3. Data Models for Engineering Data

Conventional and Specific Ways to Describe Engineering Data
Overview

• Conventional Models
  – Overview of Data Models
  – Logical Models
    • Databases and the Relational Data Model
    • Object-oriented Data Models
    • Semi-structured Data Models
  – Conceptual Models
    • The Entity Relationship Model (ER)
    • The Unified Modeling Language (UML)

• Engineering Data Models
  – The Standard for the Exchange of Product Model Data (STEP)
    • STEP EXPRESS as a modeling language
    • EXPRESS-G as a graphical/conceptual model
  – STEP files
A data model is a model that describes in an abstract way how data is represented in an information system or a database management system.

• A data model defines syntax and semantics, i.e.
  – How can data be structured (syntax)
  – What does this structure mean (semantics)

• Very generic term for many applications
  – Programming languages have their data models (e.g. C++ and Java have object-oriented data models)
  – Conceptual design methods (e.g. ER, UML) represent a data model
  – File formats either apply a data model (e.g. XML) or implement their own
  – Database management systems implement data(base) models
Information System Design Phases

Requirements Analysis

Conceptual Design

Conceptual Models:
ER, UML, EXPRESS-G

Logical Design

Logical Models:
Relational, Object-oriented, Document-oriented, EXPRESS

Physical Design

Physical Models:
SQL-92, SQL:2011, XML, JSON, C++, Java

Implementation

Schallehn: Data Management for Engineering Applications
Types of Data Models

• **Conceptual Models**
  – Describing the concepts of the given Universe of Discourse and their relationships
  – Information requirements of system/users
  – Independent of final structure implementation
  – Often using graphical notation

• **Logical Models**
  – Describes the logical structure of information (data) in the system to be developed
  – Independent of specific (database) systems or (programming) languages

• **Physical/Implementation Models**
  – Describes all details of how information is represented
The Relational Model (RM)

- Developed since early 1970s based on mathematical theory of relations and operations performed on them (relational algebra)
- **SQL** (Structured Query Language) as a strong standard to access relational databases
- Relational Database Management Systems (RDBMS) implement RM, most often based on SQL
- RDBMS are state of the art for database storage
SQL/RM: Basic Concepts

• Data is stored as **rows/records (tuples*) in tables** (relations) **with** values for each **column** (attribute)
• Rows can be identified by special columns called **primary keys**, for which a unique value must exist
• **Foreign keys** can be used to establish connections across data in different tables
• Constraints can be specified to grant consistency

* Terms in brackets relate to relational theory/mathematics
### SQL/RM: Simple Example

<table>
<thead>
<tr>
<th>PartID</th>
<th>Name</th>
<th>Weight</th>
<th>SupplierID</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT-876-140425</td>
<td>Plunger</td>
<td>143.5</td>
<td>1</td>
</tr>
<tr>
<td>FT-852-130707</td>
<td>Shaft</td>
<td>77.0</td>
<td>3</td>
</tr>
<tr>
<td>FT-855-140809</td>
<td>Bolt</td>
<td>15.7</td>
<td>1</td>
</tr>
<tr>
<td>TT-707-778</td>
<td>Case</td>
<td>22.8</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SupplierID</th>
<th>Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reed &amp; Sons</td>
<td>New York</td>
</tr>
<tr>
<td>2</td>
<td>CaseStudio</td>
<td>Boston</td>
</tr>
<tr>
<td>3</td>
<td>ToolTime</td>
<td>Austin</td>
</tr>
</tbody>
</table>
## SQL/RM: Tables

<table>
<thead>
<tr>
<th>PartID</th>
<th>Name</th>
<th>Weight</th>
<th>SupplierID</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT-876-140425</td>
<td>Plunger</td>
<td>143.5</td>
<td>1</td>
</tr>
<tr>
<td>FT-852-130707</td>
<td>Shaft</td>
<td>77.0</td>
<td>3</td>
</tr>
<tr>
<td>FT-855-140809</td>
<td>Bolt</td>
<td>15.7</td>
<td>1</td>
</tr>
<tr>
<td>TT-707-778</td>
<td>Case</td>
<td>22.8</td>
<td>2</td>
</tr>
</tbody>
</table>
### SQL/RM: Primary Keys

