Consolidation of the Generator Infrastructure MDGEN – Model Driven Generation

Date: October 16th, 2012
Produced by: Mario Lovisi / Serano Colameo
Version: 1.0

Agenda

▪ Initial Situation / Goal of the Project	(Mario Lovisi)
▪ The Multi Channel Platform (MCP)	(Mario Lovisi)
▪ IcmDsl & MDGEN Features	(Serano Colameo)
▪ Q&A	(Mario Lovisi / Serano Colameo)

Initial Situation

Generator	Artifacts	Model	Technology
TOOLS GenCAL	EJB 2.x conform Java Code for Standard, FN and MCP Services	Java Code	Velocity and Recoder Framework
JBSGEN	EJB 2.x conform Java Code for FN Classic Services	XML / DTD	oAW 3.x (Xpand only)
Data Service Framework	EJB 3.x conform Java Code and Persistence Layer stuff for FN	XML / XSD	oAW 4.x (Xpand and Xtend 1.x)
…	…	…	…

Initial Situation

Generator	Artifacts	Model	Technology
TOOLS GenCAL	EJB 2.x conform Java Code for Standard, FN and MCP Services	Java Code	Velocity and Recoder Framework
JBSGEN	EJB 2.x conform Java Code for FN Classic Services	XML / DTD	oAW 3.x (Xpand only)
Data Service Framework	EJB 3.x conform Java Code and Persistence Layer stuff for FN	XML / XSD	oAW 4.x (Xpand and Xtend 1.x)
…	…	…	…

Goal of the Project

- Unify existing Generator Technologies in one Technology (Xtext)
- To be able to include existing code (Java) and models (e.g. IDL)
- Import existing interface definitions (IDL) into the new DSL (ICM)
- Generate for MCP more artifacts (Façade Implementation, Data Mappers, Transfer Object Handler, Commands etc.) as before
- Allow a “model-driven” migration (e.g. EJB2 => EJB3, JAP etc.)

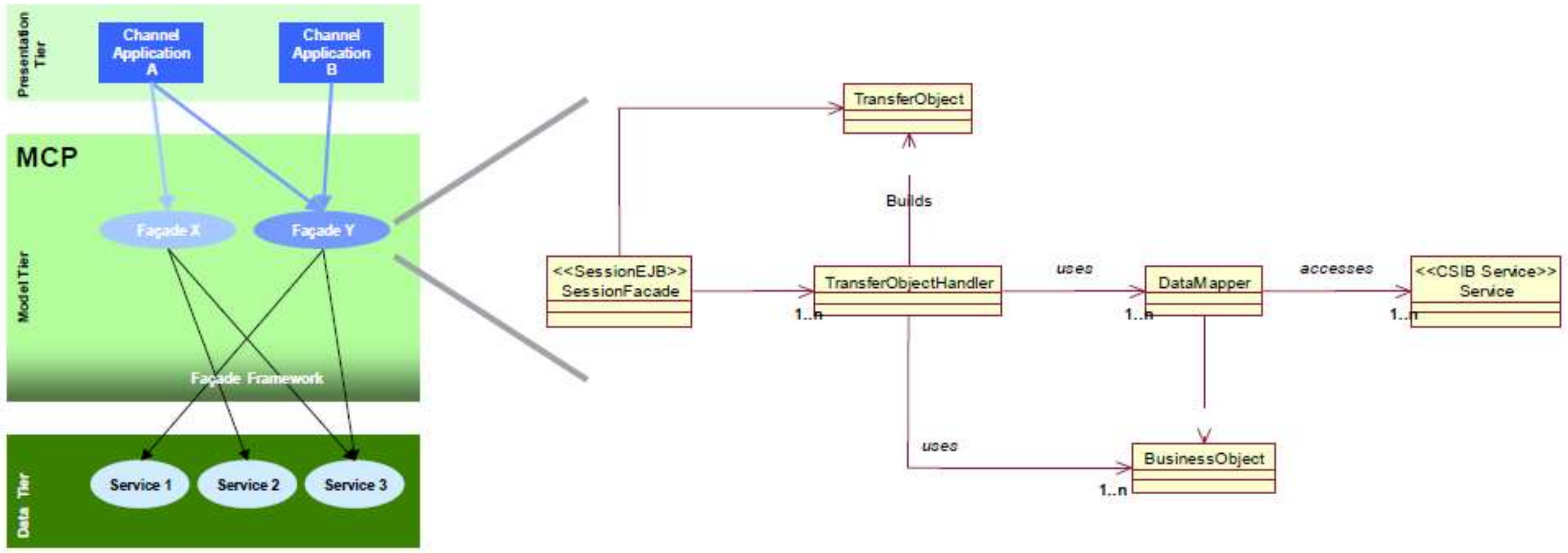
What is the Multi Channel Platform?

MCP is a pragmatic Cooperation Model for Business to increase effectiveness and synergies by:

- aligning business needs across channels
- aligning new IT initiatives concerning channel application within business
- reusing existing business functionality
- ensure consistency of functionality and data across channels
- an Integration Platform for IT to increase efficiency by sharing functionality, expertise and resources.

MCP Façade – Technology Architecture Concept

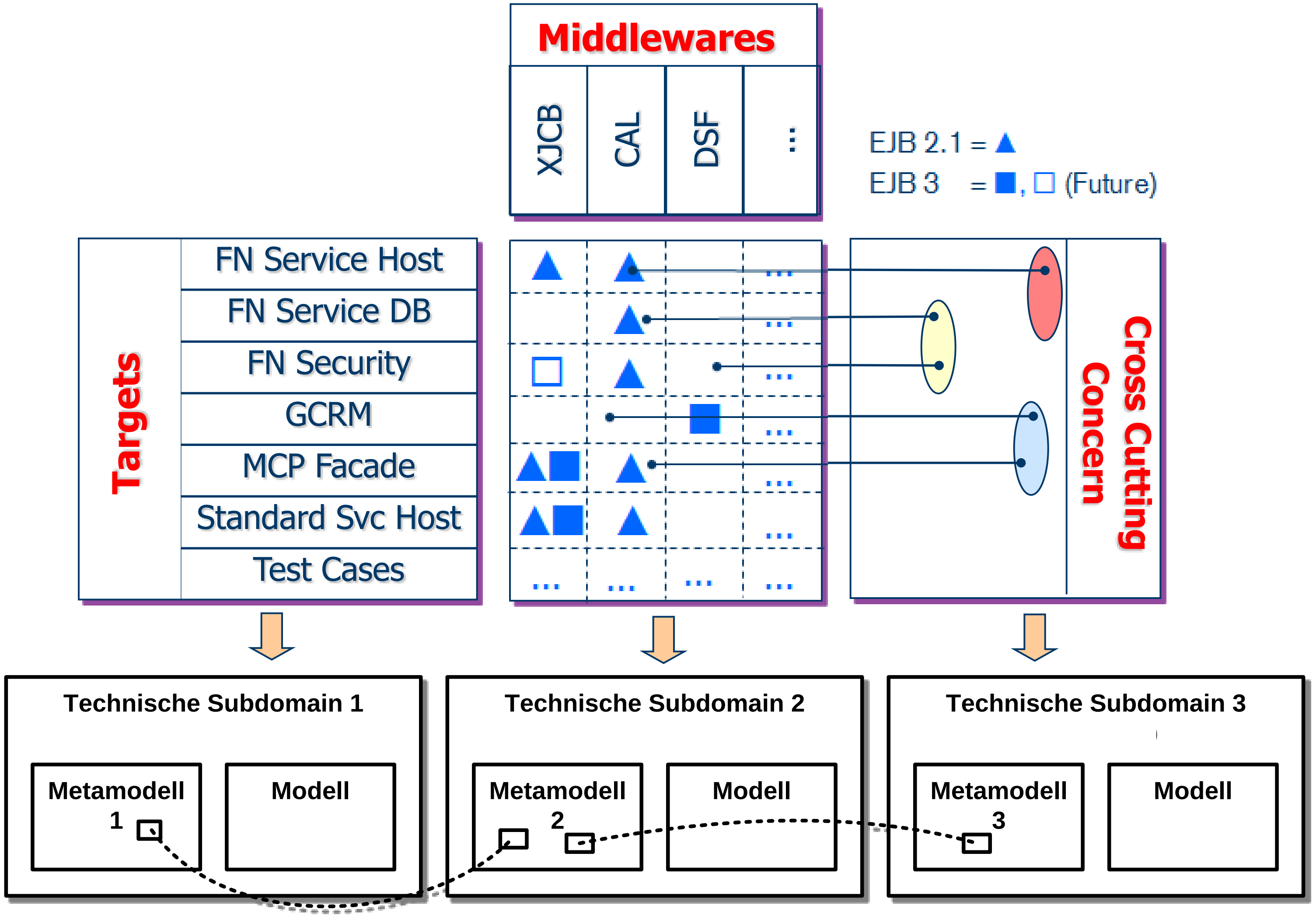
A façade is a design pattern, that provides a consistent interface for client applications and contains comprehensive logic to be reused:



