Framework for CAD/CAM/PDM Applications

the Synergy of STEP and Java™ Technology

Lothar Klein,
LKSoftWare GmbH
http://www.lksoft.com

SDAI Tutorial and the J-SDAI implementation

- Overview
  - STEP description and implementation methods
  - EXPRESS versus Java
- Basic SDAI Functionality
- Extended SDAI Functionality
- Future SDAI Directions

Java history

- 1995/96: Java 1.0 released. It is a **replacement for C++**!
- 1997: Major upgrade Java 1.1
- 1998: Sun is approved to submit Java as a Publicly Available Specification (PAS) at ISO JTC1.
- 1998: Swing -> **Java 2 Platform** (JDK 1.2) an almost complete OS-interface
- 1999: Sun moves the standardization to ECMA, but later it cancels this activities
- 2000: JDK 1.3, Split into several platforms
  - Enterprise (Enterprise JavaBeans™)
  - Standard
  - Micro

What is Java™ technology ?

Java 2 Platform, Standard Edition, actual Version 1.2.2

- Java Programming Language
- Java Virtual Machine, Platform independent
- Java 2 Platform API
  - Swing (Windows API)
  - Java Beans (reusable software components)
  - Java 2D API
  - JDBC, Database API for sql
  - Security
  - Network, Remote Method Invocation, CORBA/ IDL

Java Extensions

- Java 3D, JINI, Enterprise Java Beans (EJB), ...

STEP and associated Standards

- ISO 10303-1x, 2x
  Description and **Implementation** methods
- ISO 10303-4x, 1xx, 5xx
  Basic STEP Models
- ISO 10303-2xx, 3xx
  Applications Protocols (AP) and Tests (ATS)
- ISO 13584, PLIB (Parts Library)
- ISO 15926, OIL & GAS
- ISO 15531, MANDATE
- ISO 14649, NC programming

EXPRESS Standards

- EXPRESS(-1): ISO 10303-11:1994
  EXPRESS-G Graphical representation
- EXPRESS-TC1/2 Technical Corrigendum
- EXPRESS-Amendment

- EXPRESS-X Maps and Views of Entities
- EXPRESS-2 (behaviour, active processing of data due to events)

EXPRESS

- information modeling language
- used to define the data models of STEP
- AIM: Application Integrated model
  based on common integrated models
- ARM: Application reference model

EXPRESS Data Types

- primitive data type:
  INTEGER, REAL (NUMBER), STRING, BINARY, BOOLEAN, LOGICAL
- Entity data type and complex entity data type
- constructed data type
- enumeration data type
- select data type
- defined data type
- aggregate data type

plus constraints (expressions, functions, procedures, rules)

EXPRESS-G

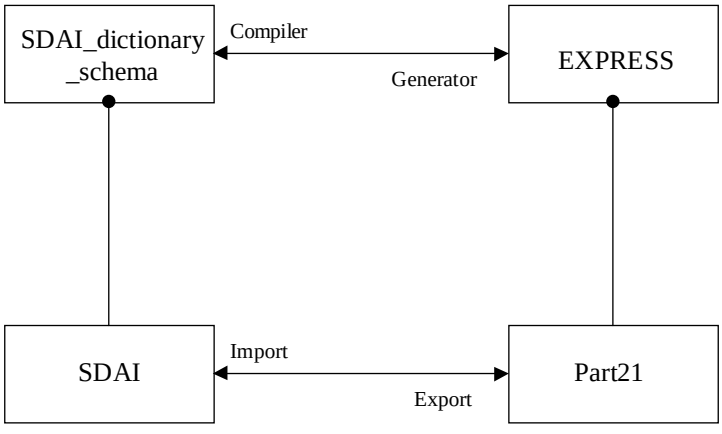
STEP Data Storage and Exchange

- exchange by file, stream or similar
  - EXPRESS-I (part12, outdated)
  - **Clear Text Encoding - "*Physical File*" - part21**
  - part28 (XML)
  - vendor specific, e.g. J-SDAI binary format
- active access through
  - rational or object oriented data base
  - **SDAI**
  - vendor specific

Processing STEP Data, a classification

- Internal representation does not reflect the EXPRESS structure. For data exchange information is mapped during export/import -> loss of information
- Internal representation fits with with the EXPRESS structure. Entity instances and aggregates are represented as objects -> information is preserved
  - Implementation directly based on EXPRESS
  - **SDAI conformant**
  - SDAI similar, but not conformant

Dependencies of the STEP Description and Implementation Methods



Standard Data Access Interface (SDAI, ISO 10303-22)

- Abstract Application Programming Interface (API) for STEP data
- Programming Language bindings to
  - C++ ISO 10303-23
  - C ISO 10303-24
  - Fortran (canceled) ISO 10303-25
  - IDL (to be canceled) ISO 10303-26
  - **complete Java** **ISO 10303-27**
  - lightweight Java ISO 10303-29

Java programming language binding to the SDAI with Internet/Intranet extensions

- **To be published as Technical Specifications (TS)**
- **Split into 2 documents**
  - **ISO 10303-27: complete**
  - **ISO 10303-29: lightweight + EJB**

Installation overview (1)

- Till now only manual installation
- J-SDAI runs with Java 1.1, but we recommend to use Java 1.2 (Java 2 platform). In future we will move to JDK 1.3 for better complex type support (alternatively)
- J-SDAI is an extension of the Java platform. As such its components are installed in the special "ext" directory.
  `c:\jdk1.2.2\jre\lib\ext`
- Tip: When following the default installation of JDK1.2.2, two ext directories are created, one for compilation and one for usual prgram execution.
- During the installing JDK 1.2.2 adjust the Java runtime system to "c:\jdk1.2.2\jre"

Installation overview (1)

- No *classpath* needed ! Files to install:
  jsdai.properties
  jsdai_m.jar
  jsdai_lib.jar
- Repository directory
  c:\sdairepositories
- Applications:
  jsdaix_ap203book.jar, jsdaix_look_feel.jar, jsdaix_view3d.jar

package structure

- `jsdai.lang` main session
- Java classes and interfaces for EXPRESS
  - `jsdai.dictionary` dictionary schema
  - `jsdai.mapping` mapping schema
  - `jsdai.SXxx` application schema xxx
- `jsdai.util` SdaiTerm, SdaiEdit
- `jsdai.bean` session beans
- `jsdaix.view3d` 3D-viewer bean
- `jsdaix.pdm_beans` PDM (p41) beans
- `jsdaix.look_feel` special L&F
- `jsdaix.ap203book` AP203 application

J-SDAI implementation facts

- 100% pure Java =
  all written in Java + platform neutral
- All pointers (Java references) for fast access
  - bi-directional between model and entity-instance
  - bi-directional for explicit attribute (implicit inverse attributes)
  - direct link form application level to meta level
- compact and specialized binary format for
  - persistent storage in repository
  - and internet access
- In parallel **late-** and **early-binding** access for entity type, attributes and aggregates

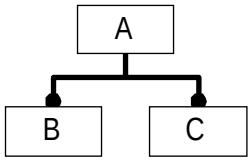
Topics: Basic SDAI Functionality

- Java representation of entities
- Management classes
- Reading and writing part21 files
- Handling of inverse relations
- examples
- Transaction
- The role of SchemaInstance
- Validation
- AP-Interoperability, usage of common data between several APs.
- late-binding and the SDAI_dictionary_schema

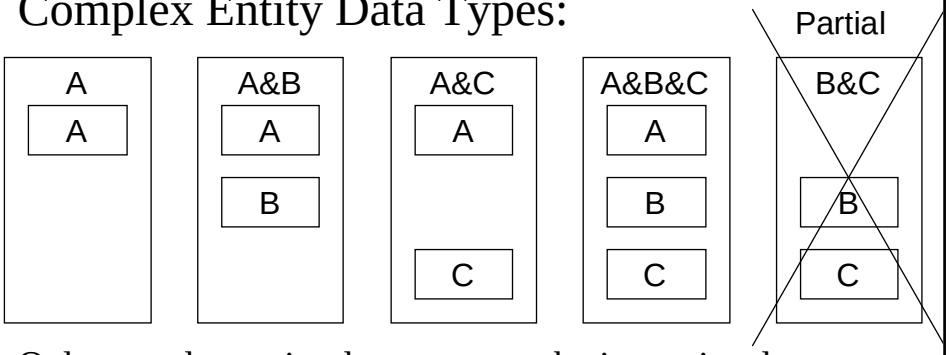
Entity, Entity Instance, Complex

- The terms *Entity* and *Entity-Instance* are not very precise. *Complex Entity* is even miss-leading.
- **Entity Data Type**s are defined in an EXPRESS schema
- **Complex Entity Data Type**s
  - generated by an algorithm for a given EXPRESS schema (ABSTRACT, AND, ONEOF, ANDOR)
  - have **one** or **several leave** *Entity Data Types*, the later one is often called a **Complex Entity**
  - can be instantiated
- **Partial Complex Entity Data Types** are incomplete *Complex Entity Data Types.* They exist only in EXPRESS but not in SDAI or Part21.

Entity Data Types:

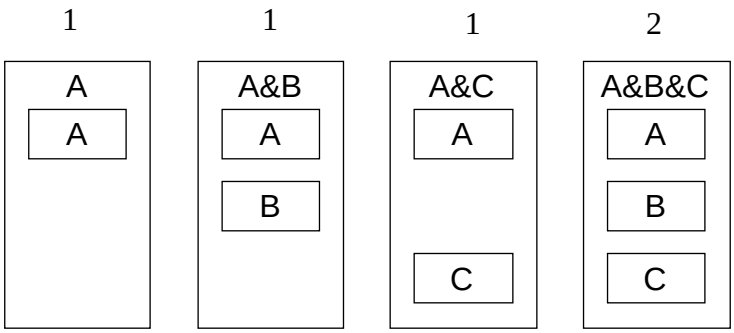


Complex Entity Data Types:



Only complex entity data types can be instantiated

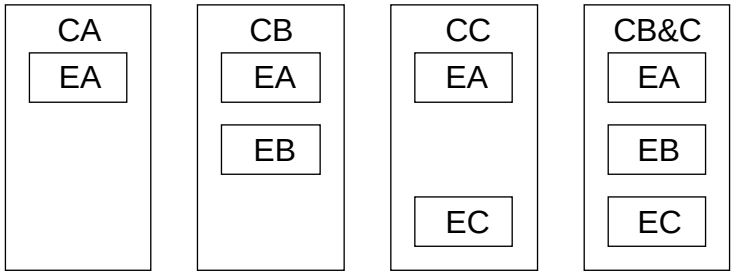
Complex Entity Data Types number of Leave Entity Data Types:



#1=A(...); #2=B(...) #3=C(...); #4=(A(...)B(...)C(...));

Entity Mapping to Java (early binding)

- **E**ntity Data Types are represented by Java Interfaces with prefix "**E**", e.g. "EXxx"
- **C**omplex Entity Data Types are represented by Java Classes with prefix "**C**", e.g. "CXxx"
- Multiple leave entity data types are separated by "&"
- **A**ggregates of Entity Data Types are represented by Java Classes with prefix "**A**", e.g. "AXxx"



Entity Mapping to Java (late binding)

- All *Entity Data Types* and *Complex Entity Data Types* are represented by the Java Interface **EEntity**
- All *Aggregates* of *Entity Data Types* are represented by the Java class **AEntity**
- Linking of early and late binding is realized by:

- EEntity is directly or indirectly extended by every early binding Entity Interface

```
public interface EXxx extends EEntity {...}
public interface EYyy extends EXxx {...}
```

