Meta-Modeling

Plan

- **Introduction**
- **Modeling & Meta-Modeling**
  - Systems, Models, Meta-models, Meta-Meta-Models
- **MOF(Meta-Object-Facility)**
  - OMG Standard for meta-modeling
- **Building software engineering process using meta-models**
  - MDE (Model Driven engineering)
  - Model Transformations
  - **…**
- **3 big Examples**
  - Model Driven Production of graphic modeling tools
    - Model Driven Process to use electronic circuits
    - Model Driven Production of web applications

مقدمة

- أسبوع مطوري البرمجيات
  - السبت = C & Sockets
  - الأحد = Java & RPC
  - الاثنين = CORBA & C++ & Java & ....
  - الثلاثاء = JSP & Servlets
  - الأربعاء = Java & EJB
  - الخميس = CORBA & CCM
  - الجمعة = Wml or HTML or VoiceXML or ...
- >>> لا رأسملة لتعريف النظم

تطور مفاهيم التقنيات

- Before 1980 : Procedures
  - functions
- 1980-1995 : Objects
  - Object, Class, Method, Attribute
- After 1995 : Components
  - Component, Container, ..., Framework
- .....

الحرب بين الـ midlewares

- Sun Microsystems
  - Java = Universal Runtime
- OMG
  - CORBA = Universal midleware
- MicroSoft & al
  - Web Services = Universal Interoperability
- >>> النتيجة تعادل و الكرة في المنتصف

تطور التقنيات

- لا يوجد تقنية مثالية (لا الآن و أبداً)
- التقنيات الجديدة = مشاكل جديدة
- تطور التقنيات لا يتوقف و لن يتوقف
- إن التقنية الأفضل هي ما سيأتي

المشكلة و الحل

- المشكلة = منطق العمل مخلوط بالكود التقني
  - هذا صحيح من أجل كل التقنيات
- تطوير التطبيقات ليس سهلاً
  - الحفاظ على منطق العمل
  - إلقاء الكود التقني
- الحل = إذاً لا بد من تجريد التقنية
  - Separation of concerns
    - Buisness concerns
    - Technical concerns
  - نمذجة النظم (غير مثالية و لكن تبدو تحدي جيد)
    - توصيف مجرد لمنطق العمل
    - إعادة الاستخدام في عدة سياقات (تقنيات)

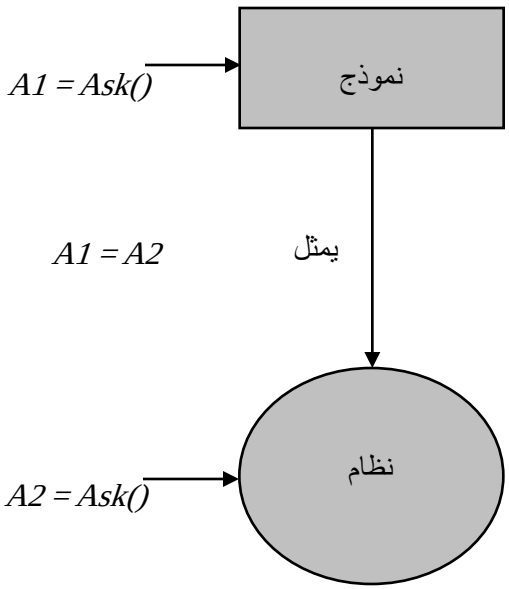
من النماذج إلى الأنظمة

- رأسملة تعريف النظم
- استخدام معايير قياسية للنمذجة
- قواعد الإسقاط :
  - للتعريف من أجل كل تقنية
  - للتطور مع تطور التقنيات
- تحسين تطور و تكامل الأنظمة

Meta-Modeling

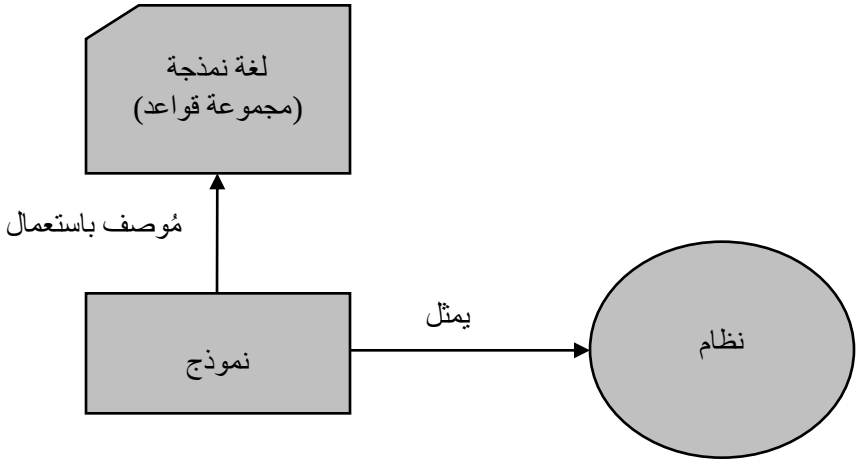
النماذج ١

- معنى كلمة "نموذج" يختلف بحسب مجالات نشاط الحياة الإنسانية [الفن، صناعة الملابس، .. إلخ]. بالمقابل فإن كلمة "نموذج" لها معاني مشتركة بين كل هذه المجالات:
  - النموذج هو تجريد لشيء ما في العالم الحقيقي.
  - النموذج يختلف عن الشيء الذي يمثله. على سبيل المثال، في تفاصيله، في حجمه، .. إلخ.
  - النموذج يمكن أن يستعمل لإنتاج شيء ما في العالم الحقيقي.
- نمذجة نظام [جزء من العالم الحقيقي] = وصف للخصائص التي تهمنا من النظام بهدف تسهيل معالجة الجزء الذي يهمنا من النظام.
- النموذج يجب أن يجيب على بعض الأسئلة [التي تهمنا] بدلاً من النظام. أجوبة النموذج يجب أن تكون مطابقة لأجوبة النظام.