<table>
<thead>
<tr>
<th>PartID</th>
<th>Name</th>
<th>Weight</th>
<th>SupplierID</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT-876-140425</td>
<td>Plunger</td>
<td>143.5</td>
<td>1</td>
</tr>
<tr>
<td>FT-852-130707</td>
<td>Shaft</td>
<td>77.0</td>
<td>3</td>
</tr>
<tr>
<td>FT-855-140809</td>
<td>Bolt</td>
<td>15.7</td>
<td>1</td>
</tr>
<tr>
<td>TT-707-778</td>
<td>Case</td>
<td>22.8</td>
<td>2</td>
</tr>
</tbody>
</table>
### SQL/RM: Foreign Keys

**PartID** | **Name**  | **Weight** | **SupplierID**
--- | --- | --- | ---
GT-876-140425 | Plunger | 143.5 | 1
FT-852-130707 | Shaft | 77.0 | 3
FT-855-140809 | Bolt | 15.7 | 1
TT-707-778 | Case | 22.8 | 2

**SupplierID** | **Name**       | **Location**
--- | --- | ---
1 | Reed & Sons | New York
2 | CaseStudio | Boston
3 | ToolTime | Austin
The Structured Query Language (SQL)

- Language to access databases structured according to Relational Model
  - Developed based on RM
  - Introduces some minor differences to RM
  - Not a programming language

- Consists of several parts, most importantly:
  - Actual query language to read data
  - Data Definition Language (DDL) to create (empty) databases, tables, etc.
  - Data Manipulation Language (DML) to insert, modify and delete data
SQL: Query Language

```
SELECT <columns>
FROM <tables>
WHERE <condition>
```

- **Declarative language:**
  - Result is described, not how it is computed
  - Actual execution can be optimized by DBMS

- **Typical structure:** SFW-block (SELECT-FROM-WHERE)

- **Input as well as result are always tables**

- **Used from programming languages via standardized or proprietary application programming interfaces (ODBC, JDBC, etc.)**
SQL: Query Language Example 1

```sql
SELECT name, weight
FROM part
WHERE weight > 50;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plunger</td>
<td>143.5</td>
</tr>
<tr>
<td>Shaft</td>
<td>77.0</td>
</tr>
</tbody>
</table>
SQL: Query Language Example 2

```
SELECT p.name, s.name
FROM part p, supplier s
WHERE p.supplierid = s.supplierid
  AND s.name LIKE 'Reed%';
```

<table>
<thead>
<tr>
<th>Part.Name</th>
<th>Supplier.Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plunger</td>
<td>Reed &amp; Sons</td>
</tr>
<tr>
<td>Bolt</td>
<td>Reed &amp; Sons</td>
</tr>
</tbody>
</table>
CREATE TABLE part (  
  partid INTEGER PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  weight DECIMAL(10,2),
  supplierid INTEGER REFERENCES supplier(supplierid)  
);

• DDL= Part of SQL language used to define schema elements (tables, constraints, views, etc.)
SQL: Data Manipulation Language (DDL)

```
INSERT INTO supplier VALUES (4,'Rex & Smith', 'Baltimore');

UPDATE supplier
SET location='Woburn'
WHERE supplierid=2;