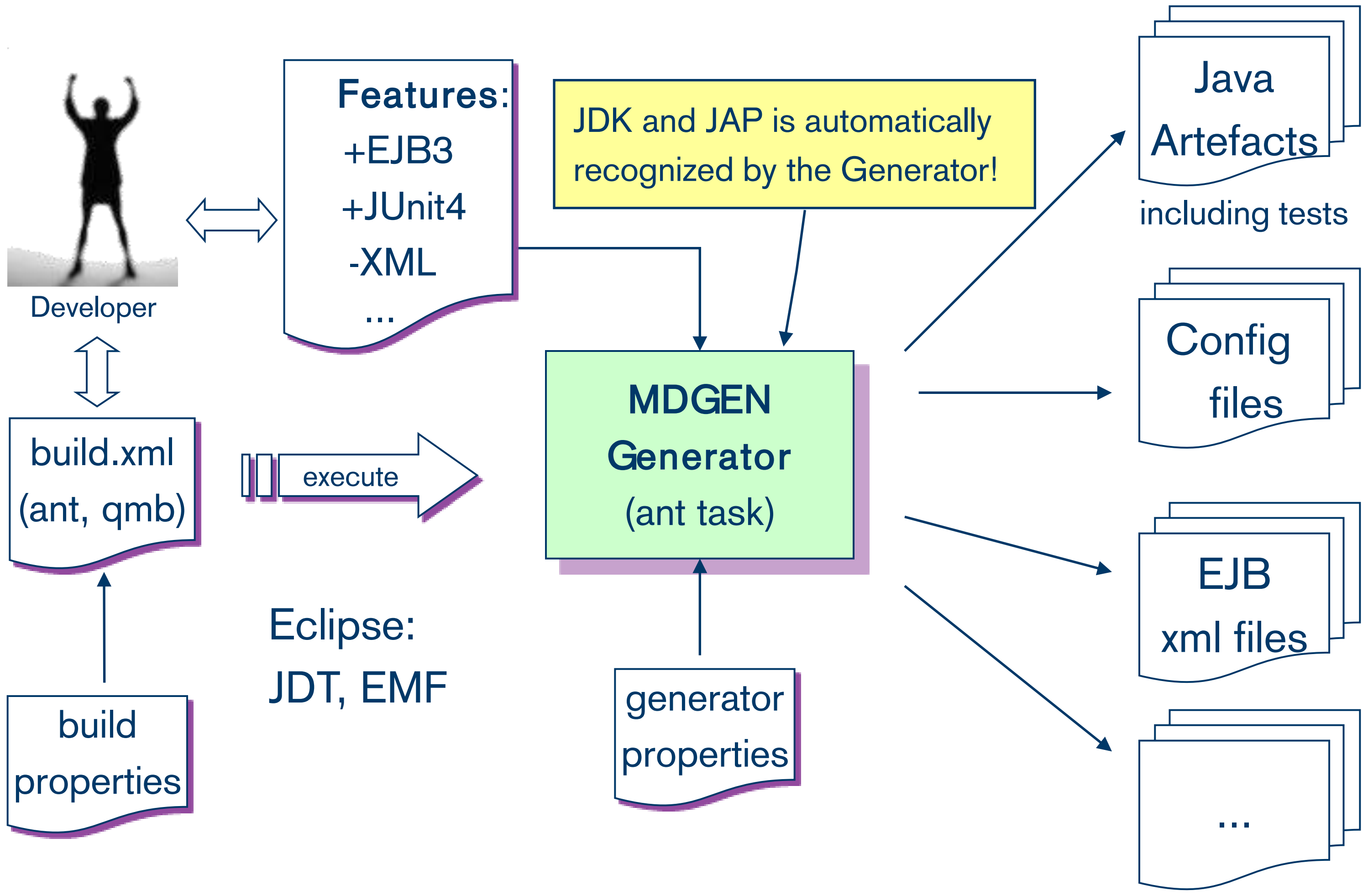
Agenda

▪ Initial Situation / Goal of the Project	(Mario Lovisi)
▪ The Multi Channel Platform (MCP)	(Mario Lovisi)
▪ IcmDsl & MDGEN Features	(Serano Colameo)
▪ Q&A	(Mario Lovisi / Serano Colameo)

1st Step: Analysis, Separation of Concerns

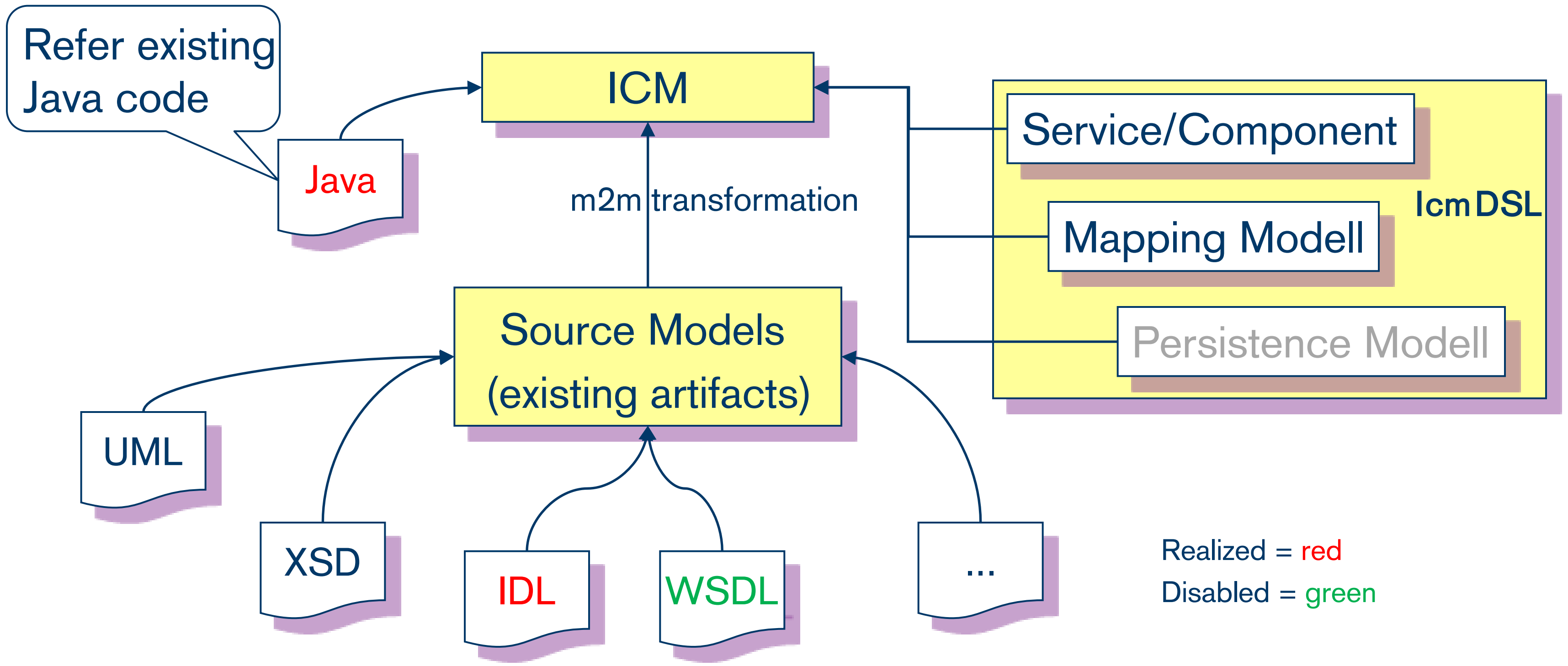


2nd Step: Define Runtime Usage of the MDGEN Tools



3rd Step: Design DSL and Generator Architecture

- Interface Component Model as a Domain Specific Language – **IcmDsl**
- Built-in and extendable **Type System with Mapping Functionalities**
- **Java is directly supported** as Model (Jvm-Model) in IcmDsl
- **IDL can be Imported** (transformed) into IcmDsl (model-2-model & model-2-text)
- Other Artifacts can be imported as well (we stopped with WSDL)