النماذج ٢

- تعريف النماذج يجب أن يتبع عدداً من القواعد المعرفة بشكل جيد.
  - ضروري لتحاشي الأخطاء في تفسير النماذج خصوصاً إذا ما كانت هذه النماذج تستخدم لتوليد الأنظمة آلياً
  - إذاً النماذج يجب أن تكون معرفة باستخدام لغة نمذجة.



Meta-Modeling

- Definition
- MOF
- MOF Architecture
- From MOF to tools
- Tools for MOF

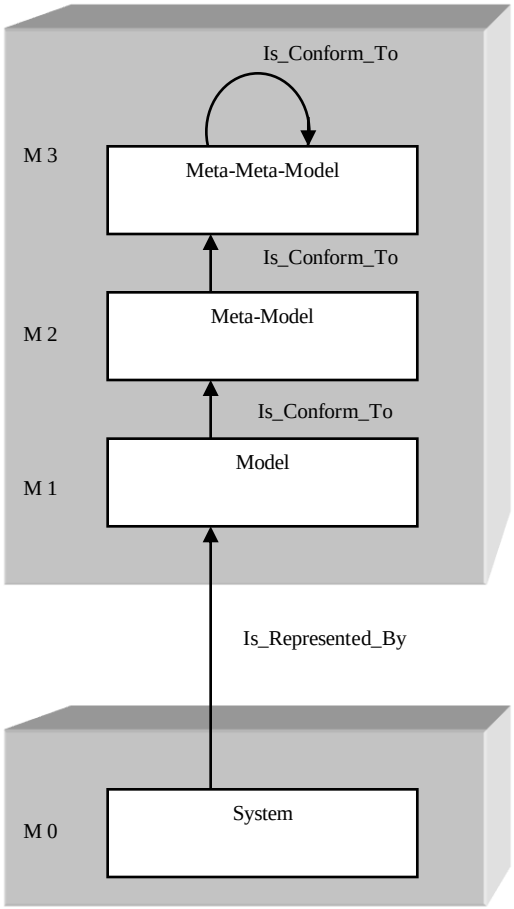
Meta-Modeling Definition

- نوع النماذج يحدد كيفية استخدام النماذج في سياق معين
  - نحتاج لمصطلحات كـ جدول Table و عمود Column و مفتاح Key لنمذجة أنظمة قواعد البيانات
  - نحتاج لمفاهيم كـ Component و Port و Connectorلنمذجة التطبيقات الموجهة نحو المركبات البرمجية
- النمذجة = تصميم الأنظمة باستعمال مفاهيم و قواعد معرفة بشكل مسبق
- تقنيات الـ Meta-Modeling تسمح بتعريف المفاهيم اللازمة لنمذجة الأنظمة
- Meta-Modeling هي تقنية تسمح بتعريف "Meta-Models".
  - Meta-Model يُعرف لغرض دقيق حيث يقدم مفاهيم متكيفة و أدوات متخصصة بمجال معين من التطبيقات.
  - Meta-Model = نوع من النماذج.
  - إن كون مختلف الـ "Meta-Models" يُعرفون عند مستوى أعلى من التجريد يسهل اندماجهم فيما بعد

MOF (Meta-Object Facility)

- الـ OMG (Object Management Group)تبنت الـ MOF كوسيلة قياسية من أجل الـ "Meta-Modeling"
  - إن الـ "MOF" مستقل عن تقنيات التنفيذ.
  - إن الـ "MOF" يسمح بالدرجة الأولى التعبير عن معنى الـ "Meta-Model" بينما التركيب اللغوي يأتي بالدرجة الثانية
  - الـ "MOF" يقترح منحى "Top-Down" لتصميم الأنظمة ما يسمح بتوصيف تدريجي للمشكلة و من ثم كيفية حل المشكلة.
  - الـ "MOF" يُعرف روابط بين مستوياته الثلاثة الأولى بطريقة تسمح بأتمتة تطوير الأدوات.
    - نستطيع معالجة المستويات المنخفضة بالاعتماد على المستويات الأعلى.