DELETE FROM part
WHERE supplierid=1;
```

- DML = Part of SQL language to insert, modify and delete data
Engineering and RDBMS

• RDBMS often used for
  – Product Lifecycle Management (Product Data Management, Engineering Data Management)
  – Applications for generic tasks, e.g. Enterprise Resource Planning, Workflow Management Systems, Supply Chain Management, etc.

• RDBMS less often or not used for
  – Direct structured storage of product definition data

• Details in Section 4
Object-oriented Data Models

- Enhanced semantic modeling
  - Allows more flexible and re-usable definitions
  - More semantic concepts add complexity to data model/languages
- Developed gradually until major breakthrough in 1980s
- Similar concepts of data modeling applied for numerous application fields in computer science, e.g.
  - Object-oriented Analysis and Design (e.g. UML)
  - Object-oriented Programming (e.g. C++, Java)
  - Object-oriented Databases (e.g. db4o, Versant)
  - Object-relational Databases (SQL since SQL:1999)
  - Object-oriented User Interfaces

Schallehn: Data Management for Engineering Applications
OO: Enhanced Semantic Modeling

- **Objects** as instances (data) of classes
- User-defined **Classes** as definitions (schema) of
  - The structure of objects with **Attributes** and **Relationships**
  - The behavior of objects by **Methods** (class functions)
- **Encapsulation** to differentiate between appearance to use user of objects of classes (interface) and their internal structure and behavior (implementation)
- Re-usability of definitions by **Specialization** among classes
  - **Inheritance**: specialized classes (subclasses) also possess the attributes, relationships and methods of the classes they were derived from (superclasses)
  - **Polymorphism**: objects of a subclass are also objects of the superclass and can be used accordingly
Attributes represent properties of objects of a class, for which an object carries concrete values.

Defined based on data types:
- Basic data types defined of implementation model (e.g. `int`, `float`, `char` in C++)
- Pre-defined complex types (e.g. `string` in C++)
- User-defined complex types (e.g. classes for `Address`, `Date`, `Coordinates`, etc.)

```cpp
class Part {
    ...
    string name;
    int version_id;
    Date lastModified;
    ...
};
```

This and all following examples on OO are in C++
• Specification of behavior of objects in terms of functions on that object

• **Interface (Signature, declaration):**
  – Specifies how the method can be used
  – External view of the method
  – Name, parameters and return value

• **Implementation (definition):**
  – Provides executable source code for method
  – Internal view of the method

• Interface and implementation may be separated (e.g. in C++)

• Constructors as special methods to create objects of that class

```cpp
class Part
{
    ...  
    Part(string n);
    void createNewVersion();
    ...  
};

...

Part::Part(string n)
{
    name = n;
    version_id = 1;
}

void Part::createNewVersion()
{
    version_id++;
}
```
1:1 and N:1 Relationships between different objects most often represented by pointers (physical address, e.g. C++) or references (logical, e.g. Java)

Bidirectional, 1:N and N:M relationships require additional type construction

```c++
class Part {
  ...
  Engineer* responsibleEngineer;
  ...
};

class Engineer {
  ...
  string name;
  string department;
  set<Part*> designedParts;
  ...
};
```
OO: Encapsulation

- External (interface) and internal (implementation) structure of class maybe specified
- Typically access modifiers such as
  - Public: attribute or method accessible from everywhere
  - Private: only accessible within methods of this class
  - Protected: accessible within this class and in subclasses
  - Package (Java only): within this library

```cpp
class Part {
    public:
        Part(string n);
        void createNewVersion();
    private:
        string name;
        int version_id;
        Date lastModified;
        Engineer* responsibleEngineer;
};
```
• Objects of classes
  – Defined within source code, i.e. function and method implementation
  – Notion class implies set of objects conforming to the defined structure
  – Carry values for attributes
  – Methods are called on objects, e.g. using notations like `obj.method()` or `obj->method()`

```cpp
class Part
{
    public:
    Part(string n);
    void createNewVersion();
    private:
    string name;
    int version_id;
    ...
};

// Main program
int main()
{
    Part* obj1 = new Part("Wheel");
    Part* obj2 = new Part("Hub");
    ...
    obj1->createNewVersion();
    ...
    return 0;
}
```
**OO: Specification**

- **Relationship between classes**: to model more specific subsets of objects with additional properties and methods.

- **Inheritance**: attributes and methods defined in superclass are also defined in subclass (also referred to as subtyping).

- **Polymorphism**: wherever objects of a superclass can be used, object of any subclass of it can be used, too.