Realized = red
Disabled = green

Xtext

A Language IDE Framework

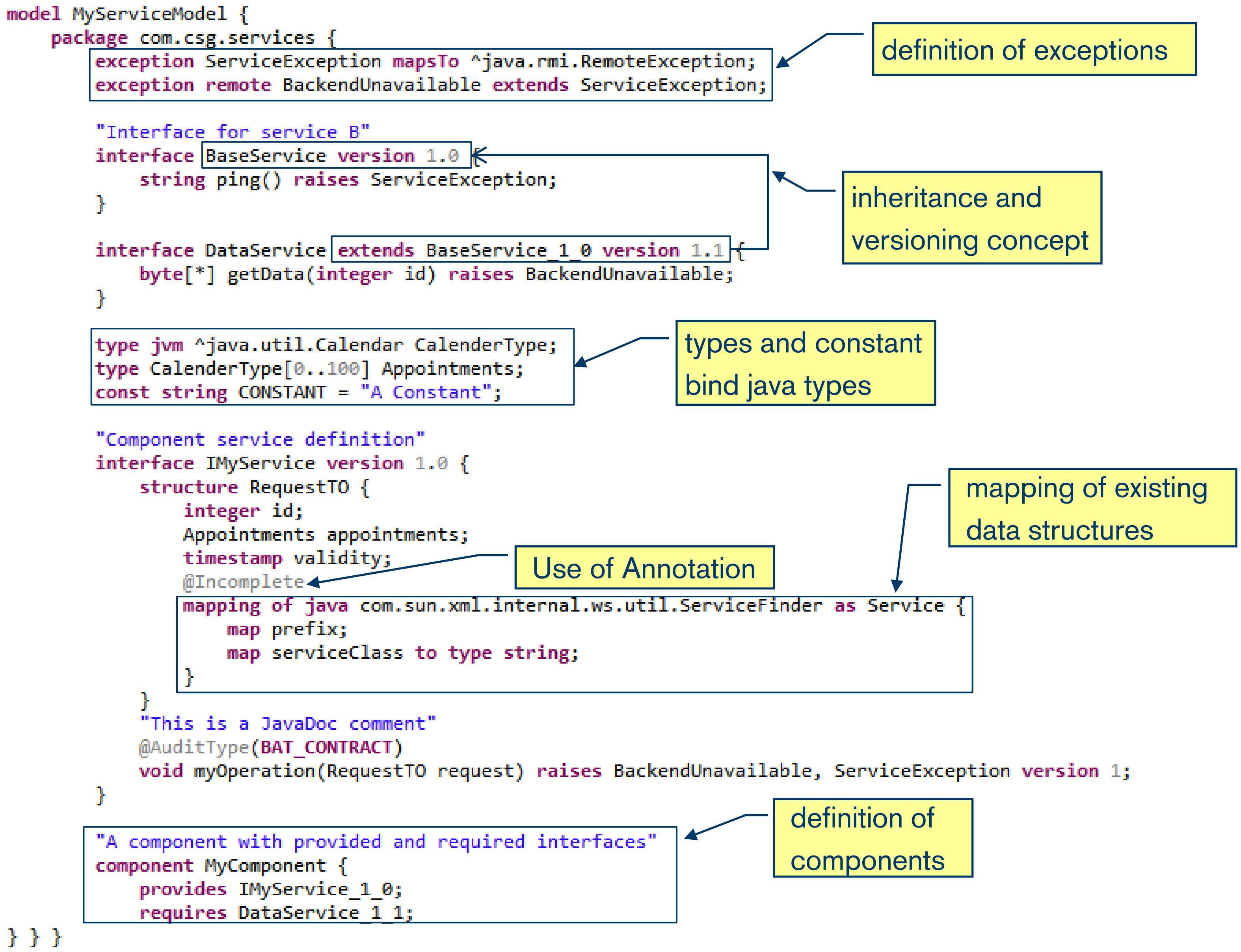
Define Your Own Language → Generate a Rich IDE → Execute Your Language

eclipse modeling PROJECT

Xtend

eclipse

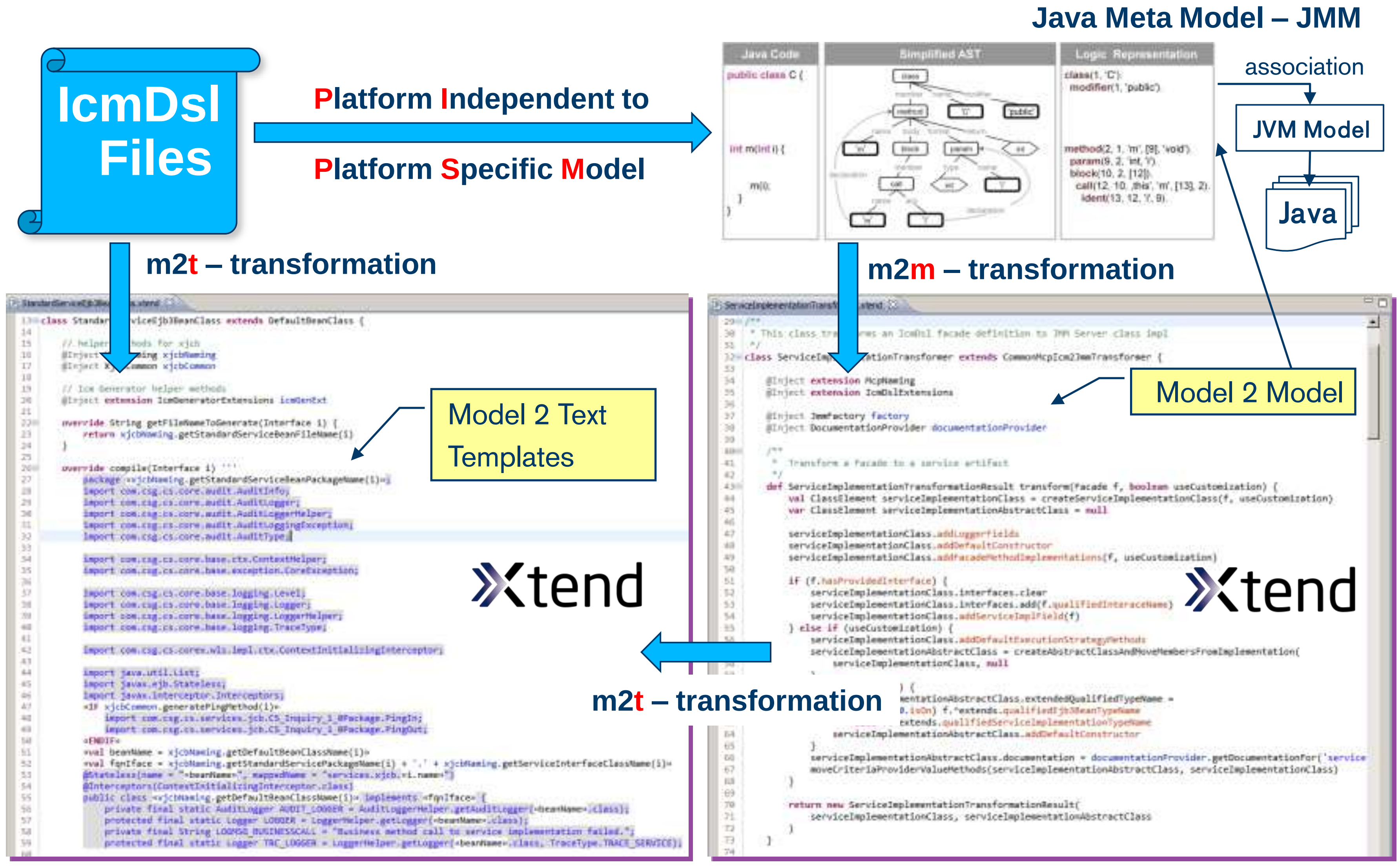
IcmDsl



```
model MyServiceModel {
    package com.csg.services {
        exception ServiceException mapsTo ^java.rmi.RemoteException;
        exception remote BackendUnavailable extends ServiceException;

        "Interface for service B"
        interface BaseService version 1.0 {
            string ping() raises ServiceException;
        }

        interface DataService extends BaseService_1_0 version 1.1 {
            byte[*] getData(integer id) raises BackendUnavailable;
        }

        type jvm ^java.util.Calendar CalenderType;
        type CalenderType[0..100] Appointments;
        const string CONSTANT = "A Constant";

        "Component service definition"
        interface IMyService version 1.0 {
            structure RequestTO {
                integer id;
                Appointments appointments;
                timestamp validity;
                @Incomplete
                mapping of java com.sun.xml.internal.ws.util.ServiceFinder as Service {
                    map prefix;
                    map serviceClass to type string;
                }
            }
            "This is a JavaDoc comment"
            @AuditType(BAT_CONTRACT)
            void myOperation(RequestTO request) raises BackendUnavailable, ServiceException version 1;
        }

        "A component with provided and required interfaces"
        component MyComponent {
            provides IMyService_1_0;
            requires DataService_1_1;
} } }
```

definition of exceptions
inheritance and versioning concept
types and constant bind java types
mapping of existing data structures
Use of Annotation
definition of components