MOF Architecture

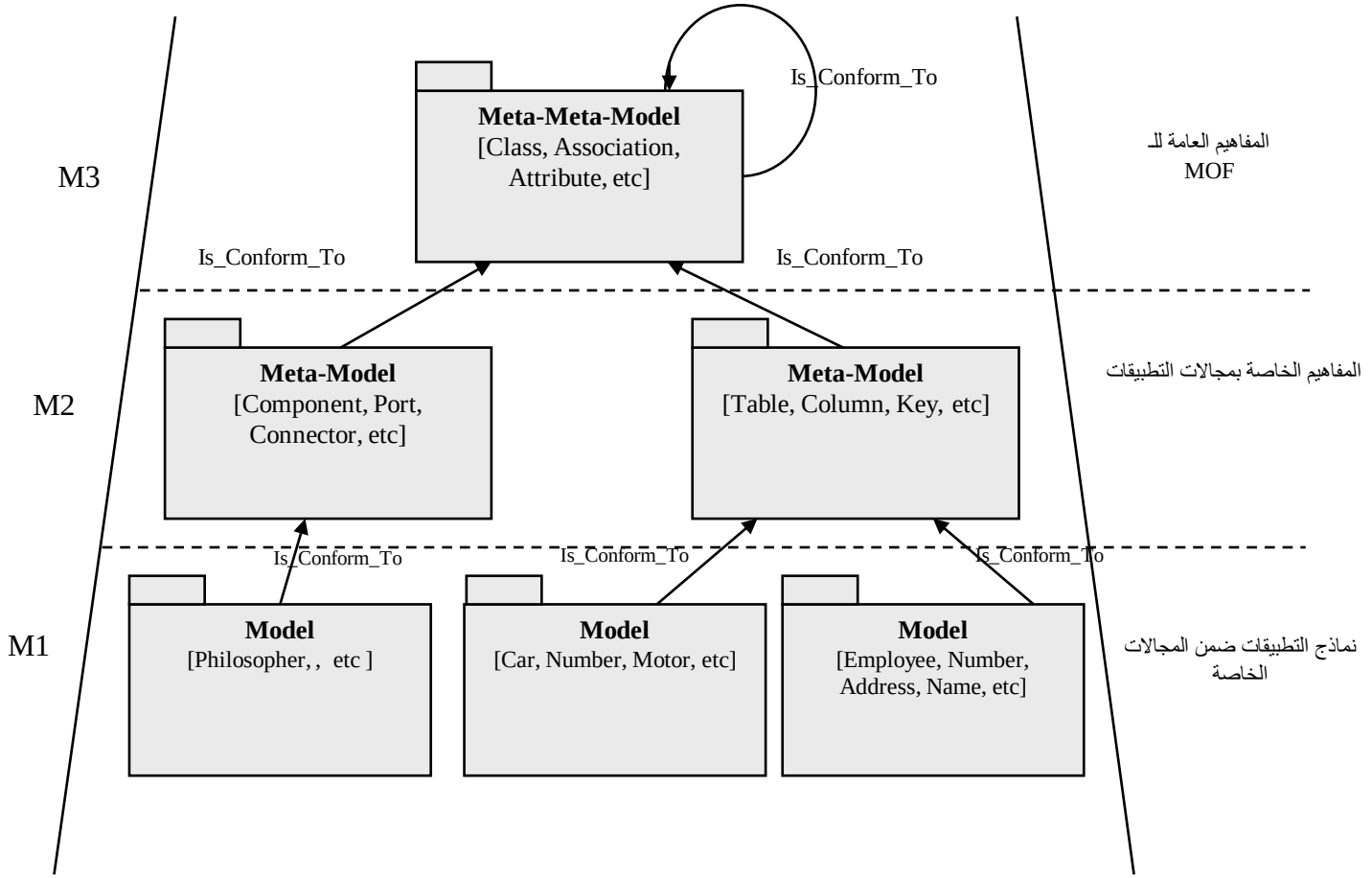


- الـ "MOF" معرف كمعمارية من أربعة مستويات M3, M2, M1, M0
  - كل مستوى يحتوي المعلومات التي تصف المستوى الأدنى
- معمارية الـ "MOF" منظمة بالأحرى من ٣+١ مستوى.
  - المستوى M0 هو النظام الحقيقي و هو ممثل بالـ Model المعرف عند المستوى M1.
  - نموذج المستوى M1 يكون موافقاً للـ "Meta-Model" المعرف لدى المستوى M2.
  - الـ Meta-model يوافق الـ "Meta-Meta-Model" عند المستوى M3.
  - الـ Meta-Meta-Model يوافق نفسه

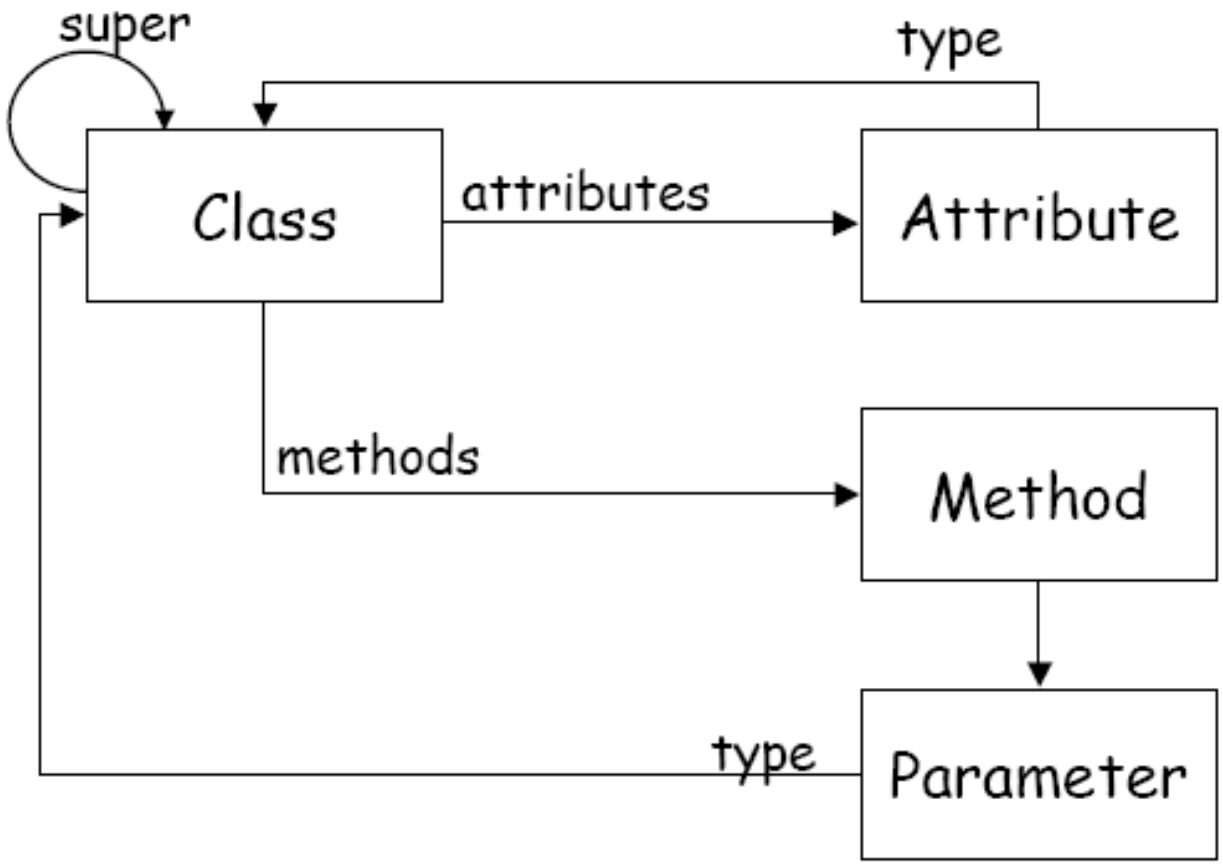
MOF Concepts

- Class
- Association
- Reference
- Package
- ....