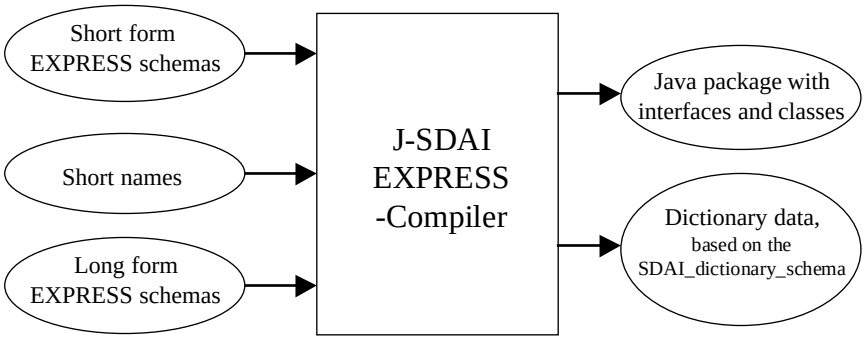
- AEntity is directly interfaced by every early binding Aggregate Class

```
public class AXxx extends AEntity {...}
```

Logical and Enumerations

LOGICAL and early binding enumerations:

```
public final class ELogical {
    // unset == 0;
    public static final int FALSE = 1;
    public static final int TRUE = 2;
    public static final int UNKNOWN = 3;
    public static final String values[] = {"FALSE", "TRUE",
                                           "UNKNOWN"};
    private ELogical( );   // no instances are allowed
    public static int toInt(String str);
    public static String toString(int value);
}
```

J-SDAI Libarary

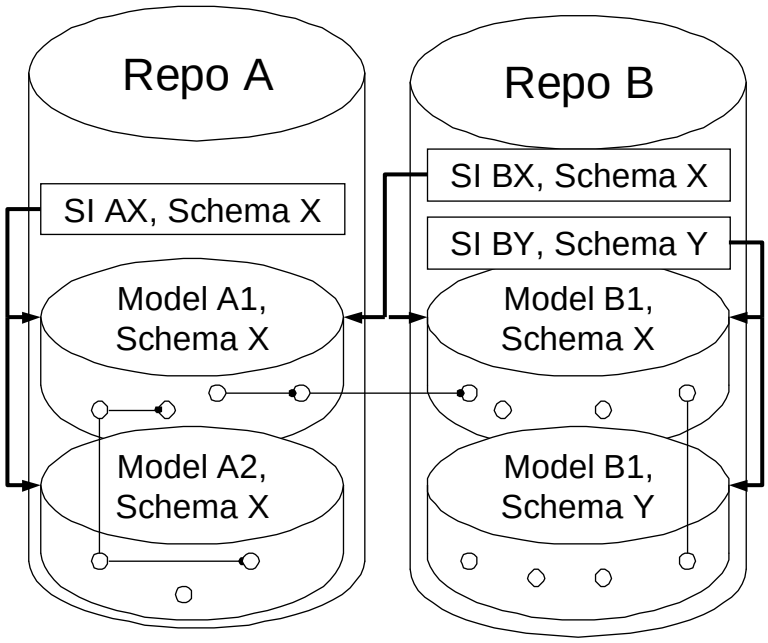
- file: ***jsdai_lib.jar*** (~8.5 MBytes)
- IR-schemas: 49
- AIC-schemas: 19
- PLIB schemas: 2 (soon more)
- AP-AIM schemas: 9
- AP-ARM schemas: 3 (210, 212, 214)
- Mapping Information: 2 (210, 214)
- one special "schema" for cross-mixed Complex Entity Data Types
- some test and meta-schemas

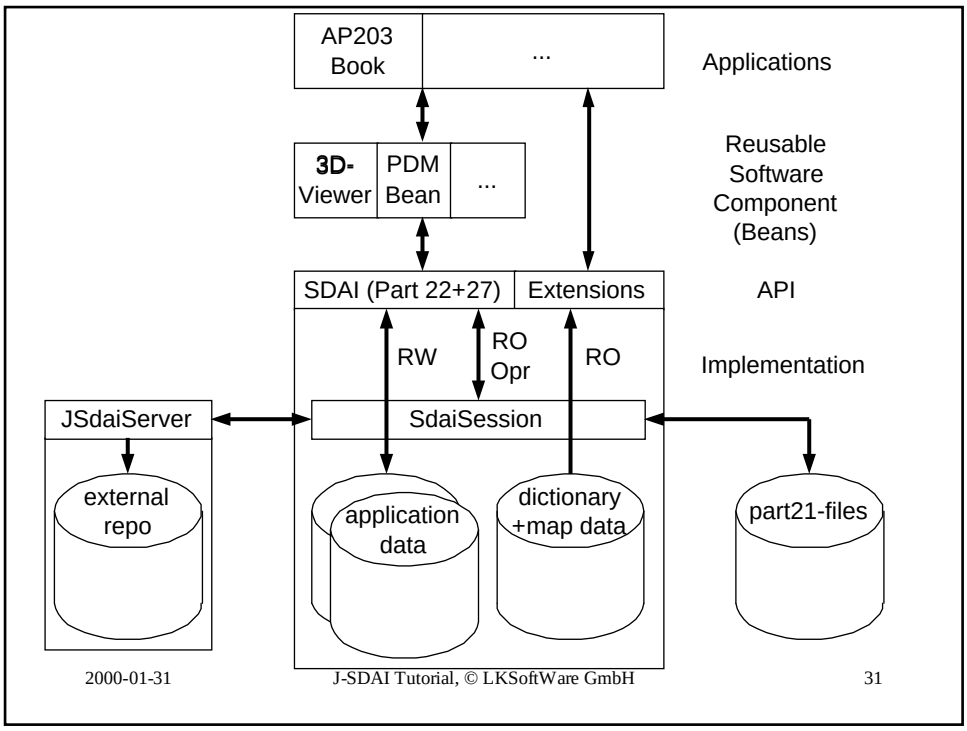
Main Management Objects

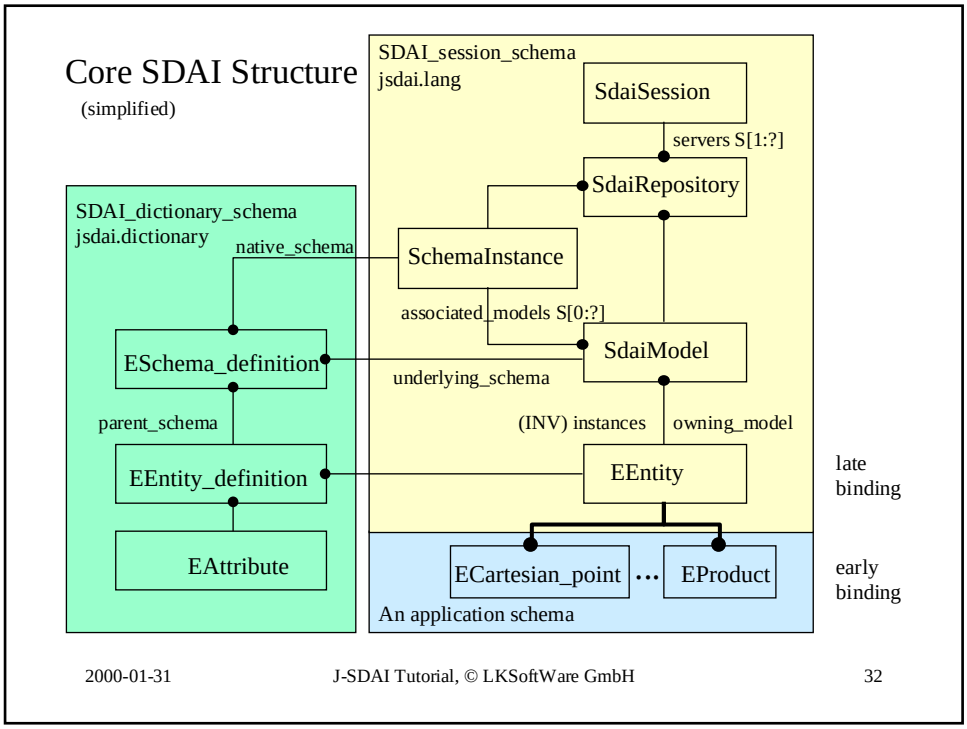
- SdaiSession
  root from which every SDAI activity starts from
- SdaiRepository
  physical storage space for SchemaInstances and SdaiModels
- SchemaInstance
  logical grouping of models to define a valid population of an express schema
- SdaiModel
  grouping of entity instances based on one express schema

Auxiliary Management Objects

- Implementation
  provides runtime information of the implementation
- SdaiTransaction
  controls simultaneous updating of changes
  prevents multiple clients from each other
- EntityExtent
  auxiliary, easy accessing entity instances
- SdaiException
  thrown in case of error conditions




AP203
Book
...
Applications
3D-
Viewer
PDM
Bean
...
Reusable
Software
Component
(Beans)
SDAI (Part 22+27)
Extensions
API
RW
RO
Opr
RO
Implementation
JSdaiServer
SdaiSession
external
repo
application
data
dictionary
+map data
part21-files
2000-01-31
J-SDAI Tutorial, © LKSoftWare GmbH
31


Core SDAI Structure
(simplified)
SDAI_session_schema
jsdai.lang
SdaiSession
servers S[1:?]
SdaiRepository
SDAI_dictionary_schema
jsdai.dictionary
native_schema
SchemaInstance
associated_models S[0:?]
SdaiModel
ESchema_definition
underlying_schema
parent_schema
(INV) instances
owning_model
EEntity_definition
EEntity
late
binding
EAttribute
ECartesian_point
...
EProduct
early
binding
An application schema
2000-01-31
J-SDAI Tutorial, © LKSoftWare GmbH
32

class SdaiException (1)

- Extends java.lang.Exception
- Thrown by every SDAI operation
- Handling of error conditions
- getErrorId(): IDs defined for every case, e.g:

```
/** Repository is not open */
static int RP_NOPN = xx; ...
```

- getErrorDescription() - some explanations
- getErrorBase() - the object causing this error
- getUnderlyingError(), e.g. file I/O or memory

class SdaiException (2)

Typical usage:

```
try {
   // ... some J-SDAI operations
   // ...  "
  } catch (SdaiException e) {
                   e.printStackTrace();
          int id = e.getErrorId();
      String desc = e.getErrorDescription();
      Object base = e.getErrorBase();
   Exception exc = e.getUnderlyingError();
   ...
}
```

class SdaiSession

- Beginning and ending an SDAI application

```
SdaiSession session =
        SdaiSession.openSession();
....
session.closeSession();
```

  after closing the session object becomes invalid.
- Only one SdaiSession object at a time

```
SdaiSession session =
        SdaiSession.getSession();
```


SdaiSession - Exceptions

- SS_NAVL: Session not available
  The implementation cannot open a session
- SS_NOPN: Session is not open
  Access to an SDAI object is possible only when the session is open
- SS_OPN : Session open
  An attempt was made to open a session while a session was still open

SdaiSession - LogWriter

Used to write out warnings, debug messages, measure times etc.