Generic Java Generator Architecture



```
class Standar[illegible]viceEjb3BeanClass extends DefaultBeanClass {

    // helper[illegible]hods for xjcb
    @Inject [illegible]ing xjcbNaming
    @Inject X[illegible]Common xjcbCommon

    // Icm Generator helper methods
    @Inject extension IcmGeneratorExtensions icmGenExt

    override String getFileNameToGenerate(Interface i) {
        return xjcbNaming.getStandardServiceBeanFileName(i)
    }

    override compile(Interface i) '''
        package «xjcbNaming.getStandardServiceBeanPackageName(i)»;
        import com.csg.cs.core.audit.AuditInfo;
        import com.csg.cs.core.audit.AuditLogger;
        import com.csg.cs.core.audit.AuditLoggerHelper;
        import com.csg.cs.core.audit.AuditLoggingException;
        import com.csg.cs.core.audit.AuditType;

        import com.csg.cs.core.base.ctx.ContextHelper;
        import com.csg.cs.core.base.exception.CoreException;

        import com.csg.cs.core.base.logging.Level;
        import com.csg.cs.core.base.logging.Logger;
        import com.csg.cs.core.base.logging.LoggerHelper;
        import com.csg.cs.core.base.logging.TraceType;

        import com.csg.cs.corex.wls.impl.ctx.ContextInitializingInterceptor;

        import java.util.List;
        import javax.ejb.Stateless;
        import javax.interceptor.Interceptors;
        «IF xjcbCommon.generatePingMethod(i)»
            import com.csg.cs.services.jcb.CS_Inquiry_1_0Package.PingIn;
            import com.csg.cs.services.jcb.CS_Inquiry_1_0Package.PingOut;
        «ENDIF»
        «val beanName = xjcbNaming.getDefaultBeanClassName(i)»
        «val fqnIface = xjcbNaming.getStandardServicePackageName(i) + '.' + xjcbNaming.getServiceInterfaceClassName(i)»
        @Stateless(name = "«beanName»", mappedName = "services.xjcb.«i.name»")
        @Interceptors(ContextInitializingInterceptor.class)
        public class «xjcbNaming.getDefaultBeanClassName(i)» implements «fqnIface» {
            private final static AuditLogger AUDIT_LOGGER = AuditLoggerHelper.getAuditLogger(«beanName».class);
            protected final static Logger LOGGER = LoggerHelper.getLogger(«beanName».class);
            private final String LOGMSG_BUSINESSCALL = "Business method call to service implementation failed.";
            protected final static Logger TRC_LOGGER = LoggerHelper.getLogger(«beanName».class, TraceType.TRACE_SERVICE);
```

```
/**
 * This class tra[illegible]rms an IcmDsl facade definition to JMM Server class impl
 */
class ServiceImp[illegible]ationTransformer extends CommonMcpIcm2JmmTransformer {

    @Inject extension McpNaming
    @Inject extension IcmDslExtensions

    @Inject JmmFactory factory
    @Inject DocumentationProvider documentationProvider

    /**
     *  Transform a facade to a service artifact
     */
    def ServiceImplementationTransformationResult transform(Facade f, boolean useCustomization) {
        val ClassElement serviceImplementationClass = createServiceImplementationClass(f, useCustomization)
        var ClassElement serviceImplementationAbstractClass = null

        serviceImplementationClass.addLoggerFields
        serviceImplementationClass.addDefaultConstructor
        serviceImplementationClass.addFacadeMethodImplementations(f, useCustomization)

        if (f.hasProvidedInterface) {
            serviceImplementationClass.interfaces.clear
            serviceImplementationClass.interfaces.add(f.qualifiedInteraceName)
            serviceImplementationClass.addServiceImplField(f)
        } else if (useCustomization) {
            serviceImplementationClass.addDefaultExecutionStrategyMethods
            serviceImplementationAbstractClass = createAbstractClassAndMoveMembersFromImplementation(
                serviceImplementationClass, null
            )
[illegible]) {
[illegible]entationAbstractClass.extendedQualifiedTypeName =
[illegible]0.isOn) f.'extends.qualifiedEjb3BeanTypeName
[illegible]extends.qualifiedServiceImplementationTypeName
                serviceImplementationAbstractClass.addDefaultConstructor
            }
            serviceImplementationAbstractClass.documentation = documentationProvider.getDocumentationFor('service
            moveCriteriaProviderValueMethods(serviceImplementationAbstractClass, serviceImplementationClass)
        }

        return new ServiceImplementationTransformationResult(
            serviceImplementationClass, serviceImplementationAbstractClass
        )
    }
```

Generated Artifacts

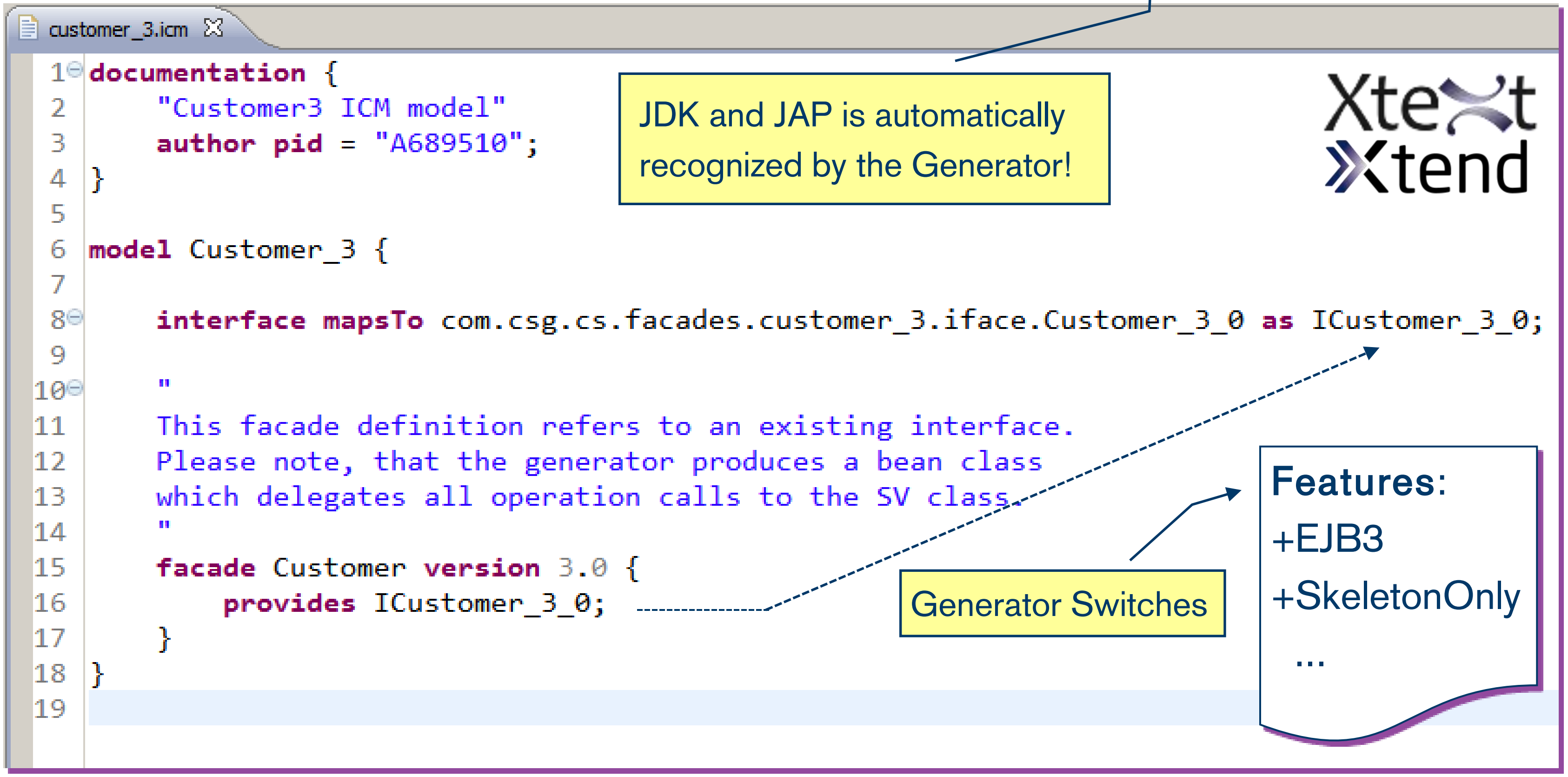
- Service Layer: Interface, Implementation, Delegator, Helper Classes
- Transport Layer: Transfer Objects, Transfer Object Handler, Command Classes
- Business Layer: Business Objects, Data Mapper Classes
- Data Layer: XJCB Services (Java Corba Bridge Service Artifacts)
- JUnit Tests for each Service Operation
- Deployment Descriptors
- Configuration Files (XML)
- ...