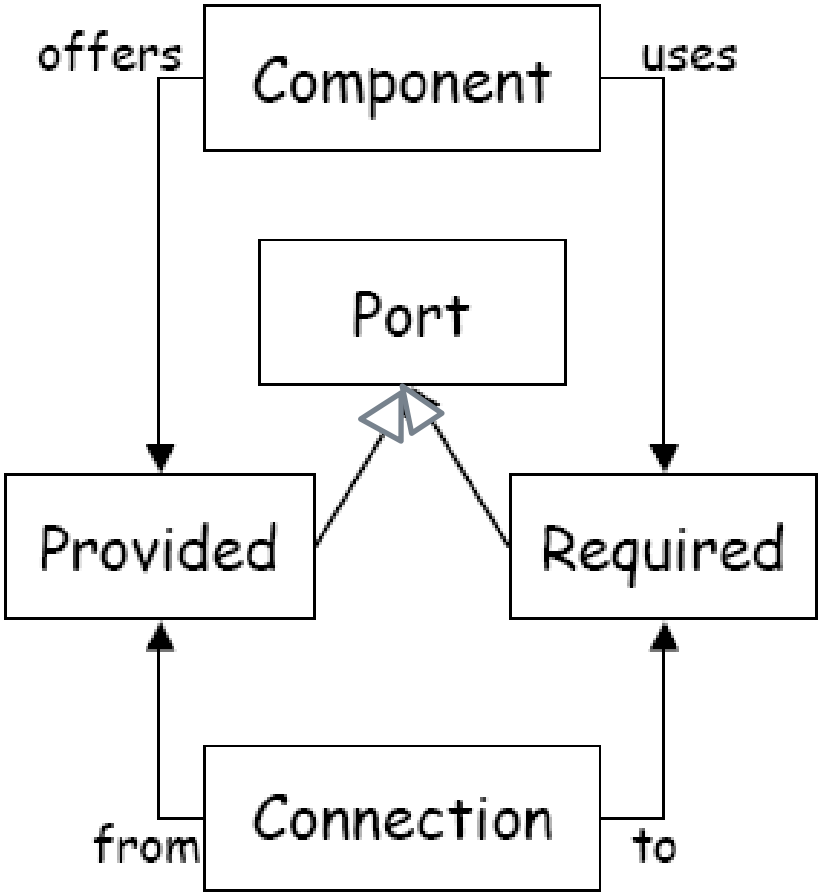
MOF Architechure - Example



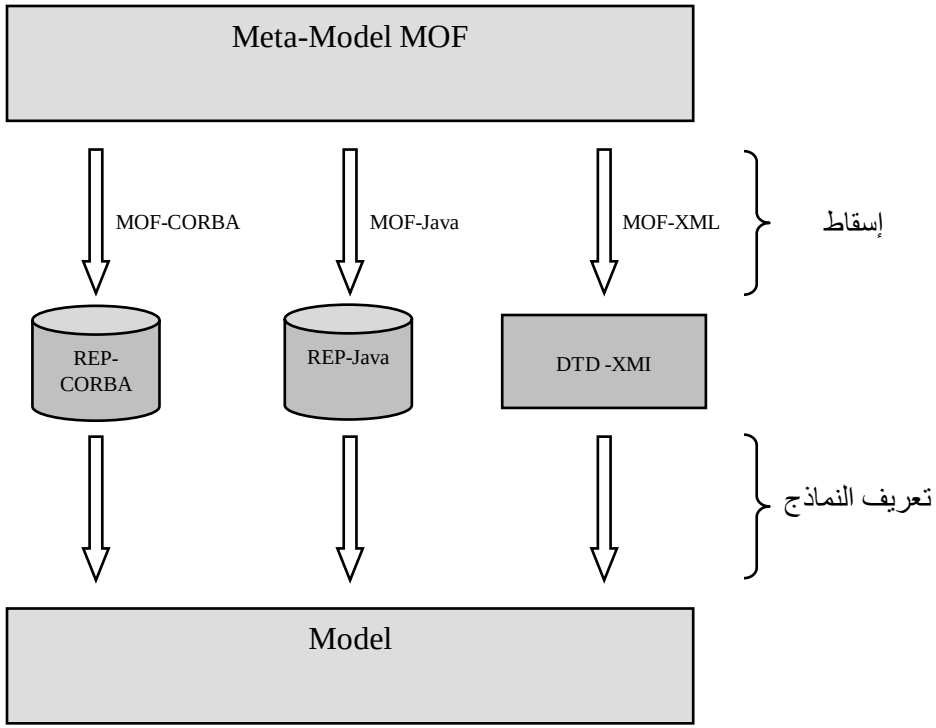
MOF Architechure – Example
M2 for Java ➔ Meta-programming