```cpp
class Part {
    public:
        Part(string n);
        void createNewVersion();
    private:
        string name;
        int version_id;
        Date lastModified;
        Engineer* responsibleEngineer;
};

class ManufacturedPart : public Part {
    private:
        string manufacturingDepartment;
};

class PurchasedPart : public Part {
    private:
        string vendor;
};
```
**OO and Engineering Data**

- Rich semantic modeling suitable to support complex data structures
- Typical implementation model of engineering applications
  - Conceptual Modeling
  - Programming and Development
  - File Storage
- Some concepts integrated with STEP data models EXPRESS and EXPRESS-G
  - Specialization
  - Relationships
- Object-oriented and Object-Relational Databases suitable but not commonly used for Engineering Data
XML

- **eXtensible Markup Language**
  - Hierarchical structure of nested elements (tags)
  - Elements may have attributes
  - Actual data on the leave level
  - Mix of content (data) and description (schema, metadata)

- Developed based on SGML (document processing) to exchange any kind of data on the Web

- Inspired by HTML (also based on SGML), which is only useful to exchange

- Can be considered a neutral text format for files

- Application-specific schemas of valid documents can be defined by Document Type Definitions (DTD) or XML Shema (XSD)

- Standard software/libraries for XML processing publically available
XML Example: EAGLE .sch File

```xml
<schematic>
  <parts>
    <part name="SUPPLY1" deviceset="GND" device=""/>
    <part name="C1" deviceset="C-EU" device="050-024X044" value="22pF"/>
  </parts>
  <sheets>
    <sheet>
      <instances> <!-- Positions the parts on the board. E. g.: -->
        <instance part="SUPPLY1" gate="GND" x="132.08" y="187.96"/>
        <instance part="C1" x="-50.8" y="200.66" rot="R270"/>
      </instances>
      <nets>
        <net name="N$1" class="0">
          <segment>
            <wire x1="9.44" y1="19.04" x2="8.9" y2="19.04" width="0.15"/>
            <wire x1="8.9" y1="19.04" x2="8.9" y2="20.66" width="0.15"/>
            <wire x1="8.9" y1="20.66" x2="2.4" y2="20.66" width="0.15"/>
            <pinref part="C1" pin="5"/>
            <pinref part="SUPPLY1" pin="1"/>
          </segment>
        </net>
      </nets>
    </sheet>
  </sheets>
</schematic>
```

[Source: Philipp Ludwig]
XML Structure and Data Model

- Markup language intended to describe structure within documents and document collections in files or databases
- Data logically represented according to **Document Object Model (DOM)** as hierarchy/tree of
  - **Element nodes** (labeled internal nodes)
  - One labeled **root node** (represents document content)
  - **Text nodes** as leaf nodes represent actual data
  - **Attribute nodes** as special sub-nodes with a child text node
- Structure is
  - **Well-formed**: conforms to general XML rules
  - **Valid**: possible nesting of elements, attributes, etc. conform to a schema defined as Document Type Definition (DTD) or XML Schema (XS)
XML DOM Example

Element node

Text node

Attribute node

schematic

parts

sheets

part

part

name

deviceset

device

text

text

text

"SUPPLY1"

"GND"

"

Schallehn: Data Management for Engineering Applications
XML Example: eagle.dtd

- DTD used for schema definition, i.e. valid .sch files
- Small excerpt of eagle.dtd (publically available):

```xml
<!ELEMENT schematic (description?, libraries?, attributes?,
    variantdefs?, classes?, parts?, sheets?, errors?)>
<!ATTLIST schematic
    xreflabel %String; #IMPLIED
    xrefpart %String; #IMPLIED>

... 

<!ELEMENT part (attribute*, variant*)>
<!ATTLIST part
    name %String; #REQUIRED
    library %String; #REQUIRED
    deviceset %String; #REQUIRED
    device %String; #REQUIRED
    technology %String; ""
    value %String; #IMPLIED>
```
XML in Engineering

• Many formats based on XML
• Especially intended for data exchange
• Some examples:
  – **Collada** for interactive 3D applications
  – **3DXML** for the exchange of geometrical data
  – **EAGLE** board (BRD) and schema (SCH) files for electronic circuits (see above)
  – **CAEX** general purpose language for the exchange of engineering data by European consortium
  – **AutomationML** for plant engineering
  – ...
JSON

- JavaScript Object Notation
- More recent, “lightweight” alternative to XML
- Also provides Schema definition language
- Developed for Web and Cloud applications
- In Engineering:
  - No major usage
  - Current development of CAD JSON export to support web-based interoperability