Use of Generation Gap Pattern
to allow manual enhancement

All needed Service Artifacts are automatically generated
Mapping of Business and Transfer Objects can be also generated
Only complex Business Logic must be manually implemented

Migration of Existing MCP Façade Services

Change Technology Stack: EJB2.1 => EJB3, JAP6 => JAP7



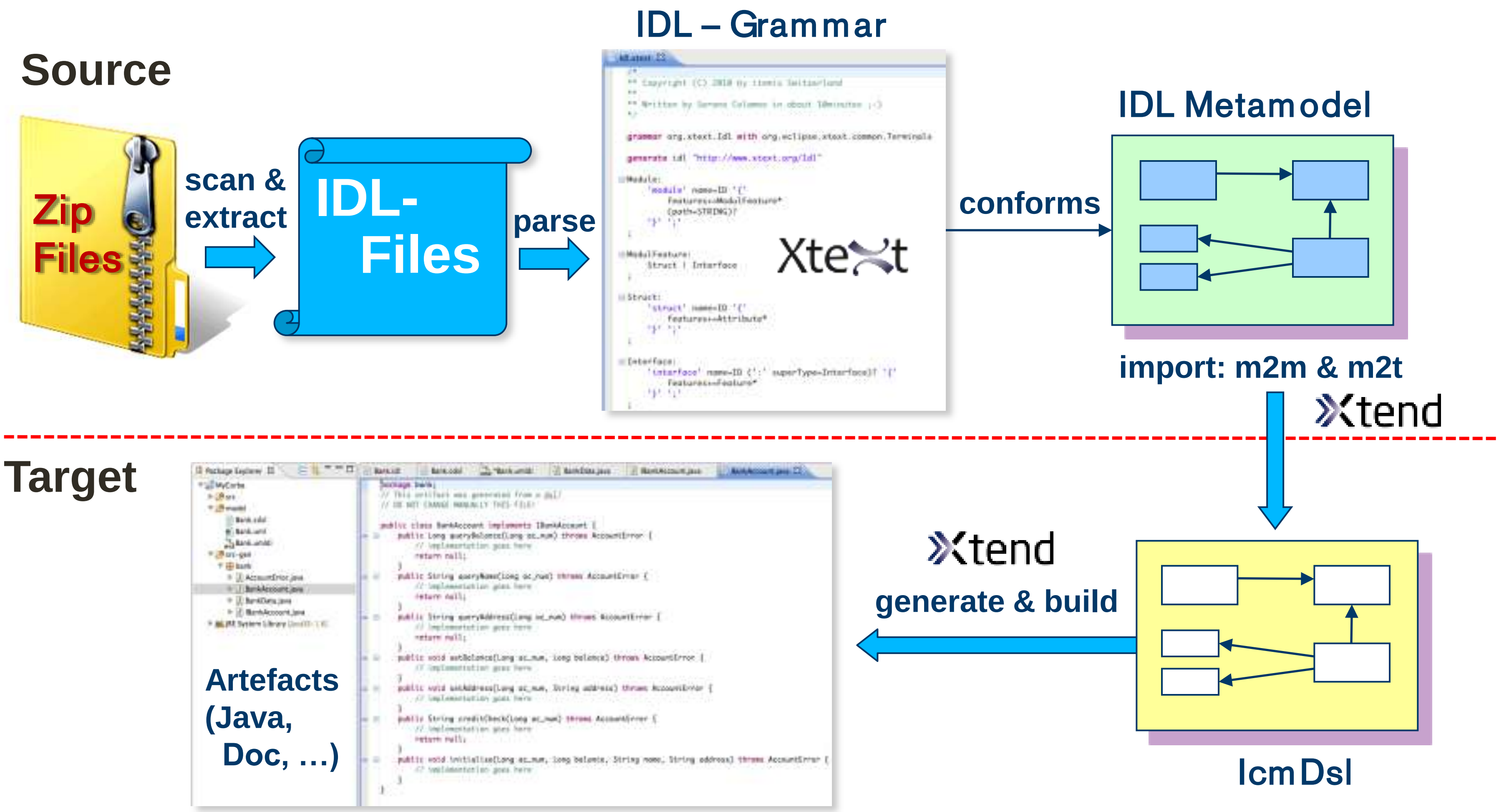
```
documentation {
    "Customer3 ICM model"
    author pid = "A689510";
}

model Customer_3 {

    interface mapsTo com.csg.cs.facades.customer_3.iface.Customer_3_0 as ICustomer_3_0;

    "
    This facade definition refers to an existing interface.
    Please note, that the generator produces a bean class
    which delegates all operation calls to the SV class.
    "
    facade Customer version 3.0 {
        provides ICustomer_3_0;
    }
}
```

JDK and JAP is automatically recognized by the Generator!

Generator Switches

Features:
+EJB3
+SkeletonOnly
...

Migration of Legacy Services (Java Corba Bridge – JCB)



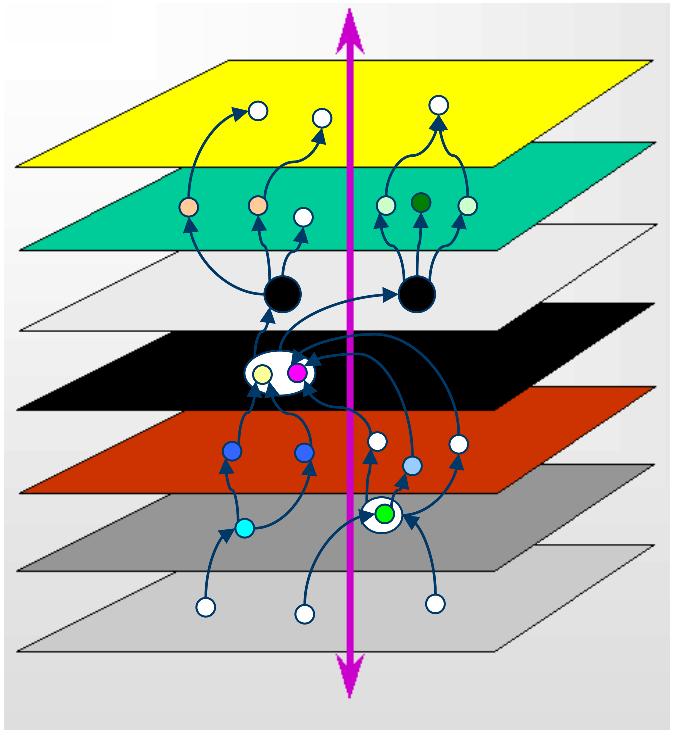
```
/*
** Copyright (C) 2010 by itemis Switzerland
**
** Written by Serano Colameo in about 10minutes ;-)
*/

grammar org.xtext.Idl with org.eclipse.xtext.common.Terminals

generate idl "http://www.xtext.org/Idl"

Module:
    'module' name=ID '{'
        features+=ModulFeature*
        (path=STRING)?
    '}' ';'
;

ModulFeature:
    Struct | Interface
;

Struct:
    'struct' name=ID '{'
        features+=Attribute*
    '}' ';'
;

Interface:
    'interface' name=ID (':' superType=Interface)? '{'
        features+=Feature*
    '}' ';'
;
```

```
package bank;
// This artifact was generated from a dsl!
// DO NOT CHANGE MANUALLY THIS FILE!

public class BankAccount implements IBankAccount {
    public Long queryBalance(Long ac_num) throws AccountError {
        // implementation goes here
        return null;
    }
    public String queryName(Long ac_num) throws AccountError {
        // implementation goes here
        return null;
    }
    public String queryAddress(Long ac_num) throws AccountError {
        // implementation goes here
        return null;
    }
    public void setBalance(Long ac_num, Long balance) throws AccountError {
        // implementation goes here
    }
    public void setAddress(Long ac_num, String address) throws AccountError {
        // implementation goes here
    }
    public String creditCheck(Long ac_num) throws AccountError {
        // implementation goes here
        return null;
    }
    public void initialise(Long ac_num, Long balance, String name, String address) throws AccountError {
        // implementation goes here
    }
}
```

We migrated over 300 existing JCB services by just importing and regenerating all existing artifacts in one step!