- `static java.io.PrintWriter getLogWriter();`
- `static void setLogWriter(java.io.PrintWriter out);`

Using the logWriter from an application:

```
s.println("My test message");
```


class SdaiImplementation

- Informs about the name, version and features of the SDAI implementation
- Access through SdaiSession

```
Implementation imp =
   session.getImplementation();
```

- Include most information of the Protocol Implementation Conformance Statement (PICS)

SdaiImplementation (SdaiTerm)

```
C:\LKSoft\current>java jsdai.util.SdaiTerm
:so
:si
----------- Implementation -------------
           name: J-SDAI MULTIPLE
          level: Version 2.0 (Build 200, 2000-01-24)
   sdai_version: { iso standard 10303 part(22) version(0) }
binding_version: { iso standard 10303 part(27) version(0) }
    implementation_class: 5
       transaction_level: 3
        expression_level: 1
domain_equivalence_level: 2
:q
```


Levels of Transaction

- Level 1 : No transaction
- Level 2 : SDAI-Model based transaction
- **Level 3 : full transaction (J-SDAI)**

Levels of Expression Evaluation

- Level 1 : No evaluation
- Level 2 : Simple evaluation (J-SDAI)
- Level 3 : Complex Evaluation
- Level 4 : Complete Evaluation

Levels of domain equivalence support

- Level 1 : No domain equivalence
- Level 2 : Domain equivalence support

Levels not relevant for the Java bindings to the SDAI

- Level of session event recording support - replaced by the Java Exception mechanism
- Level of SCOPE support
  deprecated in implementation methods (SDAI, part21)

Implementation Limits - 1

- Q: What is the basis for setting the time?
  A: The Java system time which is based on the underlying host, see class
  java.lang.System.currentTimeMillis()
- Q: What is the maximum number of repositories that may exist within an SDAI session?
  A: No limit
- Q: What is the maximum number of SDAI-models that may exist within a repository?
  A: No limit

Implementation Limits - 2

- What is the maximum length of EXPRESS STRING that is supported?
  A: 64K due to Java serialization
- Q: What is the maximum length of EXPRESS BINARY that is supported?
  A: No limit
- Q: What is the limit of EXPRESS REAL precision that is supported?
  A1: The Java type double has about 14 decimal digits precision, see IEEE 754-1985.
  A2: Arbitrary precision by java.util.BigDecimal. Not implemented in current version (late binding only)

Implementation Limits - 3

- Q: What is the maximum number of elements that may appear in a variable sized aggregate instance?
  A: No limit
- Q: What is the maximum number of index positions that may appear in an array instance?
  A: No limit
- Q: If domain equivalence is supported, what method for declaring it in the data dictionary is supported?
  A: Direct usage of EXPRESS short forms

SdaiSession ~> SdaiTransaction

- Start either ReadOnly or ReadWrite Transaction

```
SdaiTransaction t = session.
   startTransactionReadOnlyAccess()
SdaiTransaction t = session.
   startTransactionReadWriteAccess()
```

- Only one transaction at a time
- Accessing the active transaction

```
if (session.testActiveTransaction()){
   t = session.getActiveTransaction();
}
```


SdaiTransaction - operations

- Checking the actual state

```
switch(t.getMode()) {
case SdaiTransaction.NO_ACCESS: ...
case SdaiTransaction.READ_ONLY: ...
case SdaiTransaction.READ_WRITE: ...}
```

- Aborting the transaction

```
t.abort();
t.endTransactionAccessAbort();
```

- Committing the transaction

```
t.commit();
t.endTransactionAccessCommit()
```

SdaiTransaction - Pending changes

Checking if a modification is pending on

- for the whole SdaiSession
  `if (session.isModified()) { ... }`
- for a single SdaiRepository and all its models
  `if (repo.isModified()) { ... }`
- for a single SdaiModel only
  `if (model.isModified()) { ... }`

SdaiTransaction - Example

```
SdaiSession s = SdaiSession.openSession();
SdaiTransaction t = s.startTransactionReadOnlyAccess();
// ... read some data
while (...) {
      // modify/create data ...
      if (s.isModified()) {
            t.commit();
      } else {
            t.abort();
            // ... re-read the previous data
      }
}
t.endTransactionAccessAbort();
session.closeSession();
```

SdaiTransaction - Exceptions

- TR_EXS : Transaction exists
- TR_NEXS : Transaction does not exist
- TR_NRW : Transaction not read-write
- TR_EAB : Transaction ended abnormally
- TR_NAVL : Transaction currently not available

SdaiSession ~> SdaiRepository

- Accessing repositories from SdaiSession

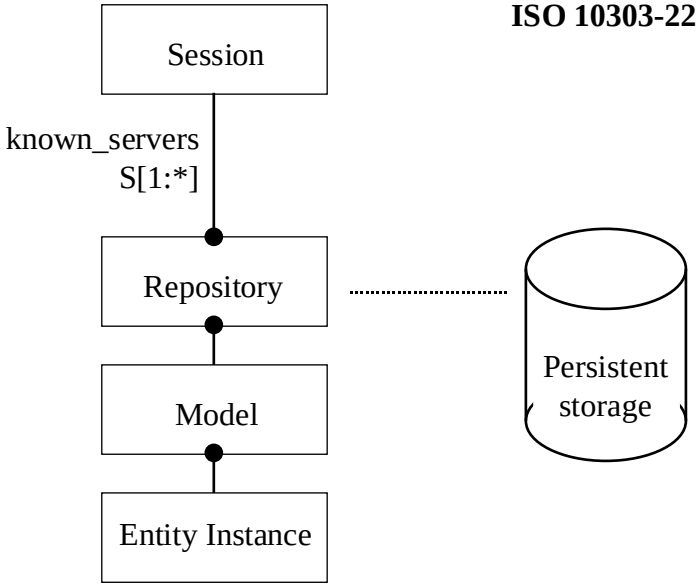
```
ASdaiRepository ar = s.getKnownServers();
ASdaiRepository ar = s.getActiveServers();
SdaiRepository r = ar.getByIndex(5);
```

- A repository is either open (active) or close

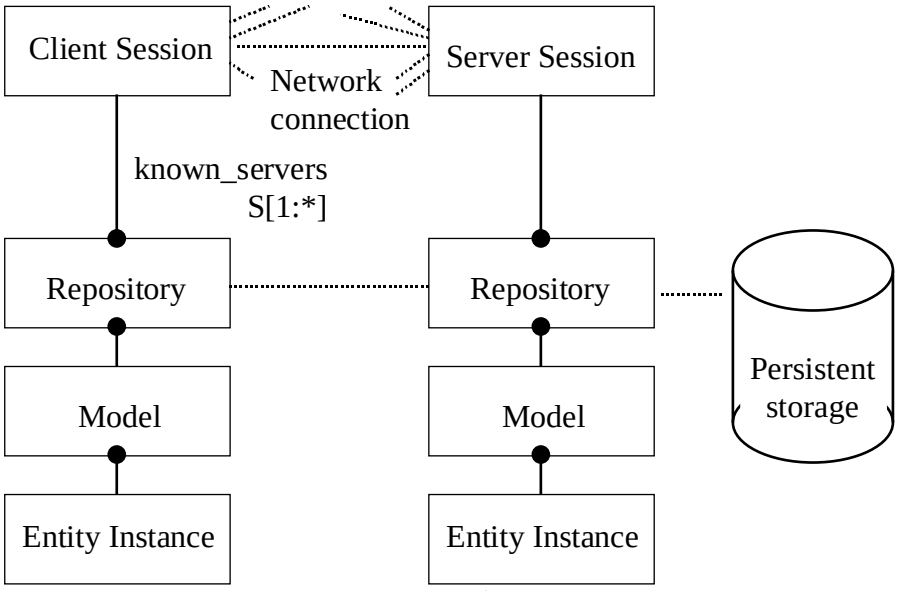
```
if (!r.isActive()) {
   r.openRepository();
}
...
if (r.isActive()) {
   r.closeRepository();
}
```

Basic SDAI Structure without Network support

ISO 10303-22



Extended SDAI with Network support



Requirements for the Network Protocol

- easy to establish (firewall, ...)
  => bi-directional Java streams, based on TCP/IP ports
- suitable for SDAI operations
  - transferring all entity instances take too long
  - transferring single entity instances on request is too slow (100ms delay each)
    => transferring a whole SDAI model at a time
- fast loading process
  => optimized arrangement, not like part21 or Java serialization
  => compact (not compressed) binary encoding
- Support references between entity instances
  - within the same Model
  - within the same Repository but different Models
  - within different Repositories
- Support references between entity instances with different underlying EXPRESS schemas.

Dynamic SdaiRepository (1)

In the SDAI standard the known repositories are static, they cannot change while a Session is open.

In the extended SDAI repositories become dynamic objects. For this new operations are introduced:
- Create Repository
- Delete Repository
- Link Repository
- Unlink Repository

In addition to **name** Repositories got a **location** attribute. During SDAI operations it may happen that a Repository is "known", but not its location. So it can't be opened.

Dynamic SdaiRepository (2)

- Access name and location
```
String name = r.getName();
String location = r.getLocation();
```
- Linking remote/external repositories, explicit and implicit
```
r = s.linkRepository("name","location");
r.unlinkRepository();
```
- Create and Delete repositories
```
r = s.createRepository("name","location");
r = s.importClearTextEncoding("name",
   "source_location","destination_location");
r.deleteRepository()
```


Dynamic SdaiRepository (3)

	Name	Location
temporarily	""	?
name from p21 file	null	?
take this name	"name"	?
local (default)	?	null
external / remote	?	"... path..."

Working with simple repository names:

- Import into a temporarily local repository:
  importClearTextEncoding("", "source-path", null)
- Take the repo-name from the part21 FILE_NAME:
  importClearTextEncoding(null, "source-path", null)
- Specify name and location (external and remote):
  createRepository("MyRepo", "c:\myrepos")
  createRepository("MyRepo", "//guest:passwd@192.123.22.11:1050")
  createRepository("MyRepo", "//user:passwd@server.lksoft.de")

Working with global unique repository names:

- Repository is a part of a domain:
  createRepository("//server.lksoft.de/MyRepo", user:passwd)
- In future:
  createRepository("//testrep.server.lksoft.de", user:passwd)

Which remote repositories are available on a remote location?

```
A_string as  =
        s.remoteRepositories("//user:passwd@server.lksoft.de");

for (int i = 0; i < as.getMemberCount(); i++) {
        System.out.println(as.getByIndex(i);
}
```

SdaiRepository - Exceptions

- RP_OPN : Repository open
- RP_NOPN : Repository is not open
- RP_DUP : Repository duplicate
- RP_LOCK : Repository locked by another user
- RP_NAVL : Repository currently not available
- RP_NEXS : Repository does not exist
- RP_RO : Repository is read-only
- LC_NVLD : Location invalid

SdaiRepository ~> SchemaInstance

- A repository stores SchemaInstances
```
ASchemaInstance asi = r.getSchemas();
SchemaInstance si = asi.getByIndex(n);
```
- A SchemaInstance is stored within a repository
```
SdaiRepository r = si.getRepository();
```
- Creation and deletion of SchemaInstances
```
ESchema_definition sd = ...;
si = r.createSchemaInstance("name", sd);
si = r.createSchemaInstance
   ("name", SAutomotive_design.class);
si.delete();
```

class SchemaInstance (1)

- The name is unique within the repository
```
String name = si.getName();
si.rename("NewSchemaInstanceName");
```
- A SchemaInstance is based on a schema
```
ESchema_definition sd = si.getNativeSchema();
String schema = si.getNativeSchemaString();
```


class SchemaInstance (2)

- Valid population defined by collection of models
```
ASdaiModel am = si.getAssociatedModels() ;
```
- associated models may be in the same or in different repositories
```
si.addSdaiModel(model1);
si.removeSdaiModel(model2);
```
- associated models may be based on the same or on different schemas, but domain_equivalent schemas.