```json
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "phoneNumber": [
        {
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

Based on [http://en.wikipedia.org/wiki/JSON]
Conceptual Models

- Used during Conceptual Design
  - Early development phase
  - Independent of implementation
  - Focus on completeness and soundness description of universe of discourse

- Typically using graphical notation

- Covered here:
  - General purpose models:
    - Entity Relations Model (ERM or ER Model)
    - Unified Modeling Language (UML)
  - Specialized model for application areas
    - EXPRESS-G for engineering data
Focus of Conceptual Models

Conceptual Models: ER, UML, EXPRESS-G

Logical Models: Relational, Object-oriented, Document-oriented, EXPRESS


Requirements Analysis
Conceptual Design
Logical Design
Physical Design
Implementation

Schallehn: Data Management for Engineering Applications
The Entity Relationship (ER) Model

• Developed by Peter Chen in 1976
• Commonly used for design of relational databases
• Set of rules for mapping ER concepts to tables
• Several derivatives with more efficient notation, e.g.
  – Idef1x
  – Crows foot/Barker’s notation
• Several extension, to introduce more powerful (e.g. object-oriented) concepts
ER Model: Basic Concepts

- **Entity types (rectangles):** represent sets of real-world entities with common attributes
- **Attributes (ovals or rounded boxes):** hold property values of entities, keys (underlined) as identifying attributes
- **Relationship types (diamond shaped boxes):** possible relationship between instances of entity types

Schallehn: Data Management for Engineering Applications
**ER Concepts: Cardinalities /1**

- **Cardinalities**: indicate how often instances of entity types might participate in a certain relations
- Min/max cardinalities or, alternatively but less precise, only maximum value
- Optional relationships: minimum cardinality is zero
- 1:1, 1:N or N:M relationships (example above: 1:N relationship) as typical classes of relationships based on cardinalities

Schallehn: Data Management for Engineering Applications
ER Concepts: Cardinalities /2

- Example above: N:M relationship
- Unspecified cardinalities indicate default case of optional N:M relationship

Schallehn: Data Management for Engineering Applications
ER Concepts: Further Relationships

- **Self-referential relationships on the type-level**

- **Relationships expressing existential dependencies (weak entity types)**

- **Relationships between more than two entity types (n-ary relationships)**

Schallehn: Data Management for Engineering Applications
Mapping ER Schema to Relational

• Simple rules
  – Entity types map to tables
  – Attributes map to columns
  – Key attributes map to primary key columns
  – N:M relationships map to tables with keys of participating entity types as columns
  – 1:1 relationships
    • Non-optional: entity types and relationship can be merged into one table
    • Optional: map to table with keys of participating entity types as columns
  – 1:N relationships
    • Non-optional: entity types and relationship can be merged into one table
    • Optional: map to table with keys of participating entity types as columns

• Some variance allowed to improve performance, simplicity, etc.

Schallehn: Data Management for Engineering Applications
The Unified Modeling Language (UML)

- Object-oriented modeling language/model for general software engineering
- Developed in mid 1990s as a combination of several languages/conceptual models
- Contains several diagram types for describing different aspects of structure and behavior
  - Class diagrams
  - Object diagrams
  - State diagrams
  - Sequence diagrams
  - Etc.
- Class diagrams useful to describe database or file schemas

Schallehn: Data Management for Engineering Applications
UML Class Diagrams

• Cover basic data model aspects such as ER Model
  – Classes entity types
  – Attributes and key attributes for classes
  – Relationships with cardinalities

• In addition, object-oriented concepts:
  – Specialization and inheritance
  – Encapsulation
  – Methods
UML Class Diagram Example

