has AP210 as underlying schema and schema instance `SI2` has AP214 as underlying schema. If the product described in `SI1` and the product described in `SI2` have the same product structure, they can share the sdai model describing the product structure.

Data sharing between schema instances with different underlying schemas is only possible if the shared data conform to a common subset of the underlying schemas. Such common subsets are defined by data sharing tables. For example, a data sharing table may define that the subschema "Product structure configuration", which is used by both AP210 and AP214, is a common subset of AP210 and AP214.

Note, that data sharing may introduce references from one schema instance into another. The reference indicated in Figure 6 is legal, since it is a reference inside schema instance `SI1`. However, at the same time, it is a reference from schema instance `SI2` into `SI1`. In this situation any operation that is applied to schema instance `SI2`, like a rule validation, treats this reference as a NULL reference.

7 Conclusion

This paper presented the Standard Data Access Interface SDAI. The SDAI provides a standardized database interface to all kinds of CAx data. We believe that SDAI will also be the leading standard interface for ECAD databases.

The definition of the SDAI is now on its way to become an ISO standard. It will be a Draft International Standard (ISO/DIS 10303-22) in the first half of 1995. For achieving this status, several features had to be added or improved to the earlier SDAI version (the ISO Committee Draft sent out for ballot in 1993). A proper distinction between the physical data organization (repositories) and the logical data organization (schema instances) was introduced and data sharing was enabled, a basic transaction mechanism was added, the query operation was added, and many other improvements were made. Nevertheless, some aspects are still under discussion.

Most people involved in STEP agree that data sharing between applications using different APs should be possible. Therefore, the SDAI enables data sharing. Common subsets of APs are defined by data sharing tables (cf. Section 6.3). However, within the STEP community, there is no common agreement on how the common subsets of different APs should be described. The *Application Interpreted Construct (AIC)* was introduced for this purpose, but currently the AICs are still under development. Consequently, the actual SDAI version does not describe how the data sharing tables are populated.

Furthermore, although the meta data in the actual SDAI version provide all the necessary information, there is an ongoing discussion on which information from the EXPRESS schemas should be stored how.

References

[1] R. Anderl, "STEP - Grundlagen der Produktmodelltechnologie", in: W. Stucky, A. Oberweis (eds.), in Proc. der *Datenbanksysteme in Büro, Technik und Wissenschaft*, Springer, 1993.

[2] ISO TC184/SC4, *Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles*, ISO/DIS 10303-1, ISO, 1993.

[3] ISO TC184/SC4/WG5, *Product Data Representation and Exchange - Part 11: The EXPRESS Language Reference Manual*, ISO/DIS 10303-11, ISO, 1993.

[4] ISO TC184/SC4/WG7, *Product Data Representation and Exchange - Part 21: Clear Text Encoding of the Exchange Structure*, ISO/DIS 10303-21, ISO, 1993.

[5] ISO TC184/SC4/WG7, *Product Data Representation and Exchange - Part 22: Standard Data Access Interface Specification*, ISO TC184/SC4/WG7 N350, ISO TC184/SC4/WG7, 1993.

[6] J. Owen, "STEP - An Introduction", *Information Geometers*, Winchester, 1993.