SchemaInstance - Validation

- Uniqueness Validation
  int validateUniquenessRule(EUniqueness_rule rule, AEntity err)
  int validateUniquenessRule(String rule, AEntity err)
- Global rule Validation
  int validateGlobalRule(EGlobal_rule rule, AEntity err)
  int validateGlobalRule(String rule, AEntity err)
- Validate Reference Domain
  int validateInstanceReferenceDomain(AEntity err)
- CompleteValidation, updates validation info
  int validateSchemaInstance()

SchemaInstance - Information

- Validation Information,
  set by validateSchemaInstance()

```
String date = si.getValidationDate();
int level = si.getValidationLevel();//0-7
boolean result = si.getValidationResult();
boolean actual = si.isValidationCurrent();
```

- Change Information

```
if (si.testChangeDate()) {
   String date = si.getChangeDate();
}
```

Pre-Defined SchemaInstances

- Dictionary SchemaInstance with all dictionary models

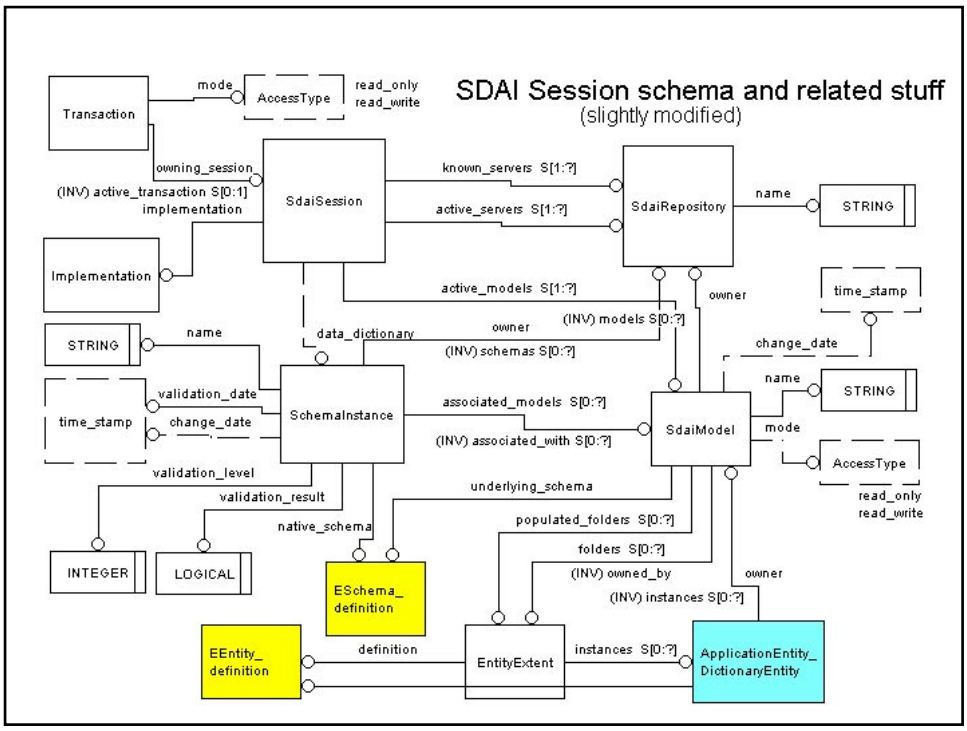
```
if (s.testDataDictionary()) {
   SchemaInstance si =
        s.getDataDictionary();
}
```

- Mapping SchemaInstance with all dictionary and mapping models

```
if (s.testDataMapping()) {
   SchemaInstance si = s.getDataMapping();
}
```


SchemaInstance - Exceptions

- SI_DUP : Schema instance duplicate
  Name must be unique within SdaiRepository
- SI_NEXS : Schema instance does not exist
  e.g. SI deleted or repository closed
- SI_LOCK : Schema instance locked
  Another application is going to modify the SI on a remote repository.



SdaiRepository ~> SdaiModel

- SdaiModels are stored in repositories:
```
ASdaiModel am = r.getModels();
SdaiModel m = am.getByIndex(n);
```
- An SdaiModel is stored within a repository
```
SdaiRepository r = m.getRepository();
```
- Creation and deletion of SdaiModels
```
ESchema_definition sd = ...;
m = r.createSdaiModel("name", sd);
m = r. createSdaiModel
    ("name", SConfig_control_design.class);
m.renameSdaiModel("newName"); // critical!
m.deleteSdaiModel();
```

SdaiModel - Exceptions

- MO_DUP : SDAI-model duplicate
- MO_NEXS : SDAI-model does not exist
- MO_NVLD : SDAI-model invalid
- MO_NDEQ : SDAI-model not domain equivalent
- MO_LOCK : Model locked by another user

SdaiModel - Header (1)

The header information is always accessible from a model, no matter if RW, RO access is started or not.

```
String name = m.getName();
SdaiRepository r = m.getRepository();
ESchema_definition schema =
               m.getUnderlyingSchema();
ESchema_definition schema =
               m.getDefinedSchema();
ASchemaInstance as = m.getAssociatedWith();
if (m.testChangeDate()) {
     String date = m.getChangeDate();
     long date = m.getChangeDateLong();
}
```

alternative: `if (m.isModified()) {... }`

SdaiModel - Header (2)

The attributes *language* and *context* are added to fully support the part21 Amendment

- `String lang = m.getDefaultLanguage();`
  `m.setDefaultLanguage("ger", true);`
- `A_string as = m.getContextIdentifiers();`
  `m.putContextIdentifiers(A_string value,`
  `boolean all)`

SdaiModel - Access Mode

- State of access mode
  ```
  switch(m.getMode()) {
  case SdaiModel.NO_ACCESS: ...
  case SdaiModel.READ_ONLY: ...
  case SdaiModel.READ_WRITE: ... }
  ```
- Start either RO or RW access
  `m.startReadOnlyAccess( );`
  `m.startReadWriteAccess( );`
- Change access mode between RO and RW
  `m.promoteSdaiModelToRW( );`
  `m.reduceSdaiModelToRO( );` // transaction !
- End access mode, take care of transaction !
  `m.endReadOnlyAccess( );`
  `m.endReadWriteAccess( );` // transaction !
- ASdaiModel am = s. getActiveModels();

SdaiModel - Access Mode Exceptions

- MX_NDEF : SDAI-model access not defined
- MX_NRW : SDAI-model access not read-write
- MX_RO : SDAI-model access read-only
- MX_RW : SDAI-model access read-write

SdaiModel - Contents (1)

- Accessing all instances of all types
```
int count = m.getInstanceCount();
AEntity ae = m.getInstances();
```
- Accessing all instances of a given *entity data types* (including subtypes)
  - Early binding:
```
int count = m.getInstanceCount(EXxx.class);
AXxx ax = (AXxx)
m.getInstances(EXxx.class);
```
  - Late binding:
```
EEntity_definition ed = ...; // xxx
int count = m.getInstanceCount(ed);
AXxx ax = (AXxx) m.getInstances(ed);
```

SdaiModel - Contents (2)

- Accessing all instances of a given *complex entity data type* (no subtypes)
  - Early binding:

```
int count =
 m.getExactInstanceCount(CXxx.class);
AXxx ax =
 (AXxx) m.getExactInstances(CXxx.class);
```

  - Late Binding:

```
EEntity_definition ed = ...; // xxx
int count = m.getExactInstanceCount(ed);
AXxx ax = (AXxx) m.getExactInstances(ed);
```


SdaiModel -> EntityExtent

EntityExtent helps to access entity instances of a given *Entity Data Type*. Only auxiliary purpose.

- Either get all or only the non-empty EntityExtents

```
AEntityExtent  aee = m.getFolders();
AEntityExtent  aee = m.
             getPopulatedFolders();
EntityExtent ee = aee.getByIndex(3);
```

- Attributes: definition, owner and instances

```
EEntity_definition ed = ee.getDefinition();
String name = ee.getDefinitionString();
SdaiModel m = ee.getOwnedBy();
AEntity ae = ee.getInstances();
AXxx ae = (AXxx) ee.getInstances();
```

SdaiModel ~> Entity Instances (1)

An Entity Instance exist only within an owning SdaiModel

- Late binding creation
```
EEntity_definition ed =
   m.getEntityDefinition("product");
EEntity p1 = m.createEntityInstance(ed);
```
- Early binding creation
```
EProduct p1  = (EProduct)
   m.createEntityInstance(CProduct.class);
```
- The owning model is always available
```
Model m1 = p1.findEntityInstanceSdaiModel();
```
- Delete also removes all references from active model
```
p1.deleteApplicationInstance();
```


SdaiModel ~> Entity Instances (2)

- Create a copy of an entity instance within the same or another model. All references remain on the original instance only.
```
EEntity p1 = m.copyApplicationInstance(p);
```
- Moving an instance to another model and/or changing the type (late and early).
  Within the same repo the persistent name is preserved.
  References from active models are maintained
```
EEntity pNew
 = m.substituteInstance(pOld);
 = m.substituteInstance(pOld, ed);
 = m.substituteInstance(pOld, CXxx.class);
```

Domain (1)

- A valid valid population of entity instances is defined by a SchemaInstance and its associated SdaiModels.
- Some operations need to know the domain where to search/access related instances. Depending on the application the domain may be defined for several SchemaInstances or only part of it. Within J-SDAI the domain is defined as an aggregate of SdaiModel (ASdaiModel).

```
get inverse attribute
usedin (late and early)
findEntityInstanceUsedin
findEntityInstanceUsers
findInstanceRoles
all mapping operations
```


Domain (2)

- Create a non-persistent Aggregate and add the desired SdaiModels

```
ASdaiModel domain = new ASdaiModel();
domain.addByIndex(model1);
```

- Access the domain from a single SchemaInstance

```
SchemaInstance si = ...;
domain = si.getAssociatedModels();
```

- Access the common domain of several SchemaInstances (set)

```
ASchemaInstance asi = ...;
domain = asi.getAssociatedModels();
```

Attribute Access

Operations to access the attributes of entity instance, available in late binding and in early binding

- `test attribute`
  does the attribute have a value or is it unset?
  If it is set what is the type - important for select types
- `get attribute`
  retrieves the actual value, either simple, entity instance or aggregate
- `set attribute`
  only for simple and entity instance values. Not for aggregates
- `create`
  only for attributes of aggregation type. The aggregates are create in place and cannot be assigned.
- `unset attribute`
  Makes test attribute to return unset. This is also if aggregate. You can't use set attribute for unset values.

Reading Attribute Value (late)

- Testing and accessing attributes

```
EDefined_type[] select = ...;
EAttribute attr = ...;
int iType = p.testAttribute(attr, select);
switch (iType) {
case 0:   ... // unset
case 1: Object o = p.get_object(attr); ...;
case 2:    int i = p.get_int(attr);...
case 3: double r = p.get_double(attr);... }
```

- For inverse attributes the domain where to search must be given. If *null* domain is local SdaiModel only.

```
EInverse_attribute ia = ... ;
ASdaiModel domain = ... ;
AEntity inverses = get_inverse(ia, domain);
```

Writing Explicit Attribute Values (late)

- Overloaded methods for double, int, Object,
```
p.set(attr1, 5.7, select);
p.set(attr2, 123, select);
p.set(attr3, "xxx", select);
p.set(attr4, entity, select);
```
- Aggregates can **never** be set, they are only created in place. Only one usage of aggregates!
```
Aggregate a =
        p.createAggregate(attr5, select);
```
- Every explicit (mandatory) attribute can be unset
```
p.unsetAttributeValue(attr1);
```
- Derived and inverse attributes are read-only, also explicit attributes redeclared as derived

Early Binding Attribute Access (1)

- The attribtute name is part of the access methods. The first parameter specifies the type of the Java Interface; this allows to solve name conflicts, always *null + casting*. Methods for *test, get, set/create, unset, usedin, attribute*:
-
```
public interface EAlpha extends EEntity {
   boolean testA1(EAlpha type);
   EKappa getA1(EAlpha type);
   void setA1(EAlpha type, EKappa value);
   void unsetA1(EAlpha type); ... }
```
-
```
public class CAlpha implements EAlpha { ...
   static int usedinA1(EAlpha type, EKappa p,
        ASdaiModel domain, AEntity result);
   static EAttribute attributeA1(EAlpha type);}
```
-
```
public interface ESigma extends EEntity {
   AIota createS0(ESigma type); ... }
```

Early Binding Attribute Access (2)

In case that the attribute is of a **select type** additional parameters are added to distinguish overloaded methods for different value types

- The approach is similar to the part21 solution
- The select cases are numbered. The ***test*** method returns the actual type (0-unset, 1-entity, 2, 3...).
- no parameter for simple entity types (== 1)
- for *Defined Data Type* this type is added (select1)
- several parameter for nested selecting of *Defined Data Types* (select2, select3 ...)
- Overloaded methods for *get, set/create*

Early Binding Attribute Access (3)

```
public interface EEpsilon extends EEntity {
        // constants and methods for SELECT attribute: e1
        int sE1Phi = 2;
        int sE1Psi = 3;
        int sE1Chi = 4;
        int sE1Tau = 5;
        int sE1Xi = 6;
        int sE1Pie = 7;
        int testE1(EEpsilon type);
        EEntity getE1(EEpsilon type) ; // case 1
        AEntity getE1(EEpsilon type, EPhi node1); // case 2
        AaEntity getE1(EEpsilon type, EPsi node1); // case 3
        A_double getE1(EEpsilon type, EChi node1); // case 4
        int getE1(EEpsilon type, ETau node1); // case 5
        int getE1(EEpsilon type, EXi node1); // case 6
        int getE1(EEpsilon type, EPie node1); // case 7  ... }
```