**Class: Drawing**
- name
- status
+ create()
+ check()
+ edit()
+ approve()
+ reject()

**Class: Employee**
- firstname
- lastname

**Class: Designer**
- domain

**Class: Inspector**

- modified by
- approved by

Schallehn: Data Management for Engineering Applications
STEP

• STandard for the Exchange of Product model data
• Developed since 1984 by international consortium
• Standardized since 1990s as ISO 10303
• Contains
  – General methods for describing data and schemas
  – Definitions of generic file formats
  – Application-specific methods for engineering domains
STEP Parts relevant for Data Modeling

- Parts most relevant for Geometric Models:
  - 10303-1x
    - 10303-11: Description Methods, e.g. EXPRESS and EXPRESS-G
  - 10303-2x
    - 10303-21: Implementation Methods, e.g. STEP files
    - 10303-22: Standard Data Access Interface SDAI
    - 10303-23, 24 ...: SDAI C++, C etc. Language Bindings
    - 10303-28: STEP XML
  - Further 10303-XX
    - 10303-42: Geometric and topological representation
    - 10303-52: Mesh-based topology
  - 10303-2XX
    - ...: Application Protocols
    - ...: ...
EXPRESS and EXPRESS-G

• Represent Data Model of STEP Standard
• EXPRESS: textual notation
  – Formal notation to describe data structures
• EXPRESS-G: graphical notation
  – Easy to understand
  – Most concepts of EXPRESSED can be described 1:1, except for complex constraints
• For storage/implementation mapped to file format (10303-21) or concrete language (10303-22 ff.)

Schallehn: Data Management for Engineering Applications
EXPRESS-G: Basic Data Types

- BINARY
- BOOLEAN
- INTEGER
- LOGICAL
- NUMBER
- REAL
- STRING
EXPRESS-G: Entity Types and Attributes

Part
  \- *name: STRING
  \- department: INTEGER
  \- last_modified: Date
Entities and Attributes (Remarks)

- Entity types as plain rectangles
- Attributes as relationships to basic types or defined types
EXPRESS-G: Defined Types

Date

- day
  - INTEGER

- month
  - INTEGER

- year
  - INTEGER

0 < day < 32
0 < month < 13
0 < year

Schallehn: Data Management for Engineering Applications
SCHEMA Parts;

TYPE Date

   day : INTEGER;
   month : INTEGER;
   year : INTEGER;

WHERE

   WR1: (SELF\day > 0) AND (SELF\day < 32);
   WR1: (SELF\month > 0) AND (SELF\month < 13);
   WR1: (SELF\year > 0);

END TYPE;

ENTITY Part

   name : UNIQUE STRING;
   department : OPTIONAL INTEGER;
   last_modified : Date;

END ENTITY;

...
Defined Types (Remarks)

• Can be used just like basic types
• Defined as
  – based on one basic or
  – composed of several basic or defined types
• Constraints maybe used to
  – Limit domain of values
  – Specify any consistency requirement
EXPRESS-G: Enumeration Data Type

Schallehn: Data Management for Engineering Applications
SCHEMA Parts;

... 

ENTITY Engineer
    name : STRING;
    status : ENUMERATION OF (internal, external);
END ENTITY;

... 

END SCHEMA;
Enumeration Data Type (Remarks)

- Enumeration is special type for categorical attribute
- Consists of definition of small set of possible values
EXPRESS Relationships

SCHEMA Parts;

ENTITY Part
  ...
  responsibleEngineer    : Engineer;
  versions               : LIST[1:?] OF PartVersion;
END ENTITY;

ENTITY Engineer
  designedParts          : SET[0:?] OF Part;
END ENTITY;

...

END SCHEMA;
Relationships (Remarks)

- Relationships between entity types are directional
- Bidirectional relationships represented as two relationships
- Multiple participation can be represented by Aggregation types
  - List (L): ordered collection
  - Set (S): unordered collection without duplicates
  - Bag (B): unordered collection with duplicates
  - Array (A): collection of fixed size (ordered, with duplicates)
- Cardinalities with [min:max] notation where ? indicates an arbitrary cardinality
EXPRESS-G: Subtyping