MOF Architechure – Example
M2 for Distributed Components



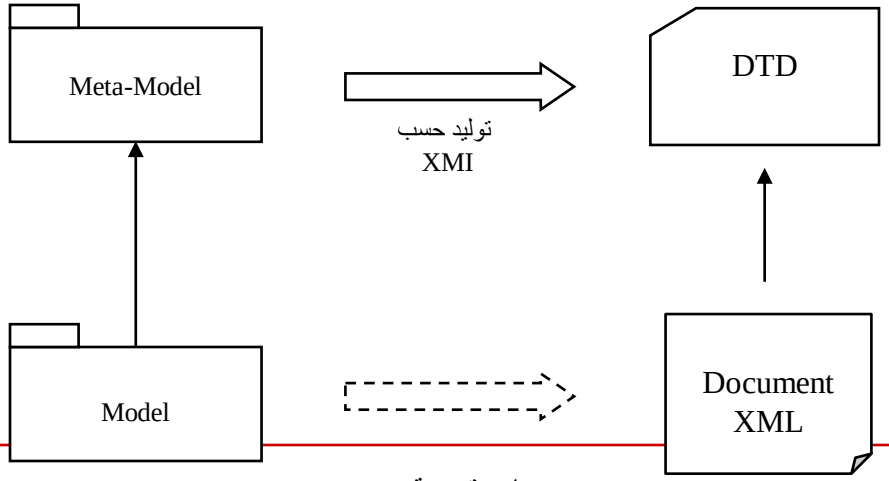
From MOF to tools



- MOF يقترح مصطلح الـ Repository كوسيلة لمعالجة و تخزين النماذج بشكل فيزيائي.
  - تخزين النماذج يمكن أن يتم ضمن قاعدة بيانات، ضمن ملف أو ضمن الذاكرة فقط .
- الـ Repository هو مجموعة من العمليات التي تسمح بمعالجة نوع من النماذج. MOF
- يعرف فقط القواعد التي تسمح اعتباراً من Meta-Model بتوليد واجهة الـ Repository.
- الـ MOF مستقل عن تقنيات التنفيذ
  - يعرف عدة إسقاطات نحو التقنيات الأكثر أهمية CORBA, XML, Java.

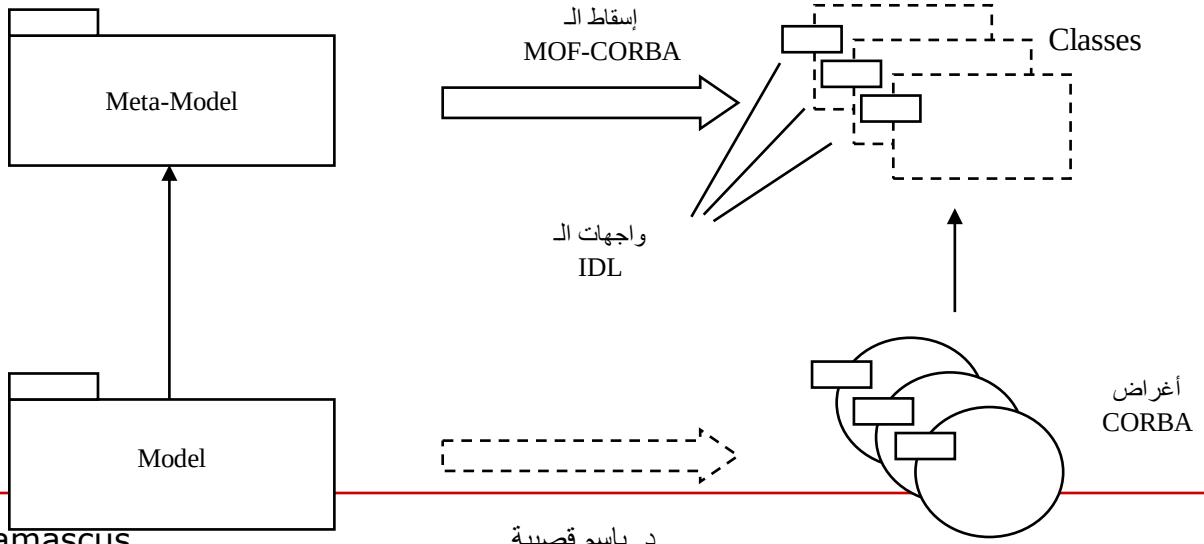
Mapping MOF – XML = XMI

- XMI(XML Meta-Data Interchange) = لدعم تبادل تعريفات الـ Meta-Models بين الأدوات.
- XMI تعرف مجموعة القواعد التي تسمح بتوليد الـ DTD لوثيقة XML اعتباراً من Meta-Model
  - إمكانية تبادل النماذج الموافقة لهذا الـ Meta-Model على هيئة ملفات XML موافقة للـ DTD المولدة
- XMI تقترح أيضاً قواعد التي تسمح بتمثيل النماذج على هيئة ملفات XML بدون توليد الـ DTD

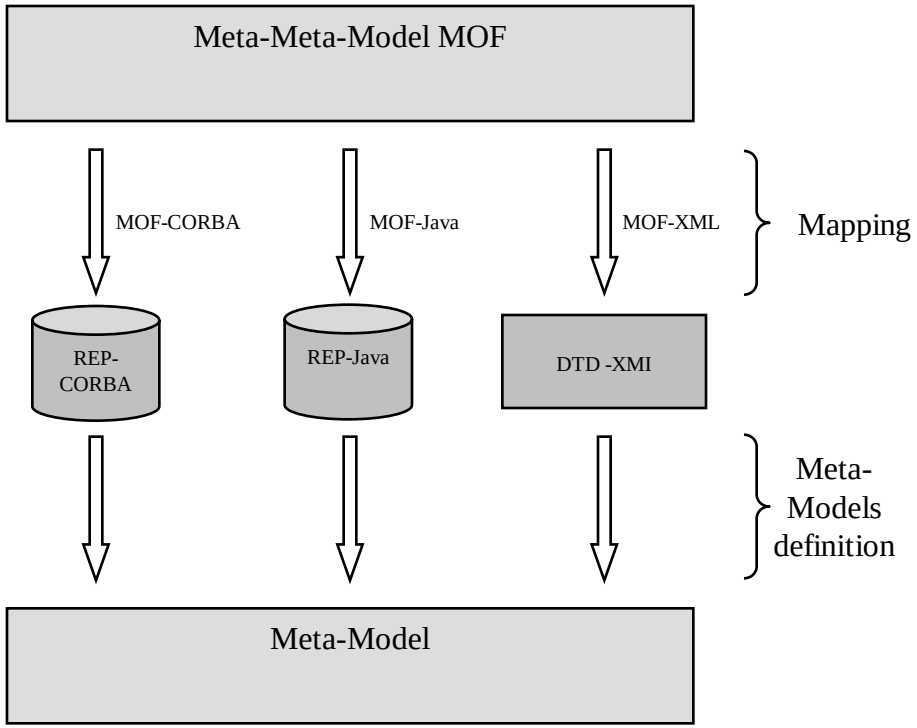


Mapping MOF - CORBA

- MOF يعرف القواعد التي تسمح بتوليد واجهات الـ IDL اعتباراً من Meta-Model.
  - تنفيذ هذه الواجهات يسمح بالحصول على Repositories التي تسمح بتمثيل النماذج على هيئة أغراض CORBA.
- الواجهات IDL ترث من واجهات عامة التي تدعم الـ Reflexivity
  - إمكانية تطوير أدوات قادرة على استثمار الواجهات العامة لمعالجة النماذج المخزنة بدون معرفة نوعها.