Diverse Entity Instance Operations

- Finding the EAttribute definition (dictionary)
```
EAttribute ea =
   p.getAttributeDefinition("attr1");
```
- Population dependent Bound of a REAL, INTEGER, or BINARY value
```
int bound = p.getAttributeValueBound(ea) ;
```
- Instance comparison
```
if (p1 == p2) {...}
```
- Value comparison as defined in EXPRESS (logical)
```
int i = p1.compareValues(p2);
```
- flat value comparison (non-recursive, shallow)
```
if (p1.compareValuesBoolean(p2)) {...}
```


Instance Identifiers / Persistent Label

- Every entity instance has an unchangeable persistent label which is assigned during its creation
```
String label = p.getPersistentLabel();
```
- J-SDAI use part21 type instance identifiers for this: "#" + positive 63 bit number, e.g. "#1234"
- J-SDAI ensures that a persistent label is used only once within a repository.
- Finding an entity instance within a repository
```
EEntity p = r.getSessionIdentifier("#1234");
```
- Some strings
```
String descr = p.getDescription();
String part21Line = p.toString();
```

Entity Type (1)

- Does an entity instance contain a Entity Data Type?
```
EEntity_definition ed = ...;
if (p.isKindOf(ed)) {...}
if (p instanceof ECartesian_point.class){}
```
- Is an entity instance exactly of a particular type?
```
if (p.isInstanceOf(ed)) {...}
if (p.getClass()== CCartesian_point.class){}
```
- Accessing the *Complex Entity Data Type*
```
EEntity_definition ed = p.getInstanceType();
Class c = p.getClass();
```
- Find the named_tpes, like EXPRESS TypeOf operation
```
ANamed_type result = ...;
p.findInstanceDataTypes(result);
```


Entity Type (2)

- For entity instances of the dictionary type
```
EEntity_definition p1, p2;
```
- Checking if one is a subtype of the other
```
if (p1.isSubtypeOf(p2)) {...}
```
- Are entities from different schema equivalent
```
if (p1.isDomainEquivalentWith(p2) {...}
```
No supported by SDAI because short-form solution
- Creation of new Complex Entity Data Types
```
ESchema_definition sd = ...;
AEntity_definition aed = ...;
EEntity_definition ed =
  s.getComplexEntityDefinition(aed,sd);

Class ac[] = { EBbb.class, ECcc.class };
Class c = s.getComplexEntityDefinition(ac);
```

Relating Entity Instances with App. Data

- While a model is active, it's entity instances can be associated with application specific data of any kind.
- The application is responsible for the data, not J-SDAI. The association is lost when access ends.

```
Object o = ... ; // from Application
p.setTemp(o);
...
Object o = p.getTemp()
```

- A common place to exchange entity instances

```
AEntity ai = s. getClipboard();
```

Applications can clear, add, remove instances

Related instances

- Find all referencing instances within a domain

```
ASdaiModel domain = ...;
AEntity users = new AEntity();
p.findEntityInstanceUsers(domain, users);
```

- Find all referencing instances of a particular role within a domain

```
EAttribute role = ...;
p.findEntityInstanceUsedin(role,domain,users);
```

- Find all roles by which this instance is referenced

```
AAttribute result = ...;
p.findInstanceRoles(domain, result);
```

Entity Instance Validation

Validation of the attributes and rules of a entity instance

- validateWhereRule(EWhere_rule rule)
- validateRequiredExplicitAttributesAssigned(AAttribute nonConf)
- validateExplicitAttributesReferences(AAttribute nonConf)
- validateInverseAttributes(AAttribute nonConf)
- validateAggregatesSize(AAttribute nonConf)
- validateAggregatesUniqueness(AAttribute nonConf)
- validateArrayNotOptional(AAttribute nonConf)
- validateStringWidth(AAttribute nonConf)
- validateRealPrecision(AAttribute nonConf)
- validateBinaryWidth(AAttribute nonConf)

Aggregate Interface and Classes

- One fits all: SET, BAG, LIST or ARRAY
- Interface implemented by all aggregate classes except
  `ASdaiRepository`
  `ASchemaInstance`
  `ASdaiModel`
  `AEntityExtent`
- Single (A) and nested Aggregates (Aa...a)
- Specialized aggregates for primitive types
  `A_binary, Aa_binary, Aaa_binary,`
  `A_boolean, Aa_boolean, Aaa_boolean,`
  `A_double, Aa_double, Aaa_double,`
  `A_enumeration, Aa_enumeration, Aaa_enumeration`
  `A_integer, Aa_integer, Aaa_integer,`
  `A_string, Aa_string, Aaa_string`

Early Binding Aggregate classes

- For all *Entity Data Types* XXX
  single: **A**Xxx
  nested as needed: **Aa**Xxx, **Aaa**Xxx ...
- For some *Select Data Types* (as needed) SSS
  single: **A**Sss
  in extreme cases also nested: **Aa**Sss
  Accessing members of different type by overloaded methods, similar to part21 encoding (see select for early binding attribute access)

Iterating and indexing Aggregates

The members of an aggregate can be accessed either by an iterator or by index:

- RO and RW access with SdaiIterator

```
SdaiIterator iter =
agg.createIterator();
value = agg.getCurrentMember(iter);
```

- RO access by index for all types, RW only for LIST and ARRAY.

```
value = agg.getByIndex(5);
```

Aggregate creation

- Entity instance attribute (late and early)

```
Aggregate a =
    p.createAggregate(attr5, select);
AXxx = p.createAttrib1(null);
```

- Nested Aggregates

```
createAggregateAsCurrentMember
createAggregateAfter
createAggregateBefore
createAggregateByIndex
addAggregateByIndex
createAggregateUnordered
```

- User aggregates (*Non Persistent Lists*)

```
AXxx ax = new AXxx();
ASdaiModel am = new ASdaiModel();
```


Aggregate Access Rights

- Session aggregates are RO. See the special operations to modify them. Exception when creates with *new*.

```
ASdaiRepository, ASchemaInstance,
ASdaiModel, AEntityExtent,
SdaiModel.getInstances(...);
SdaiModel.getExactInstances(...);
EntityExtent.getInstances();
```

- All data in the special "*SystemRepository*" repository containing dictionary and mapping data is read-only
- Aggregates in application data is RW if RW-transaction and RW model access is started
- *Non Persistent List* aggregates are always RW

Aggregate: General and Bounds

General	acc	BAG	SET	LIST	ARRAY
createIterator	I	x	x	x	x
attachIterator	I	x	x	x	x
clear	-	x	x	x	x
getAggregationType	-	x	x	x	x
Size, Bounds					
getMemberCount	-	x	x	x	x
getLowerBound	-	x	x	x	x
getUpperBound	-	x	x	x	x
getLowerIndex	-	-	-	-	x
getUpperIndex	-	-	-	-	x
SdaiIterator.getValueBound	I	x	x	x	x
getValueBoundByIndex	i	x	x	x	x

Aggregate: Reading and Writing members

Reading members	acc	BAG	SET	LIST	ARRAY
testCurrentMember	I	x	x	x	x
testByIndex	i	x	x	x	x
isMember	-	x	x	x	x
getCurrentMember	I	x	x	x	x
getByIndex	i	x	x	x	x
Writing members					
setCurrentMember	I	x	x	x	x
setByIndex	i	-	-	x	x
SdaiIterator.unset	I	-	-	-	x
unsetValueByIndex	i	-	-	-	x

Aggregate: add, remove, create members

Adding and removing		BAG	SET	LIST	ARRAY
addAfter	I	-	-	x	-
addBefore	I	-	-	x	-
addUnordered	-	x	-	-	-
SdaiIterator.remove	-	x	x	x	x
removeByIndex		-	-	x	-
removeUnordered	-	x	x	-	-
Creating new sub-Aggregates					
createAggregateAsCurrentMember	I	x	x	x	x
createAggregateAfter	I	-	-	x	-
createAggregateBefore	I	-	-	x	-
createAggregateByIndex	i	-	-	x	x
addAggregateByIndex	i	-	-	x	-
createAggregateUnordered	-	x	x	-	-

Aggregate: special

special on SdaiIterator	acc	BAG	SET	LIST	ARRAY
SdaiIterator.beginning	I	x	x	x	x
SdaiIterator.end	I	-	-	x	x
SdaiIterator.next	I	x	x	x	x
SdaiIterator.previous	I	-	-	x	x
special on ARRAYs					
reindexArray	-	-	-	-	x
resetArrayIndex	-	-	-	-	x

Query

- Source where to search
  - SdaiRepository
  - SchemaInstance
  - SdaiModel
  - ASdaiModel
- Expression
  spec: attribute.name, {...}
  operator: =, <=, =>, <, >, :=:, :<>:, IN, LIKE
  value or UNSET
- Result added to AEntity, return count
- Not implemented in J-SDAI 2.0

Import / Export of ISO 10303-21

Clear Text Encoding of the Exchange Structure

- Technical Corrigendum (TC1) fix select bug.
  Amendment extends for multiple data sections.
- Originally part21 and SDAI are not related
- The complete Java Binding defines a mapping based on SdaiRepository. A mapping to SchemaInstance would also be a good alternative.
- SDAI is more powerful. Import is always possible but export is restricted because of:
  - missing schema_instance (domain, validation)
  - references into other repositories

Mapping between part21 and SDAI

complete Exchange Structure	SdaiRepository
FILE_NAME	SdaiRepository.name
Data-Section	SdaiModel
instance identifier	Persistent Label

Repository / part21 attributes

```
ENTITY file_description;
    description : LIST [1:?] OF STRING (256) ;      -> description
    implementation_level : STRING (256) ;          -> "2.1" or "3.1"
END_ENTITY;

ENTITY file_name;
    name : STRING (256) ;                          -> name
    time_stamp : STRING (256) ;                    -> timeStamp
    author : LIST [ 1 : ? ] OF STRING (256) ;      -> author
    organization : LIST[ 1 : ? ] OF STRING (256) ; -> organization
    preprocessor_version : STRING (256) ;          ->
preProcessorVersion
    originating_system : STRING (256) ;            ->
originatingSystem
    authorization : STRING (256) ;                 -> authorization
END_ENTITY;
```

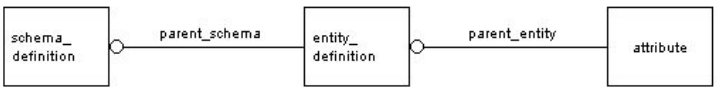
Repository / part21 attributes

String getAuthorization()
setAuthorization(java.lang.String value)
A_string getAuthor()
A_string getDescription()
A_string getOrganization()
String getOriginating_system()
setOriginating_system(java.lang.String value)
String getPreprocessor_version()
setPreprocessor_version(java.lang.String value)

suppressShortNames(), allowShortNames()

SDAI_dictionary_schema: Principles

Meta model and data:



#10=SCHEMA_DEFINITION('test_schema',...);
#20=ENTITY_DEFINITION('A', ..., #10, ...);
#30=ENTITY_DEFINITION('B', ..., #10, ...);
#40=ATTRIBUTE('x', ..., #30, ...);

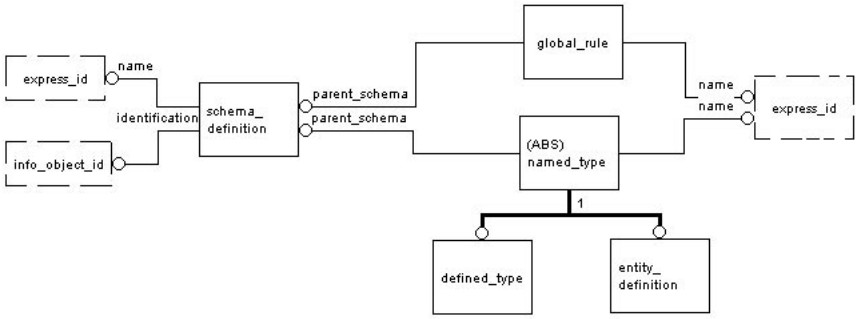
Application model (test_schema) and data:

#80=A( );
#90=B(#80),

SDAI_dictionary_schema: Overview

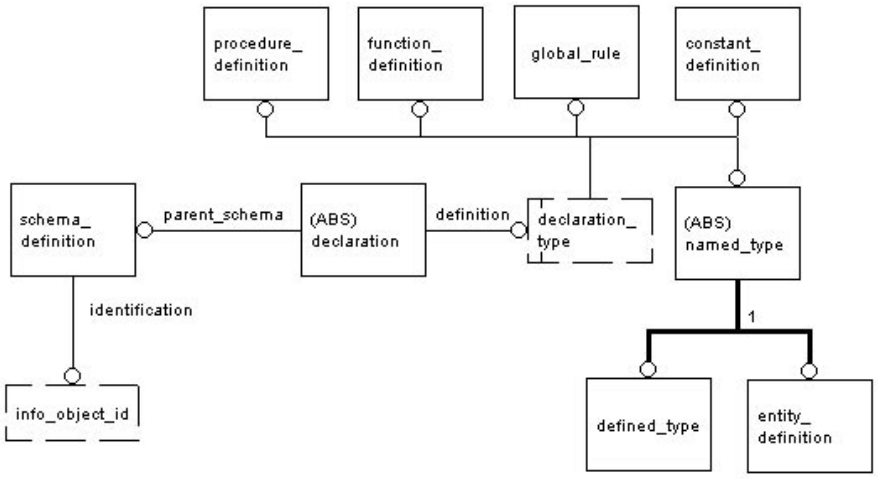
- Semantic Meta model of the EXPRESS language
- Alternative representation of EXPRESS schemas.
- The SDAI_dictionary_data itself can be represented as a population of the SDAI_dictionary_schema.
- It is a kernel part of J-SDAI in package *jsdai.dictionary*
- Basis for *Late Binding* entity and attribute access.
- In J-SDAI the SDAI_dictionary_schema is slightly modified for explicit support of short form schemas.
- The SDAI_dictionary_schema will probably be substituted by a unified Semantic Meta Model for EXPRESS when available.

SDAI_dictionary_schema: Original

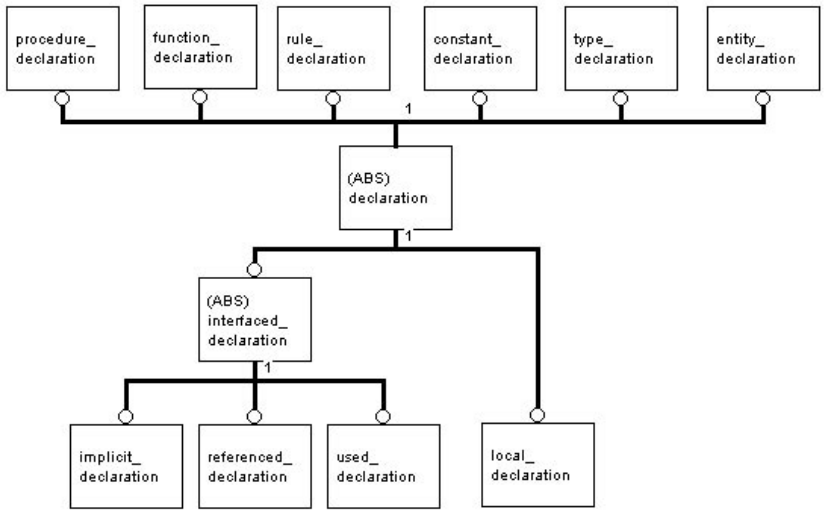


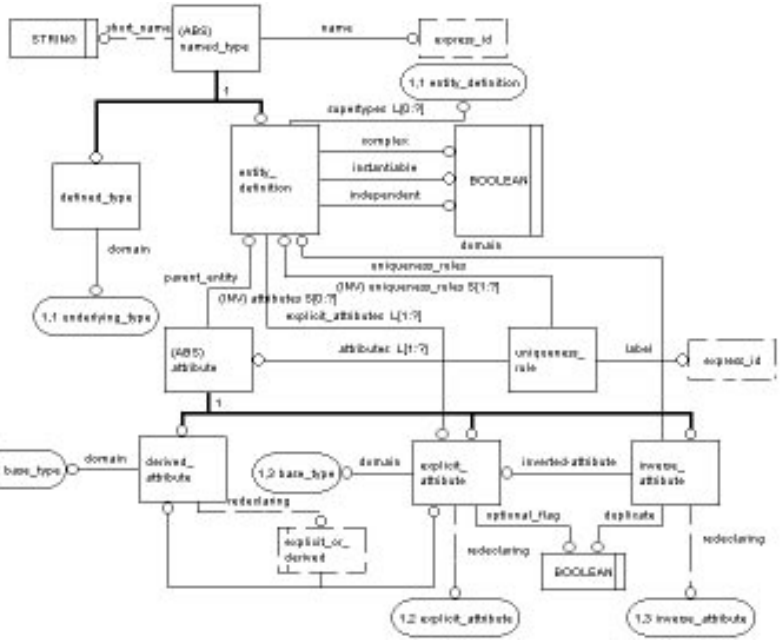
entity_definitions cannot be explicitly interfaced into other schemas

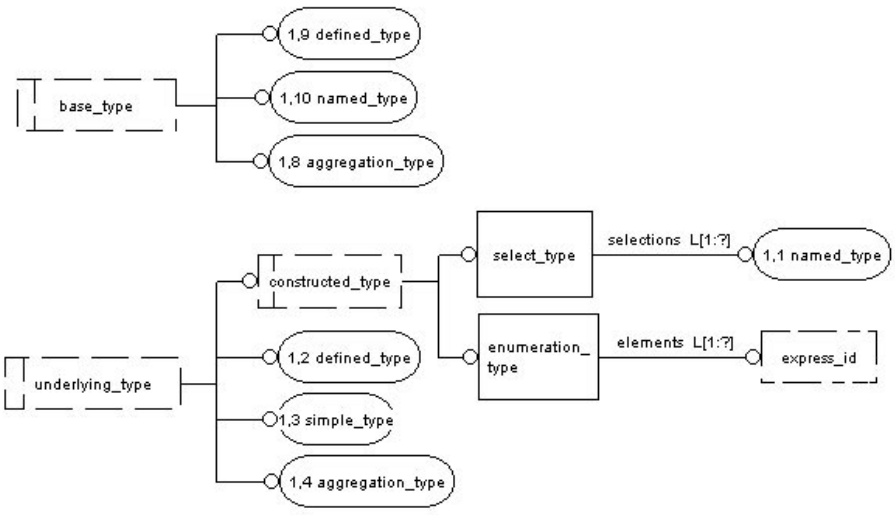
SDAI_dictionary_schema: Modified

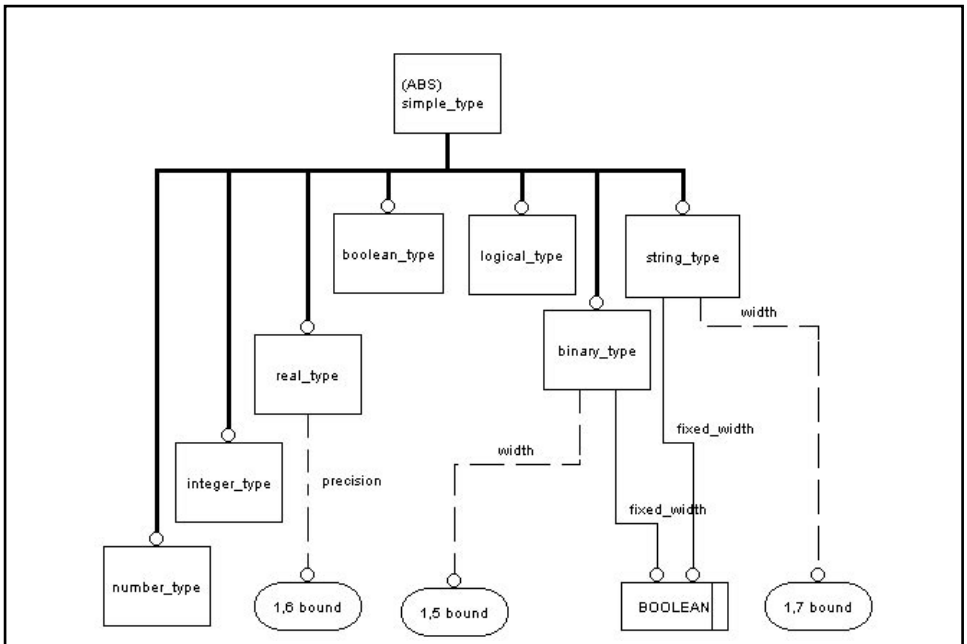


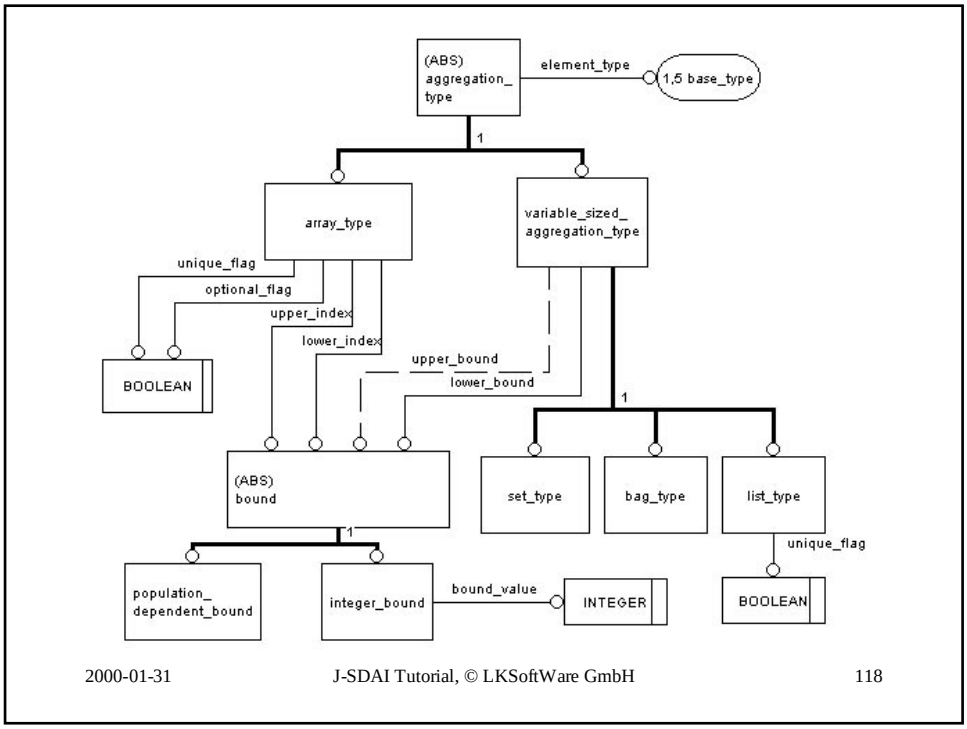
SDAI_dictionary_schema: Declarations

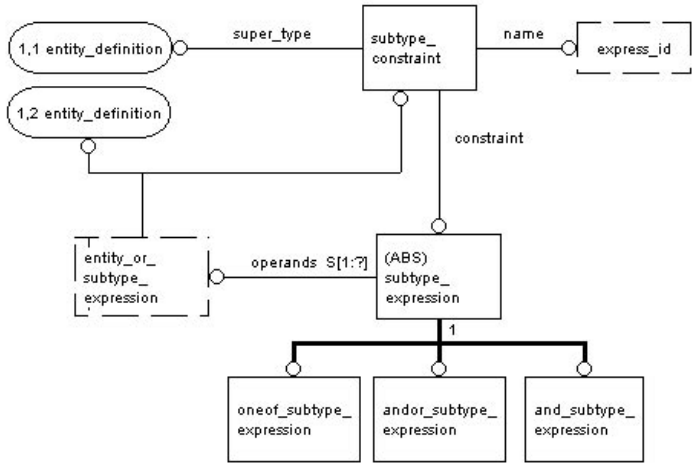



STRING
short_name
(ABS)
named_type
name
express_id
1,1 entity_definition
supertypes L[0:?]
defined_type
entity_
definition
complex
instantiable
independent
BOOLEAN
domain
domain
parent_entity
uniqueness_rules
(INV) uniqueness_rules S[1:?]
1,1 underlying_type
(INV) attributes S[0:?]
explicit_attributes L[1:?]
(ABS)
attribute
attributes L[1:?]
uniqueness_
rule
label
express_id
1
1,2 base_type
domain
derived_
attribute
redeclaring
1,2 base_type
domain
explicit_
attribute
inverted_attribute
inverse_
attribute
optional_flag
duplicate
redeclaring
redeclaring
explicit_or_
derived
BOOLEAN
redeclaring
1,2 explicit_attribute
1,3 inverse_attribute


base_type
1,9 defined_type
1,10 named_type
1,8 aggregation_type
underlying_type
constructed_type
1,2 defined_type
1,3 simple_type
1,4 aggregation_type
select_type
selections L[1:?]
1,1 named_type
enumeration_
type
elements L[1:?]
express_id