(ABS)Part

- *name : STRING
- department : INTEGER
- last_modified : Date

ManufacturedPart

PurchasedPart

vendor : STRING

Schallehn: Data Management for Engineering Applications
EXPRESS: Subtyping

```express
SCHEMA Parts;

ENTITY Part
  ABSTRACT SUPERTYPE OF
    (ONEOF (ManufacturedPart, PurchasedPart));
  ...
END ENTITY;

ENTITY ManufacturedPart
  SUBTYPE OF (Part);
END ENTITY;

ENTITY PurchasedPart
  SUBTYPE OF (Part);
  vendor : STRING;
END ENTITY;

...
Subtyping (Remarks)

- Inheritance (supertype attributes are also defined for subtype) and polymorphism (substitutability) are supported
- Multiple inheritance (more than one supertype) is possible
- Instances may be of several subtypes at the same time
  - Can be constrained by cardinalities, e.g. ONEOF = instance only of either one of the specified subtypes
Further EXPRESS-G Constructs

- **Schemas** as blocks consisting of entities and relations
- **Select types** to represent alternatives of various (entity or defined) types to use for relationship
- **Methods** according to object-oriented concepts
- Derived attributes as calculated properties
- **Communication relationships** to indicate interactions
- Entity and page **references** for complex or
- ...

Schallehn: Data Management for Engineering Applications
ISO 10303-21: STEP Files

- ASCII-based textual file format for step data
- File extensions `.stp` or `.step` for files according to application protocols
- Commonly used for data exchange in engineering
- Typically structured according to an EXPRESS schema
- Files typically consists of
  - ISO-10303-21-declaration in first line
  - Short HEADER section containing metadata, including a reference to the schema (typically STEP Application Protocol)
  - DATA section with lines each representing a numbered entity instance according to schema
ENTITY cartesian_point
  SUPERTYPE OF (ONEOF(cylindrical_point, polar_point, spherical_point))
  SUBTYPE OF (point);
  coordinates : LIST [1:3] OF length_measure;
END_ENTITY;
Example AP214 .STEP File

ISO-10303-21;
HEADER;
FILE_DESCRIPTION( ( ' ' ), ' ' );
FILE_NAME( 'pumpHousing.stp', '2004-04-13T21:07:11', ( 'Tim Olson' ), ( 'CADSoft Solutions Inc' ), ' ', 'ACIS 12.0', ' ' );
FILE_SCHEMA ( ( 'AUTOMOTIVE_DESIGN { 1 0 10303 214 2 1 1 }' ) );
ENDSEC;
DATA;

#3716 = POINT_STYLE( ' ', #6060, POSITIVE_LENGTH_MEASURE( 1.000000000000000E-06 ), #6061 );
#3717 = CARTESIAN_POINT( ' ', ( -1.10591425372267, 3.05319777988191, 0.541338582677165 ) );
#3718 = CURVE_STYLE( ' ', #6062, POSITIVE_LENGTH_MEASURE( 1.000000000000000E-06 ), #6063 );
#3719 = LINE( ' ', #6064, #6065 );
#3720 = CURVE_STYLE( ' ', #6066, POSITIVE_LENGTH_MEASURE( 1.000000000000000E-06 ), #6067 );
#3721 = CIRCLE( ' ', #6068, 1.75849340964528 );
#3722 = CURVE_STYLE( ' ', #6069, POSITIVE_LENGTH_MEASURE( 1.000000000000000E-06 ), #6070 );
#3723 = CIRCLE( ' ', #6071, 0.540114611464642 );
#3724 = SURFACE_STYLE_USAGE(.BOTH., #6072 );
#3725 = FACE_OUTER_BOUND( ' ', #6073, .T. );

ENDSEC;
END-ISO-10303-21;

[Source: Paul Bourke
http://paulbourke.net/dataformats/]
**STEP SDAI**

- **Standard Data Access Interface ISO 10303-22** defines standard bindings to languages (C, C++, Java) for STEP data access
- Similar to an API for an RDBMS (ODBC, JDBC) or ODBMS defines basic functionality such as
  - Sessions
  - Database connectivity
  - Data dictionary
- Defines mappings of EXPRESS types to language constructs, e.g.
- Not specific to geometrical data → used more often for other applications

Schallehn: Data Management for Engineering Applications
Further Readings