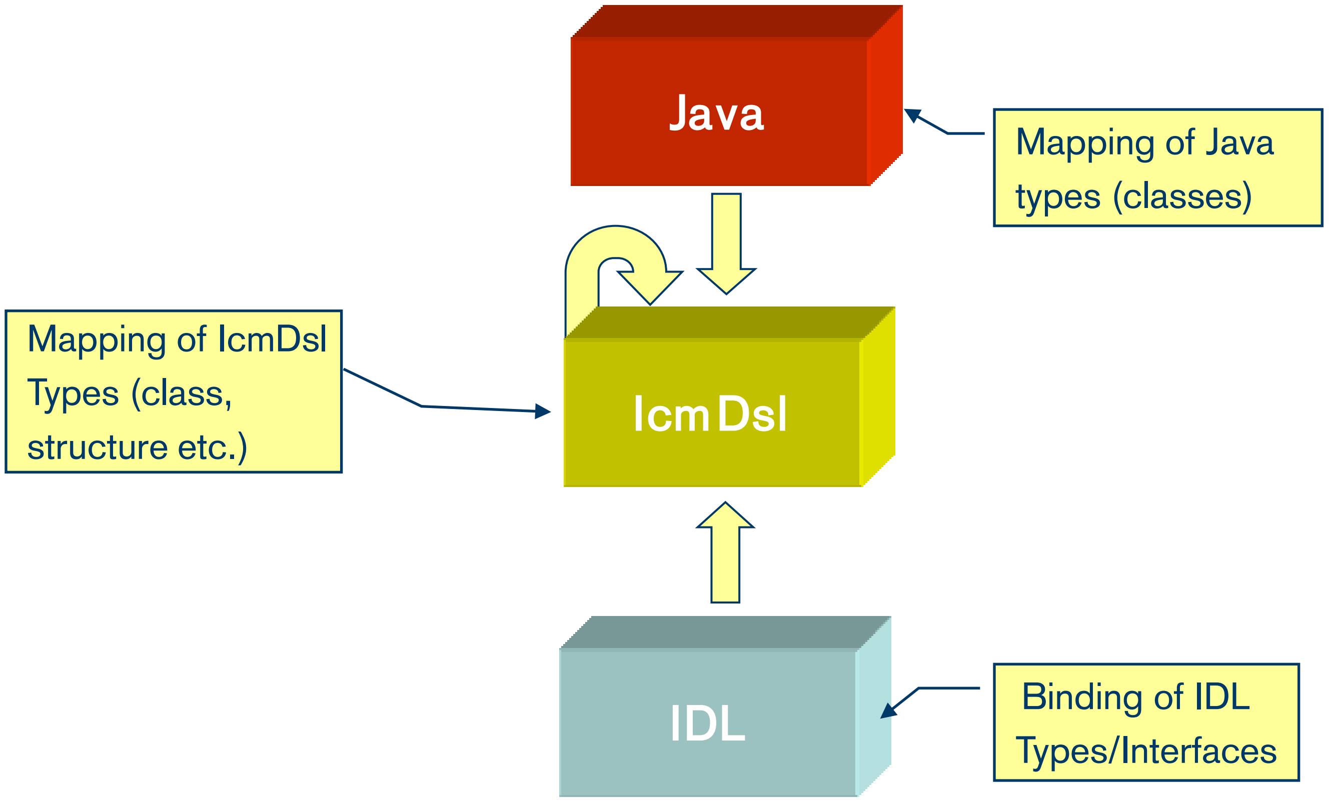
Mapping of Tier Layer Data Structures

- Enterprise Architectures consists of many Layers...



Mapping of Data Structures

Mapping and Binding of existing Models and Artifacts


Java
Mapping of Java types (classes)
Mapping of IcmDsl Types (class, structure etc.)
IcmDsl
IDL
Binding of IDL Types/Interfaces

Mapping (i) of Model and/or Java Types in Combination

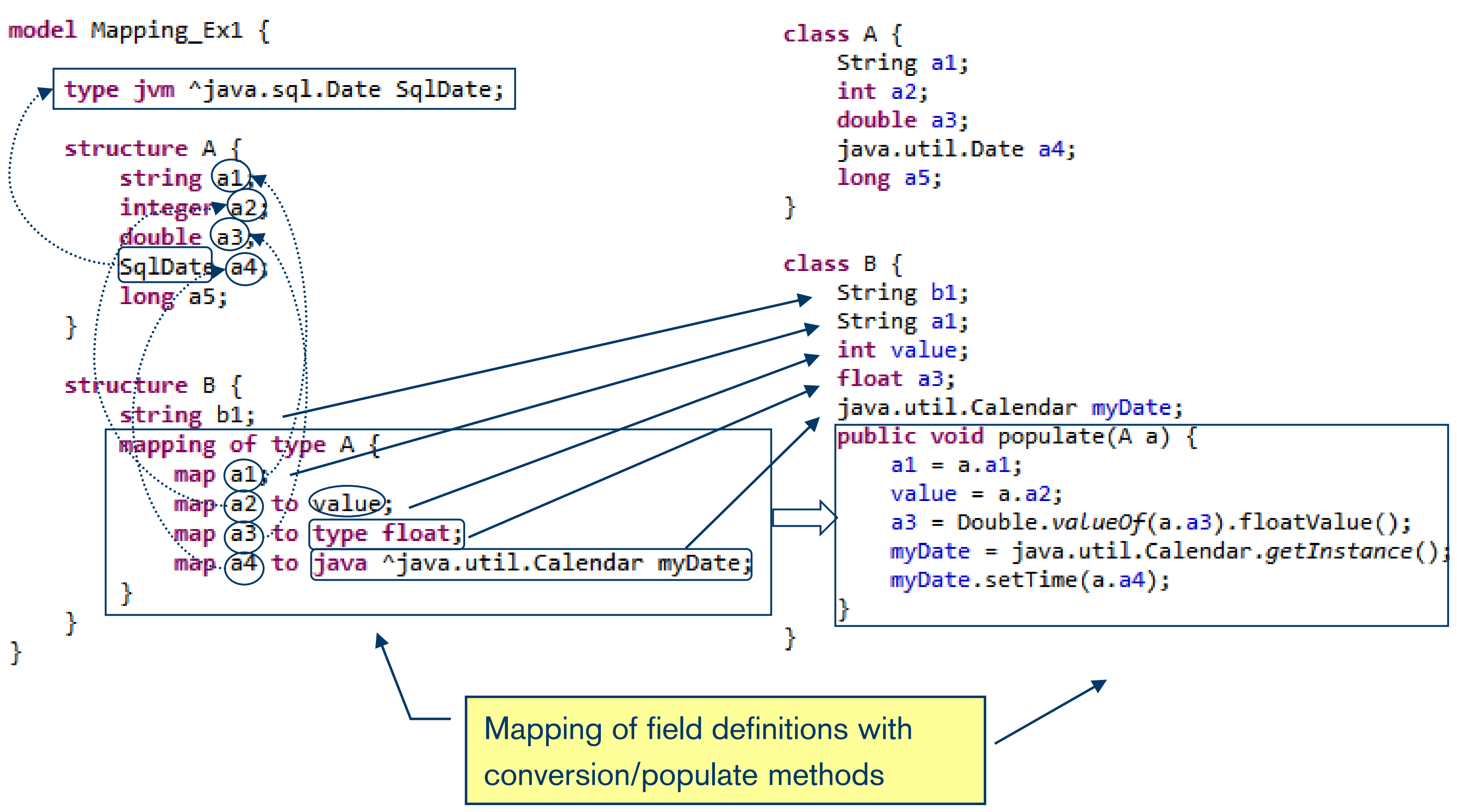
IcmDsl:

```
structure MappedStructure {
    mapping of type A {
        map a1;
        // ...
    }
    mapping of java A {
        map a2;
        // ...
    }
    mapping from type A to type B {
        map a1 to b1;
        // ...
    }
    mapping from type B to java A {
        map b1 to a1;
        // ...
    }
    mapping from java A to type B {
        map a1 to b1;
        // ...
    }
    mapping from java A to java B {
        map a1 to b1;
        // ...
    }
}
```

Generated Java Code:

```
class MappedStructure {
    String a1;
    int b1;

    public void populate(A a) {
        a1 = a.a1;
    }

    public void populate(B b) {
        b1 = b.b1;
    }

    public B toB(A a) {
        B result = new B();
        result.b1 = Integer.valueOf(a.a1).intValue();
        return result;
    }

    public A toA(B b) {
        A result = new A();
        result.a1 = String.valueOf(b.b1);
        return result;
    }

    // ...
}
```

Mapping (ii) of Model and/or Java Types



IcmDsl:

```
model Mapping_Ex1 {

  type jvm ^java.sql.Date SqlDate;

  structure A {
      string a1;
      integer a2;
      double a3;
      SqlDate a4;
      long a5;
  }

  structure B {
      string b1;
      mapping of type A {
          map a1;
          map a2 to value;
          map a3 to type float;
          map a4 to java ^java.util.Calendar myDate;
      }
  }
}
```

Generated Java Code:

```
class A {
    String a1;
    int a2;
    double a3;
    java.util.Date a4;
    long a5;
}

class B {
    String b1;
    String a1;
    int value;
    float a3;
    java.util.Calendar myDate;
    public void populate(A a) {
        a1 = a.a1;
        value = a.a2;
        a3 = Double.valueOf(a.a3).floatValue();
        myDate = java.util.Calendar.getInstance();
        myDate.setTime(a.a4);
    }
}
```

Mapping of field definitions with conversion/populate methods

Mapping (iii) Type Selection with the "."-Dot-Notation

IcmDsl:

```
model Mapping_Ex2 {

    class Z {
        string z1;
    }

    class A {
        string a1;
        Z az;
    }

    class B extends A {
        Z bz;
        string b1;
    }

    class C {
        string c1;

    mapping from type A[*] to type B[*] {
        map az.z1 to bz.z1;
        map a1 to b1;
    }
    }
}
```

Generated Java Code:

```
import java.util.ArrayList;
import java.util.List;

class C {
    String c1;

    public String getC1() {
        return c1;
    }

    public void setC1(String c1) {
        this.c1 = c1;
    }

    public List<B> populate(List<A> input) {
        List<B> out = new ArrayList<B>();
        for (A a : input) {
            B b = new B();
            b.setB1(a.getA1());
            b.setBz(a.getAz());
        }
        return out;
    }
}
```

Mapping of Types on Field definitions using the dot (".") notation. Cardinality is of course also supported…