Tools for MOF



- الـ MOF يمكن اعتباره M2 لأنه يوافق لنفسه.
  - => يمكننا تطبيق القواعد المعرفة من أجل الـ M2 على الـ M3
  - => و بالتالي نحصل على M2 Repositories

استثمار النماذج

- مقدمة
- تحويلات النماذج
- MDE
- MDA

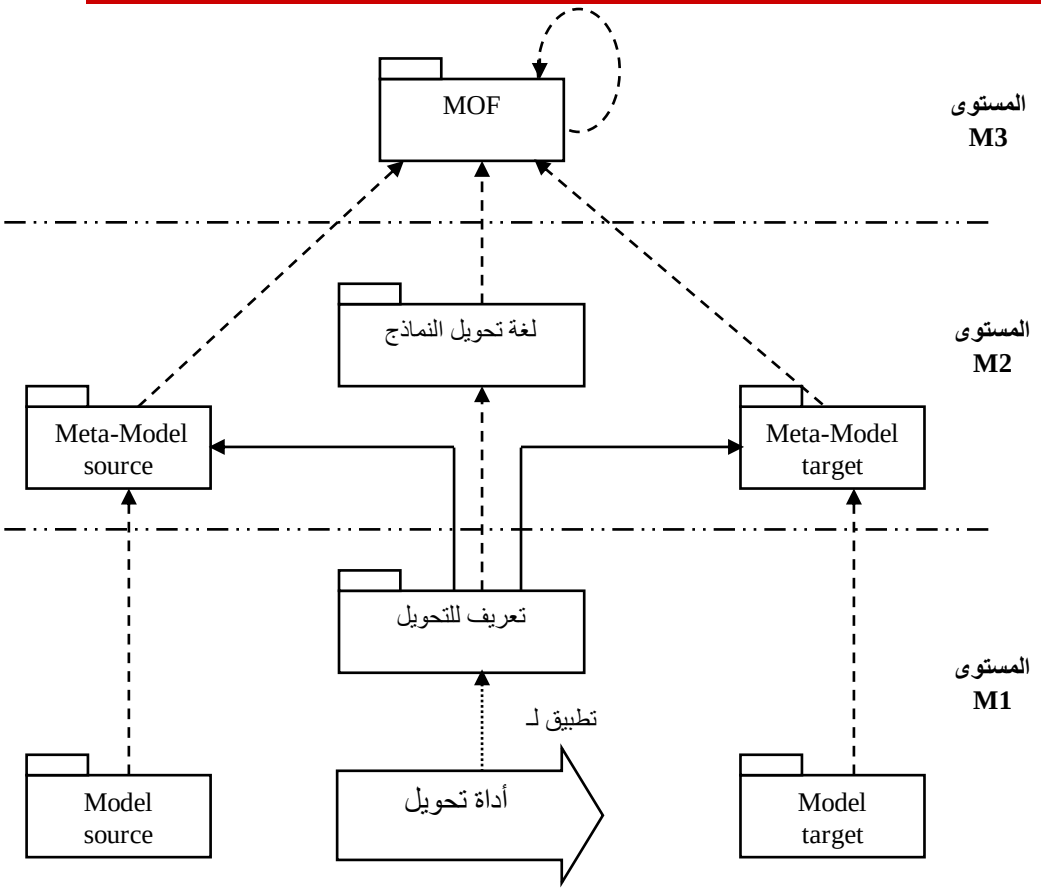
مقدمة

- النماذج تسمح بفهم و معالجة تجريد من النظام حسب وجهة نظر معينة.
- الاتجاه الحالي لهندسة البرمجيات يحاول أتمتة تطوير الأنظمة اعتباراً من النماذج التي تصف تجريداتها.
  - النماذج يجب أن تكون قابلة للتفسير من قبل الحاسب.
  - إمكانية تطبيق عمليات كالدمج و الإسقاط على النماذج بهدف الانتقال من مستوى مجرد إلى آخر أو إلى شيفرة النظام.

تحويلات النماذج

- يمكن امتلاك عدة نماذج التي تخص نفس النظام.
  - هذه النماذج تصف الأجزاء المختلفة للنظام
  - هذه النماذج تصف النظام حسب عدة وجهات نظر
  - على عدة مستويات تجريد مختلفة.
- بما أن عدة نماذج تناقش نفس النظام فإنه يتوجب :
  - الانتقال من نموذج مجرد إلى نموذج أقل تجريداً.
  - دمج النماذج التي تصف عدة مظاهر من النظام في نموذج واحد
- تحويل النماذج هو مختلف العمليات التي نطبقها على نموذج أو مجموعة نماذج بهدف إنتاج نموذج آخر أو شيفرة النظام.

لغة تحويل النماذج
QVT (Query View Transformation)



- تعريف تحويل النماذج = مجموعة من قواعد التحويل.
- قاعدة التحويل تحدد كيف أن مفهوم من النموذج المصدر أو أكثر يتحول إلى مفهوم أو أكثر من النموذج الهدف .