(ABS)
simple_type
boolean_type
logical_type
string_type
width
real_type
binary_type
fixed_width
width
integer_type
precision
fixed_width
number_type
1,6 bound
1,5 bound
BOOLEAN
1,7 bound

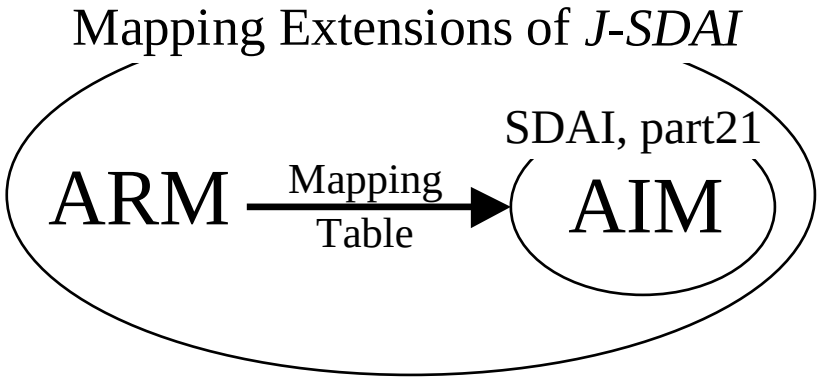

(ABS)
aggregation_
type
element_type
1,5 base_type
1
array_type
variable_sized_
aggregation_type
unique_flag
optional_flag
upper_index
lower_index
upper_bound
lower_bound
BOOLEAN
1
(ABS)
bound
set_type
bag_type
list_type
1
unique_flag
population_
dependent_bound
integer_bound
bound_value
INTEGER
BOOLEAN



SDAI_dictionary_schema: Summery

- It fulfills the primary needs of SDAI (late binding)
- Added "declaration" entities allows full short form support, needed for interoperable schemas
- Added sub-super type constraints (compatible to E-Amendment)
- It lacks modeling of
  - expressions, statements, constants and variables needed for the body of functions and procedures
  - splitting of entity data type and complex entity data type
  - simple types should be represented as constant instances, not as entities.

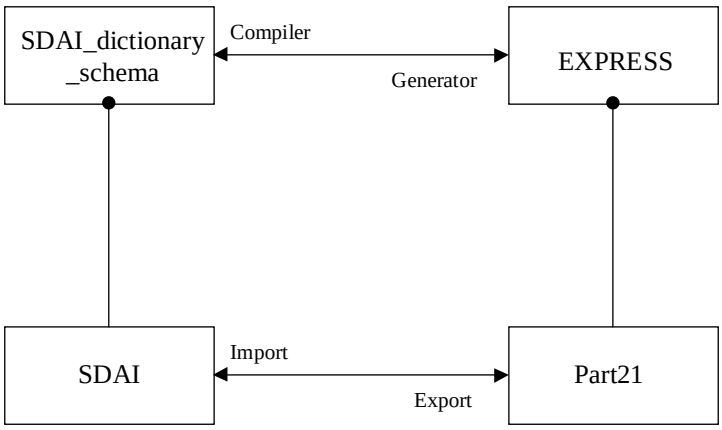
Structure of STEP Application Protocols

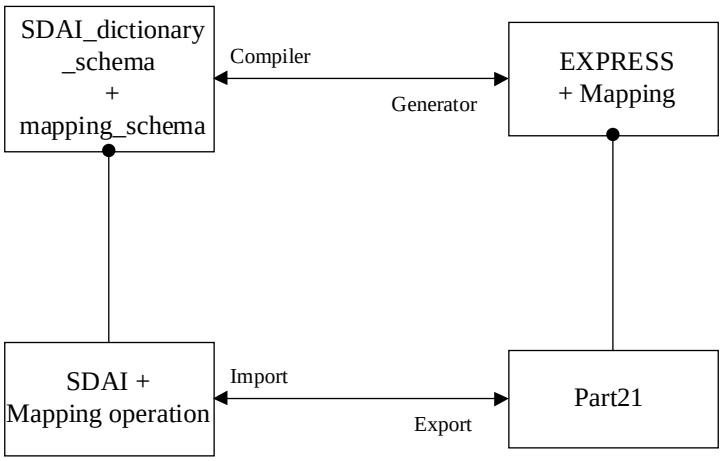


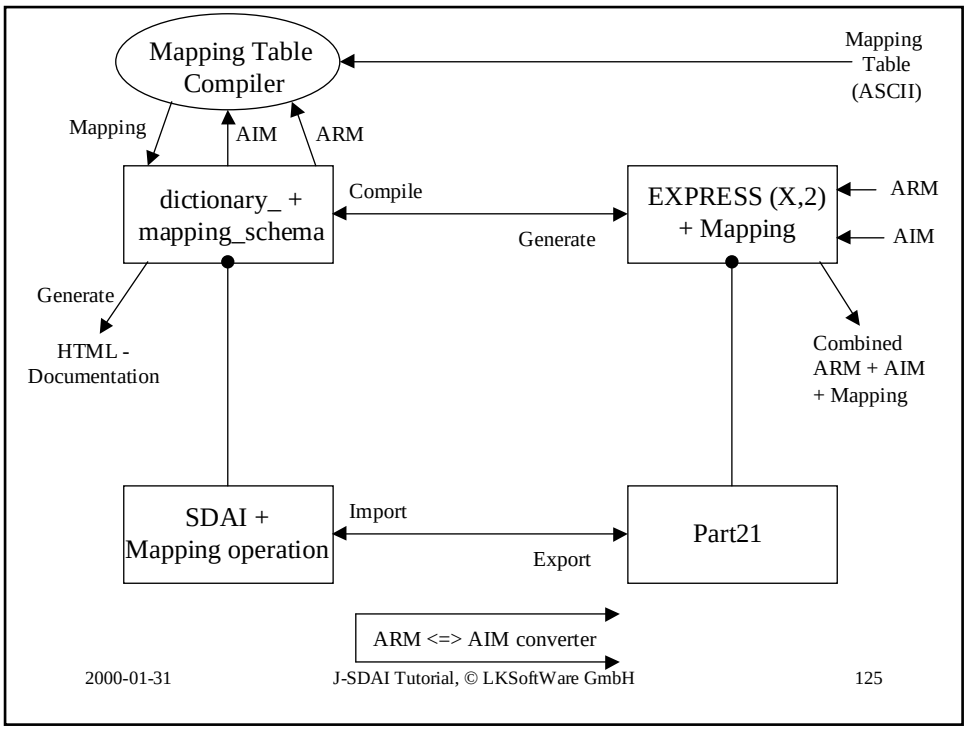
Mapping Schema and Mapping Operations, History

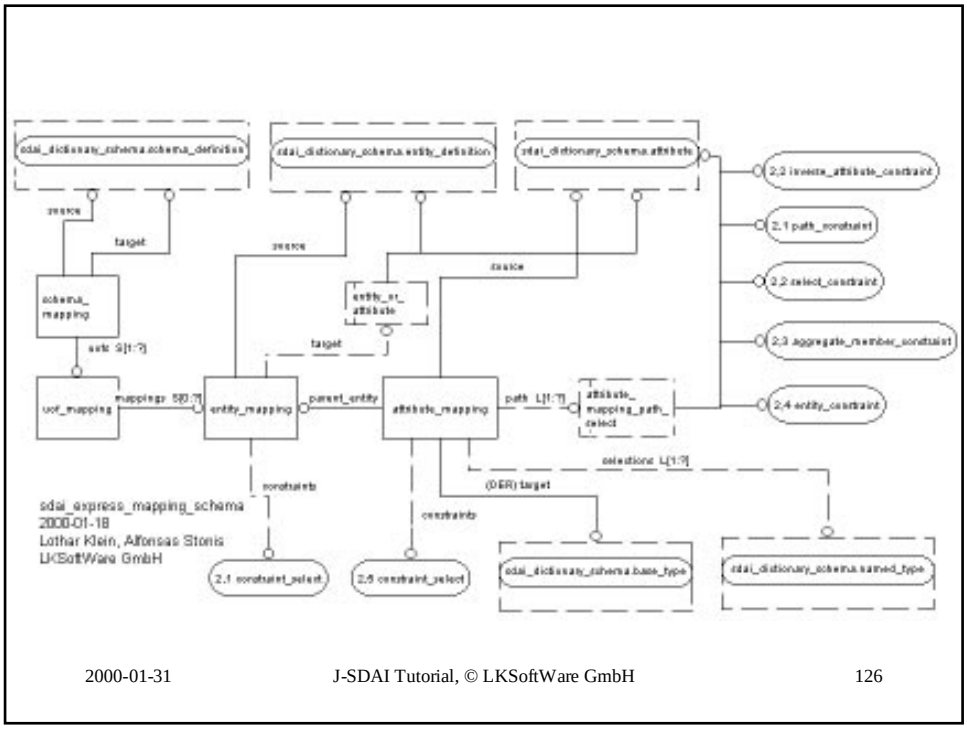
- The mapping extensions of the SDAI_dictionary_schema and associated operators
  - WG11/N050 SDAI Mapping Schema
  - WG11/N051 SDAI Mapping Schemas II
  - WG11/N068 Requirements for the next version of SDAI
  - WG11/N075 Mapping Operations for the SDAI
- Data conversion: AIM <=> ARM <=> EPISTLE
  but not: AIM <=> EPISTLE

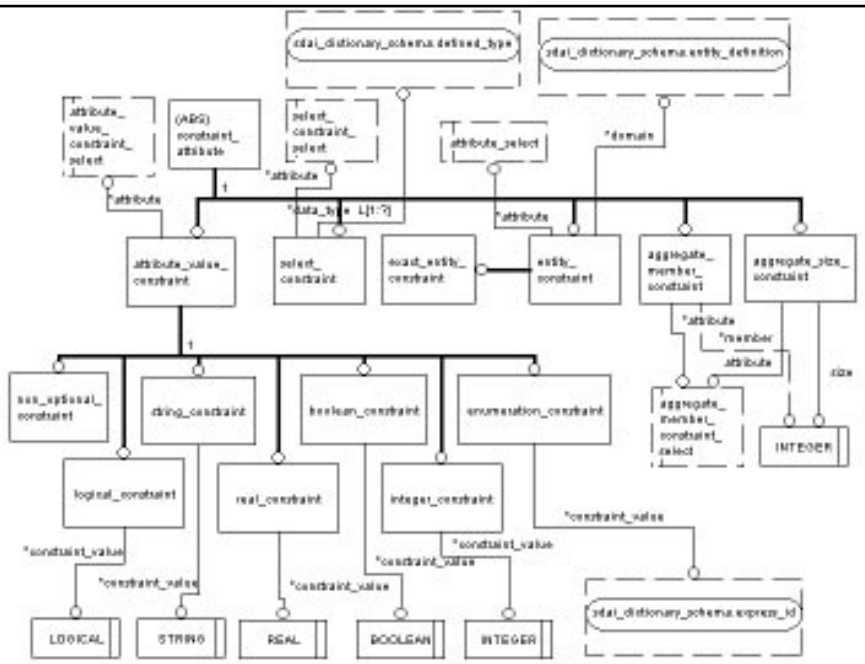
Dependencies of the STEP Description and Implementation Methods