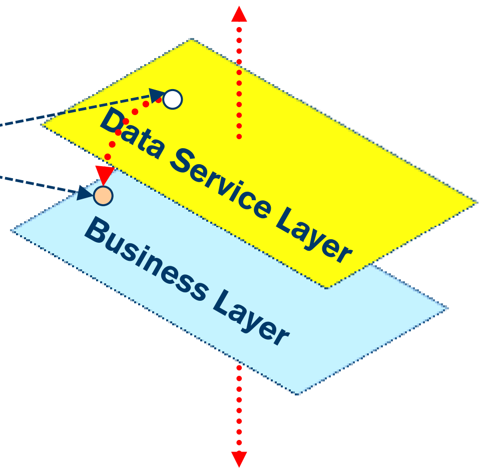
Mapping of Service Structures to Business Objects

```
businessObject CifDetails {
    @MappedBy(GetCifsShortDataMapper_1_0)
    mapping of java CifShortStruc_T {
        map address;
        map birthDate to type date;
        map buId;
        map cifNo;
        map cifUwi;
        map creditOfficer;
        map deceasedDate to type date;
        map incorpDate to type date;
        map nationalityCd to nationalityCode;
        map domicileCd to domicileCode;
        map customerLanguageCd to customerLanguageCode;
        map customerStateCd to customerStateCode;
        map custTypeCd to customerTypeCode;
        map custSegCd to customerSegmentCode;
        map maritalStateCd to maritalStateCode;
        map occupationCd to occupationCode;
        map economicSector;
        map relationManager;
        map customerOffice;
        map phoneNoP to phoneNumberPrivate;
        map phoneNoB to phoneNumberBusiness;
        map shortAddress to shortAddress;
    }
    mapping of java CifShort_T {
        @Incomplete map busEx to java Remark[*] remark;
    }
}
```

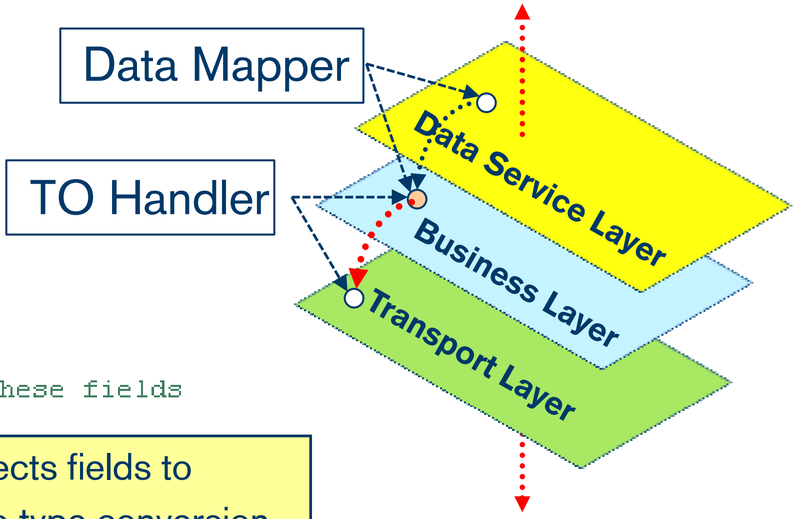
Business Object

Service Data Structure

Data Mapper



Mapping of Business Objects to Transfer Objects



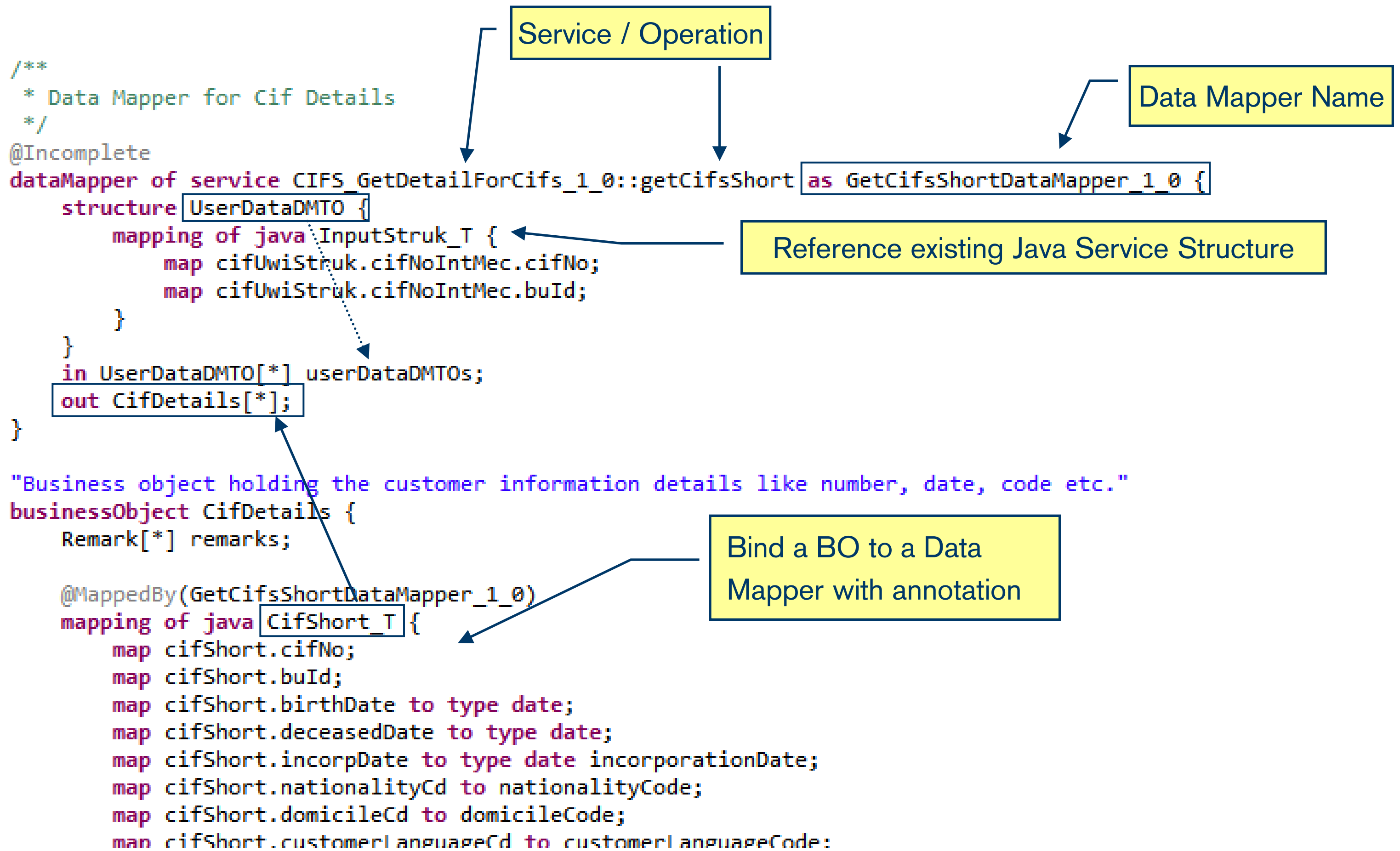
```
"User data Input Transfer Object"
transferObject UserDataINTO {
    "This mapping is based on the CifDetals BO"
    mapping of type CifDetails {
        select cifNo, buId, address, cifUwi; // only these fields
    }
}
```

Mapping of Business Objects fields to Transfer Objects allows no type conversion

```
"User data Output Transfer Object"
transferObject UserDataTO {
    mapping of type CifDetails; // map all fields
}
```