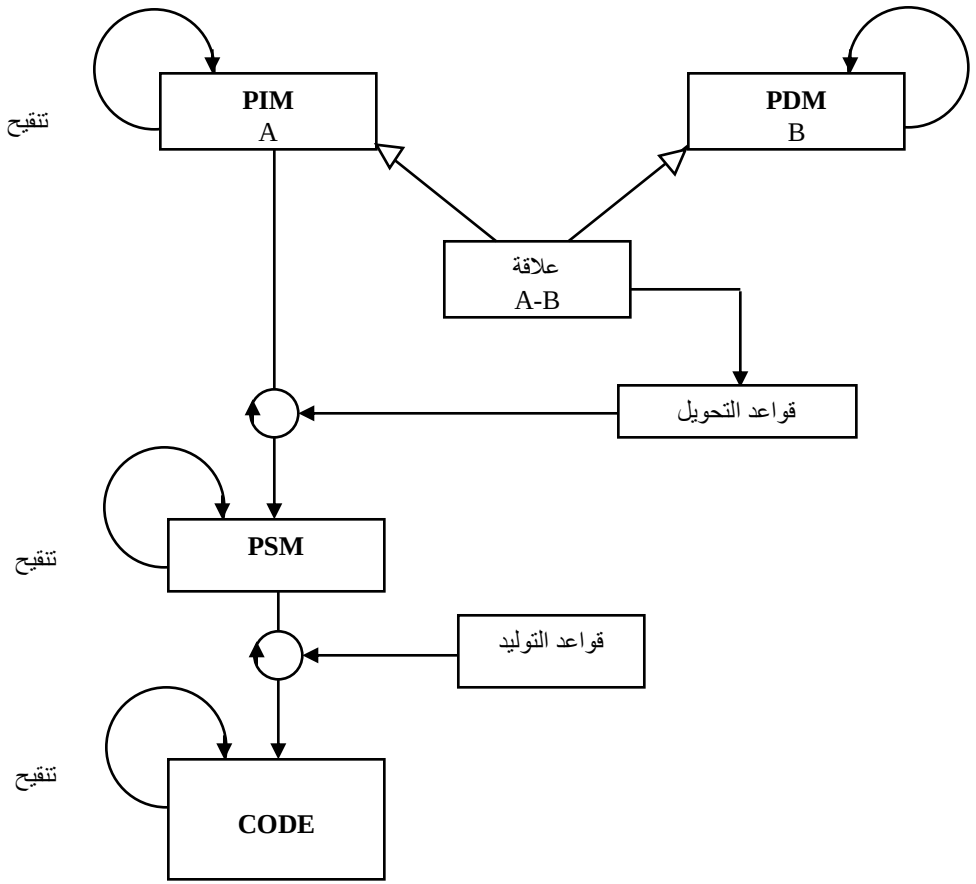
MDE (Model Driven Engineering)

- فكرة MDE = استخدام النماذج و تحويلات النماذج لتنظيم فعالية تطوير نظام ما
- MDE يسمح بشكل أفضل بالتعبير عن المشكلة. إنه يشجع علي تحديد و فصل الاهتمامات المختلفة لنظام. ما يسمح بنمذجة كل اهتمام بشكل مستقل عن الآخر و من ثم نعرف دمج توصيفات النظام.
- MDE يسمح بتوصيف منهجية لتعريف المشكلة و كيفية الذهاب إلى حلها. إنه يقسم فعالية تطوير البرمجيات على عدة مستويات من التجريد. النماذج في مستوى معين تختلف عن النماذج في مستوى آخر بدقتهم التقنية أو المهنية. الانتقال بين هذه المستويات يتم عبر تحويلات النماذج.
- MDE يسمح برأسملة التوصيفات البدائية للنظام و رأسملة معرفة تطوير هذا النظام. معرفة تطوير النظام تُعرف ضمن تحويلات النماذج. هذا يسمح بامتلاك وثائق محدثة لشيفرة النظام.
- ميزة هامة للـ MDE هي أن فعالية التطوير لبرمجية ما بإمكانها أن تتكيف بسرعة لتغيير في شروط تحقيق النظام. الآثار المترتبة عن تغيير ما في شروط النظام يمكن حصرها بسهولة بينما يمكن إعادة استخدام الأجزاء الأخرى الغير مصابة.
- الـ MDE يمثل تطور ضمن مجال تطوير البرمجيات حيث أنه يرفع مستوى التطوير من التجميع المباشر للأغراض نحو تحويلات النماذج. هذه التحويلات مقادة بالنماذج المعرفة في مستوى تجريد أعلى مما يسمح بأتمتة عملية تجميع الأغراض => هكذا فإننا نتجه نحو جعل تطوير البرمجيات قابلة للتصنيع.

تعريف فعالية MDE

- كم مستوى تجريد يتواجد؟ و ما هي المنصات التي يجب دمجها؟
- ما هي المفردات المجردة للاستخدام في كل مستوى تجريد؟
- ما هي المعلومات الإضافية للدمج في مستوى التجريد الأخفض؟
- كيف نولد الشيفرة؟
- كيف نتوثق من نموذج بالمقارنة مع نماذج المستوى الأعلى ؟

MDA (Model Driven Architechure)



- MDA هي حالة خاصة من الـ MDE حيث الـ MDA تتوقف عند مشكلة تعدد تقنيات التنفيذ
- الـ OMG يقترح الـ MDA بهدف الحفاظ على الاستثمارات الفكرية و المالية المنجزة أثناء تطوير النظم وذلك عندما تقنية التنفيذ تتغير أو ترتقي
- الفكرة الرئيسية للـ MDA هي الفصل أثناء بناء النظام بين التصميم المهني و التحقيق التقني لهذا التصميم