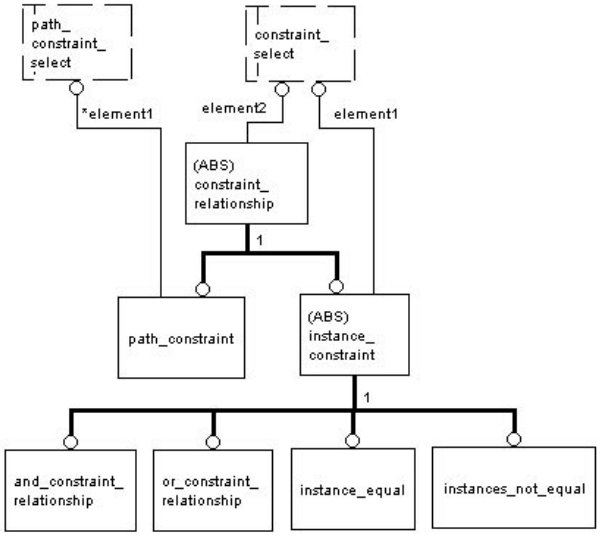





Mapping Table Compiler
Mapping Table (ASCII)
Mapping
AIM
ARM
dictionary_ + mapping_schema
Compile
Generate
EXPRESS (X,2) + Mapping
ARM
AIM
Generate
HTML - Documentation
Combined ARM + AIM + Mapping
SDAI + Mapping operation
Import
Export
Part21
ARM <=> AIM converter


sdai_express_mapping_schema
2000-01-18
Lothar Klein, Alfonsas Stonis
LKSoftWare GmbH


sdai_dictionary_schema.defined_type
sdai_dictionary_schema.entity_definition
attribute_value_constraint
select_constraint
exact_entity_constraint
entity_constraint
aggregate_member_constraint
aggregate_size_constraint
non_optional_constraint
string_constraint
boolean_constraint
enumeration_constraint
logical_constraint
real_constraint
integer_constraint
INTEGER
LOGICAL
STRING
REAL
BOOLEAN
INTEGER
sdai_dictionary_schema.express_id


path_constraint_select
constraint_select
*element1
element2
element1
(ABS) constraint_relationship
path_constraint
(ABS) instance_constraint
and_constraint_relationship
or_constraint_relationship
instance_equal
instances_not_equal

Mapping Operations (1)

Allow to work on AIM instances with ARM types

- Require dictionary data for ARM, AIM and mapping data => late-binding
  (Early binding also possible, but many classes and code)
- Read operations:
  - find instances which fits to a given ARM type
  - test if instance fits to an ARM type
  - test ARM attribute
  - get ARM value
- Write operations:
  - create AIM instances for ARM type
  - set/unset ARM attribute values on AIM instance
- Mapping methods are defined in SdaiModel and EEntity

Mapping Operations (2)

- Take care of all constraints in the reference path.
  User can ignore the reference path column.
- Define a higher level API on top of the lower level SDAI
- User need to know only column 1 -*Application element*- and 2 -*AIM element*-.
- Are limited to the quality of the Mapping Tables
- The read-operations (AIM to ARM) find **all** solutions, either 0, 1 or many. Often not only one solution.
- Extensions are needed to cover Conformance Classes and top - entities from where to start searching; e.g. product_concept

Mapping Operations (3)

Example for testing and accessing ARM

```
EAttribute armAttribute = ...;
ASdaiModel dataDomain = ...;
ASdaiModel mappingDomain = ...;
AAttribute_mapping attAlternatives =
     p.testSourceAttribute(armAttribute,
               dataDomain, mappingDomain);
if (attAlternatives != null) {
     Object o =
          p.getMappedAttribute(armAttribute,
               dataDomain, mappingDomain);
}
```


Memory Limitations

- J-SDAI stores the persistent information of repositories and models on disk
- The contents of a closed repository is stored on disk only.
- The contents of an in-active SdaiModel (no RO/RW access) is also stored on disk only.
- When opening a repository its models and SchemaInstances are loaded into RAM
- When activating a model in RO or RW access mode its entity instances are loaded into RAM

Memory (2)

The available RAM limits the number of available entity instances at a time. Consequences for practical works:

- Don't make SdaiModels too big. Several of them should always fit into RAM
- De-active models when not needed (end access)
- General, don't keep pointers to objects you don't need:
  - set them to null when no longer needed
  - minimize data fields in classes - define as many fields as possible in methods (stack)

Memory (2)

- A 128 MByte machine can typically handle 250,000 entity instances or more **simultaneously**.
- This is sufficient for many applications. The problem is more that the STEP-APs and IRs give no guidance how to use SdaiModels (devide and conquer)
- e.g. an A-BREP into one model

SdaiEvent and -Listeners (1)

Applications may want to be informed if an SDAI object is changed either through local activities or from remote actions.

- Objects of class `SdaiEvent` inform on changes
- SDAI classes implementing the interface `SdaiEvenSource` can send out SdaiEvents
- Application classes implementing `SdaiListener` can receive SdaiEvents when having registered at `SdaiEventSources` as listener.

SdaiEvent and -Listeners (2)

Currently (J-SDAI 2.0) the interface `SdaiEvenSource` is implemented by:

- `EEntity` the basis for all application and dictionary instances
- `SdaiRepository`

For future versions we plan to have `SdaiEventSource` also implemented by:

- All kind of `Aggregates`
- `SdaiModel`
- `SchemaInstance`

SdaiEvent and -Listeners (3)

`SdaiEvent` inform on these modifications:

- ATTRIBUTE_MODIFIED
- INVALID
- MEMBER_ADD
- MEMEBER_REMOVE
- SERVER_DATA_CHANGED
- Other SdaiEvent types may be considered for the future

SdaiEvent and SdaiListener define an information system on pur SDAI basis - it is on persistend data. It can be a replacement for other Java APIs such as InfoBus for this specific purpose

Other possible part27 implementations

Dynamic loading of only the needed entity instances from disk/network

- handling unlimited number of entity instances
- Implementation cannot know which data will be needed. Therefor entity instances are only loaded on request, e.g. with *get attribute* methods. This will significantly slow down
- Operations like SdaiModel.getInstances or EntityExtent.getInstances are not useful. How to start initially (-> query)
- For every get Attribute access if is checked if this instance is already loaded

EXPRESS support for Modularization and it's usage in STEP today

EXPRESS supports modular data modeling by interfacing elements from other schemas into a local one. Elements are interfaced either

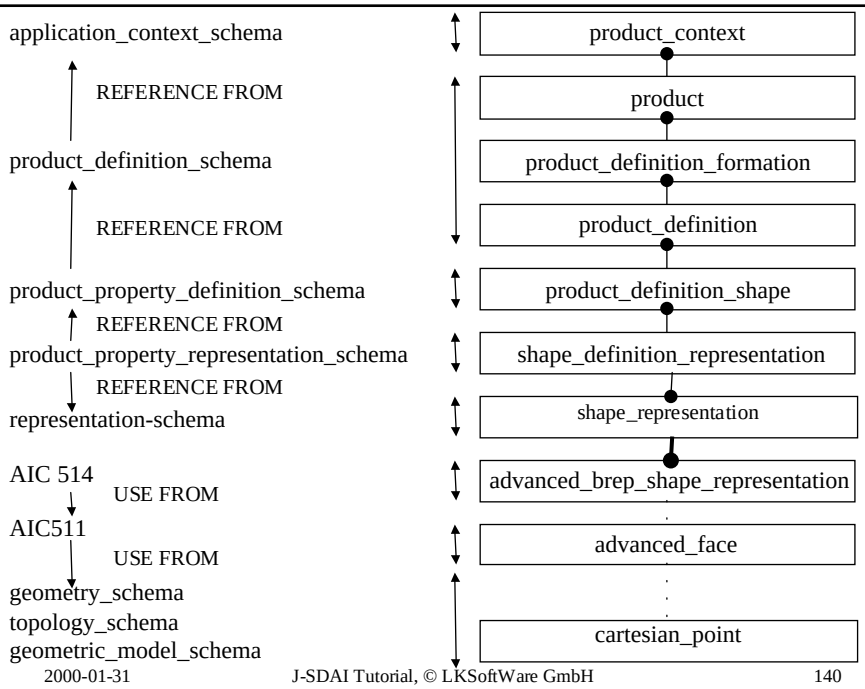
- **explicitly** with **REFERENCE FROM** or **USE FROM** or
- **implicitly** when used by explicitly interfaced elements.

Within the context of a schema only locally declared entities or entities interfaced with USE FROM can be individually instantiated. Instances of entities which are interfaced either implicitly or with REFERENCE FROM are only allowed if they are referenced from locally defined or USE FROM entity instances.

Today STEP consists of 66 schemas in the Integrated Resources (IR) and Application Interpreted Construct (AIC). While the IR schemas interfaces elements from other IRs with REFERENCE FROM the AIC do this with USE FROM.

Every Application protocols (AP) have a short form and a long form schema. The short form schema interfaces elements from the integrated and interpreted schemas with USE FROM. In the long form schema all interfaced elements are resolved in one global schema. So far STEP implementations are based on the long-form schemas.

For AP-interoperability and for component based programming (beans) the usage of the AP short form is essential !



Problems with short form implementation

The proposed p21 amendment allows more than one Data-Section.
A Data-Section in a p21 file or an SdaiModel is based on one EXPRESS schema.

The Data-Section /SdaiModel can contain only instances of entities defined in its underlying schema or of interfaced entities. What to do with AP specific subtypes?

- Exclude them from AP interoperability
- Use CONNOTATIONAL subtypes (Express V2)
- allow foreign subtypes in Data-Section / SdaiModel

How many instances shall a Data-Section / SdaiModel contain?
E.g. for the application_context_schema typically only 4 instances are needed.
But with too few instances the overhead of having a separate Data-Section / SdaiModel is too big.
A complete assembled product with all shapes can easily consists of several millions of instances.
There is a practical upper limit of less than one million instances for an SdaiModel.
Otherwise working on repositories with billions of instances, referencing each other will be hard to implement and inefficient. (Optimal less than 100,000 instances per SdaiModel)
Guidelines are needed for practical / typical sizes of Data Sections / SdaiModels.

Are Data-Sections / SdaiModel needed for every schema or only for the major ones?
Guidelines are needed for which schemas to use,
e.g. AIC schemas and product_property_representation_schema,
but probably not application_context_schema

Some hot topics from our perspective

- Implementations based on EXPRESS short forms:
  It makes us a lot of work to get mating sets of all the EXPRESS schemas - and we have to update the old APs (203) for the AICs ourselves.
- Identifying the possible complex entity data types with several leaves. These are those which require external mapping in a part21 file.
  Today's STEP schemas allow an astronomic big number of those complexes - but only a few are really relevant and are supported by applications.
- Making mapping tables (clause 5.1 of APs) and conformance classes (4.1 of APs) computer readable.

Topics: Future SDAI Directions

- Effects of the EXPRESS Amendment and EXPRESS-2 on SDAI
- How the EXPRESS-2 semantic Meta-model will influence the SDAI_dictionary_schema
- PLIB 32, the Geometric Programming interface for Java
- How SDAI may support Parametrics
- What SDAI can do for the special needs of handling life cycle information as defined in STEP-AP221, ISO 15926, EPISTLE