No «select» keyword means map all fields

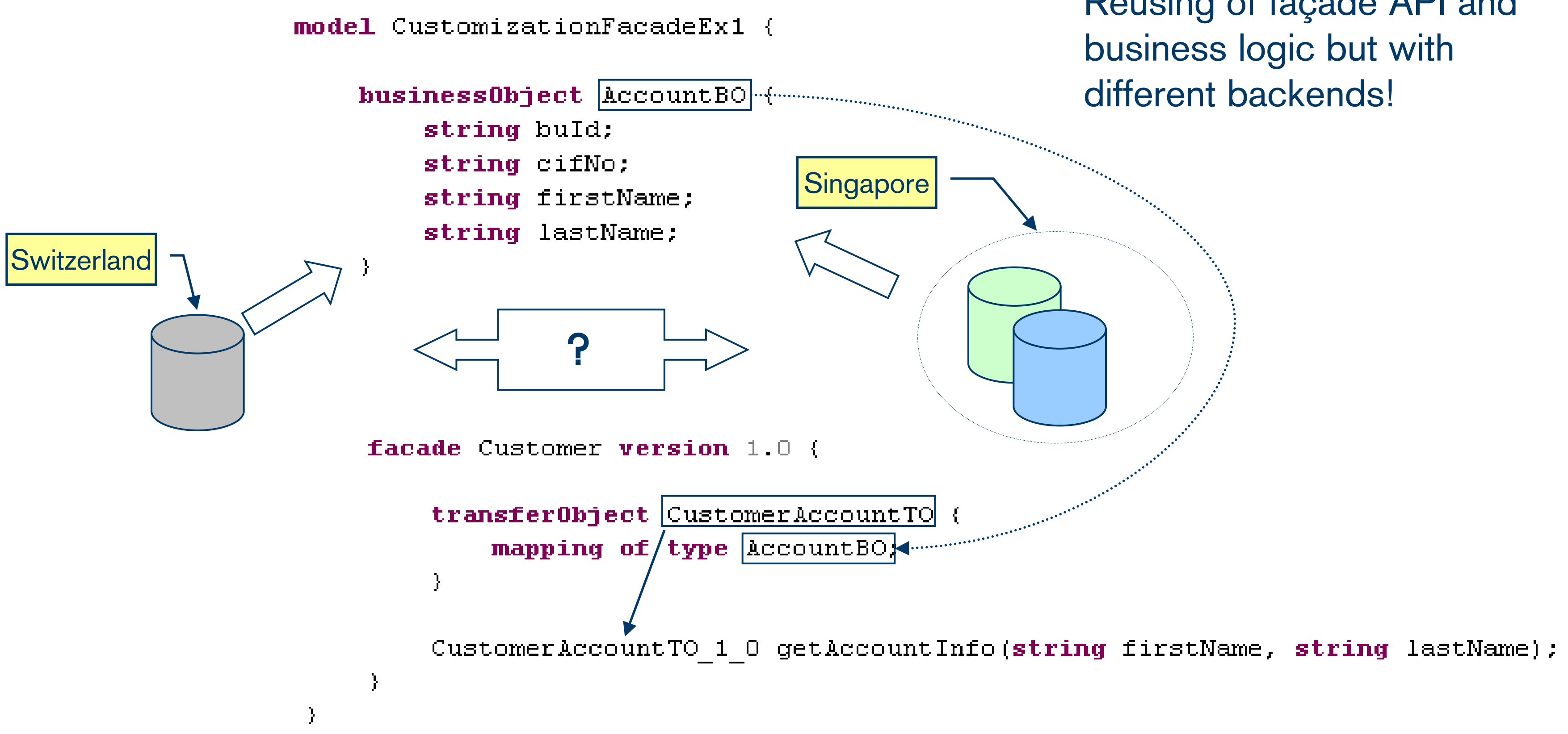
Mappings are managed by Data Mappers


Service / Operation
Data Mapper Name
Reference existing Java Service Structure
Bind a BO to a Data Mapper with annotation

```
/**
 * Data Mapper for Cif Details
 */
@Incomplete
dataMapper of service CIFS_GetDetailForCifs_1_0::getCifsShort as GetCifsShortDataMapper_1_0 {
    structure UserDataDMTO {
        mapping of java InputStruk_T {
            map cifUwiStruk.cifNoIntMec.cifNo;
            map cifUwiStruk.cifNoIntMec.buId;
        }
    }
    in UserDataDMTO[*] userDataDMTOs;
    out CifDetails[*];
}

"Business object holding the customer information details like number, date, code etc."
businessObject CifDetails {
    Remark[*] remarks;

    @MappedBy(GetCifsShortDataMapper_1_0)
    mapping of java CifShort_T {
        map cifShort.cifNo;
        map cifShort.buId;
        map cifShort.birthDate to type date;
        map cifShort.deceasedDate to type date;
        map cifShort.incorpDate to type date incorporationDate;
        map cifShort.nationalityCd to nationalityCode;
        map cifShort.domicileCd to domicileCode;
        map cifShort.customerLanguageCd to customerLanguageCode;
```

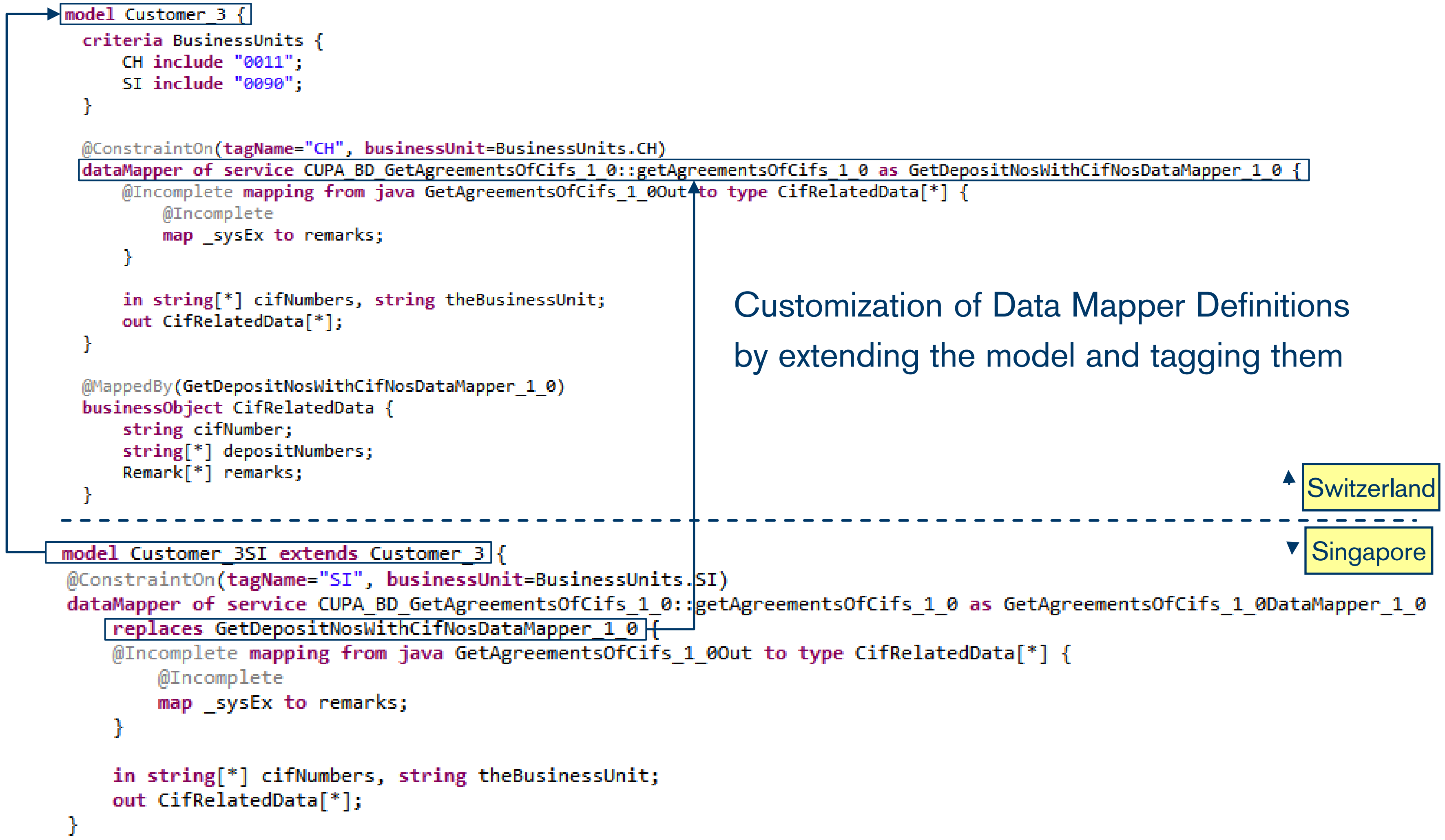
MCP – Location Specific Customization (I)

Reusing of façade API and business logic but with different backends!



```
model CustomizationFacadeEx1 {

    businessObject AccountBO {
        string buId;
        string cifNo;
        string firstName;
        string lastName;
    }

    facade Customer version 1.0 {

        transferObject CustomerAccountTO {
            mapping of type AccountBO;
        }

        CustomerAccountTO_1_0 getAccountInfo(string firstName, string lastName);
    }
}
```

MCP – Location Specific Customization (II)



```
model Customer_3 {
    criteria BusinessUnits {
        CH include "0011";
        SI include "0090";
    }

    @ConstraintOn(tagName="CH", businessUnit=BusinessUnits.CH)
    dataMapper of service CUPA_BD_GetAgreementsOfCifs_1_0::getAgreementsOfCifs_1_0 as GetDepositNosWithCifNosDataMapper_1_0 {
        @Incomplete mapping from java GetAgreementsOfCifs_1_0Out to type CifRelatedData[*] {
            @Incomplete
            map _sysEx to remarks;
        }

        in string[*] cifNumbers, string theBusinessUnit;
        out CifRelatedData[*];
    }

    @MappedBy(GetDepositNosWithCifNosDataMapper_1_0)
    businessObject CifRelatedData {
        string cifNumber;
        string[*] depositNumbers;
        Remark[*] remarks;
    }
```

```
model Customer_3SI extends Customer_3 {
  @ConstraintOn(tagName="SI", businessUnit=BusinessUnits.SI)
  dataMapper of service CUPA_BD_GetAgreementsOfCifs_1_0::getAgreementsOfCifs_1_0 as GetAgreementsOfCifs_1_0DataMapper_1_0
      replaces GetDepositNosWithCifNosDataMapper_1_0 {
      @Incomplete mapping from java GetAgreementsOfCifs_1_0Out to type CifRelatedData[*] {
          @Incomplete
          map _sysEx to remarks;
      }

      in string[*] cifNumbers, string theBusinessUnit;
      out CifRelatedData[*];
  }
```

Customization of Data Mapper Definitions by extending the model and tagging them

Questions