مفردات الـ MDA

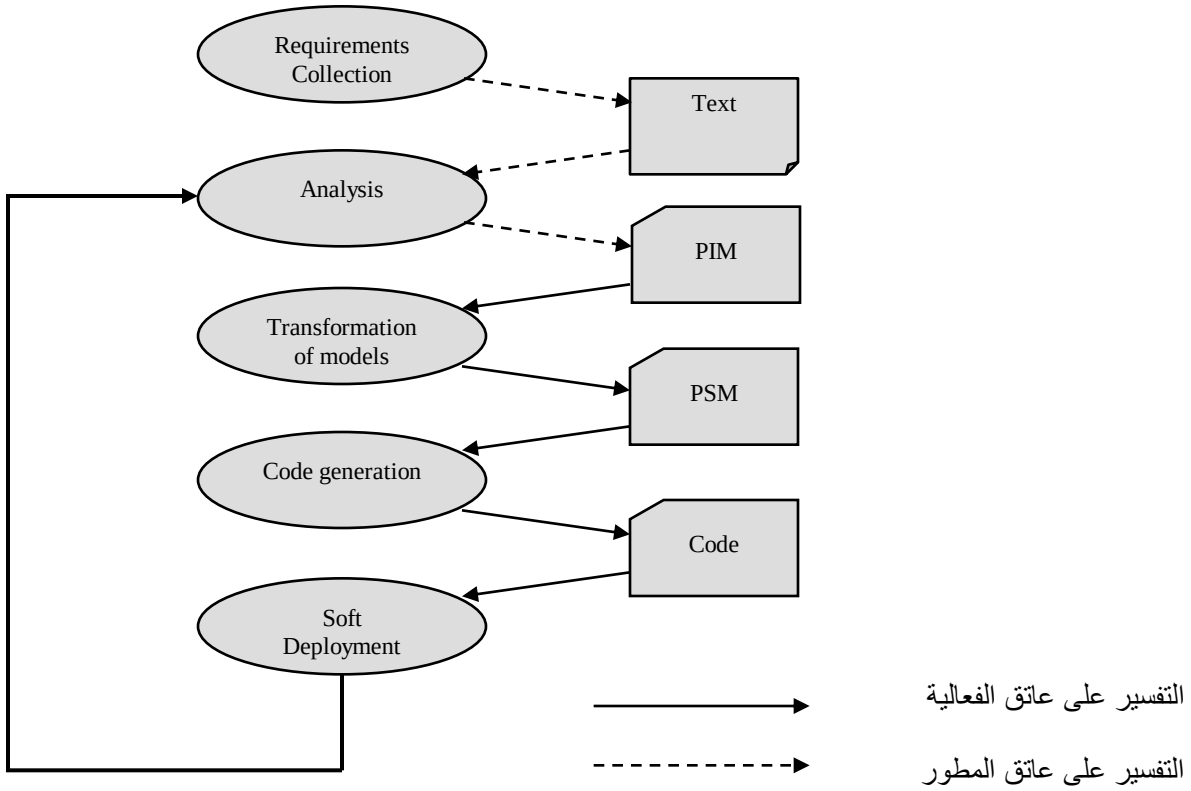
- PIM (Platform Independent Model) يمثل النموذج المهني المستقل عن أي تقنية تنفيذ و يعبر عن مشكلة باستخدام مفاهيم مجالها.
- PDM (Platform Definition Model) يمثل تعريف لمفاهيم تقنية التنفيذ المستخدمة لتحقيق النموذج المهني.
- PSM (Platform Specific Model) يحوي نفس كائنات الـ PIM مع تفاصيل خاصة بتقنية التنفيذ المختارة.

تحقيق الـ MDA

- منحى ترجمة
  - نستخدم قواعد لتوليد الشيفرة النهائية و يمكن اعتبار هذا المنحى كترميم للـ DSLs.

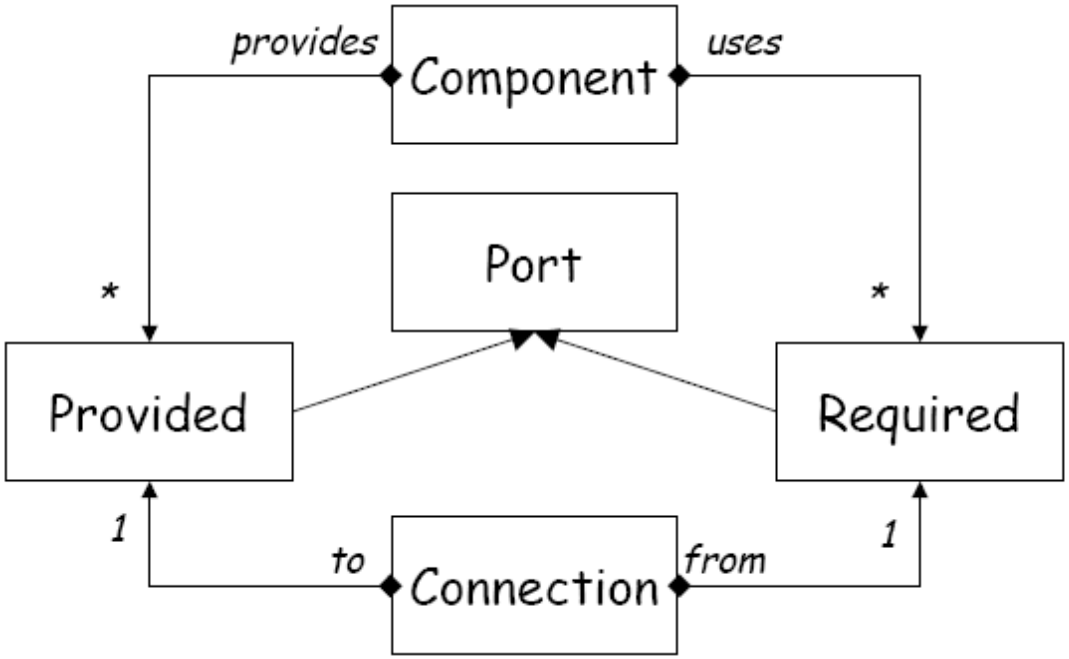
- منحى ارتقائي
  - يتم بناء التطبيق بشكل تدريجي و غالباً بشكل يدوي. هذا المنحى مشابه لمنهجية التطوير غرضية التوجه.

MDA & Soft Engineering process

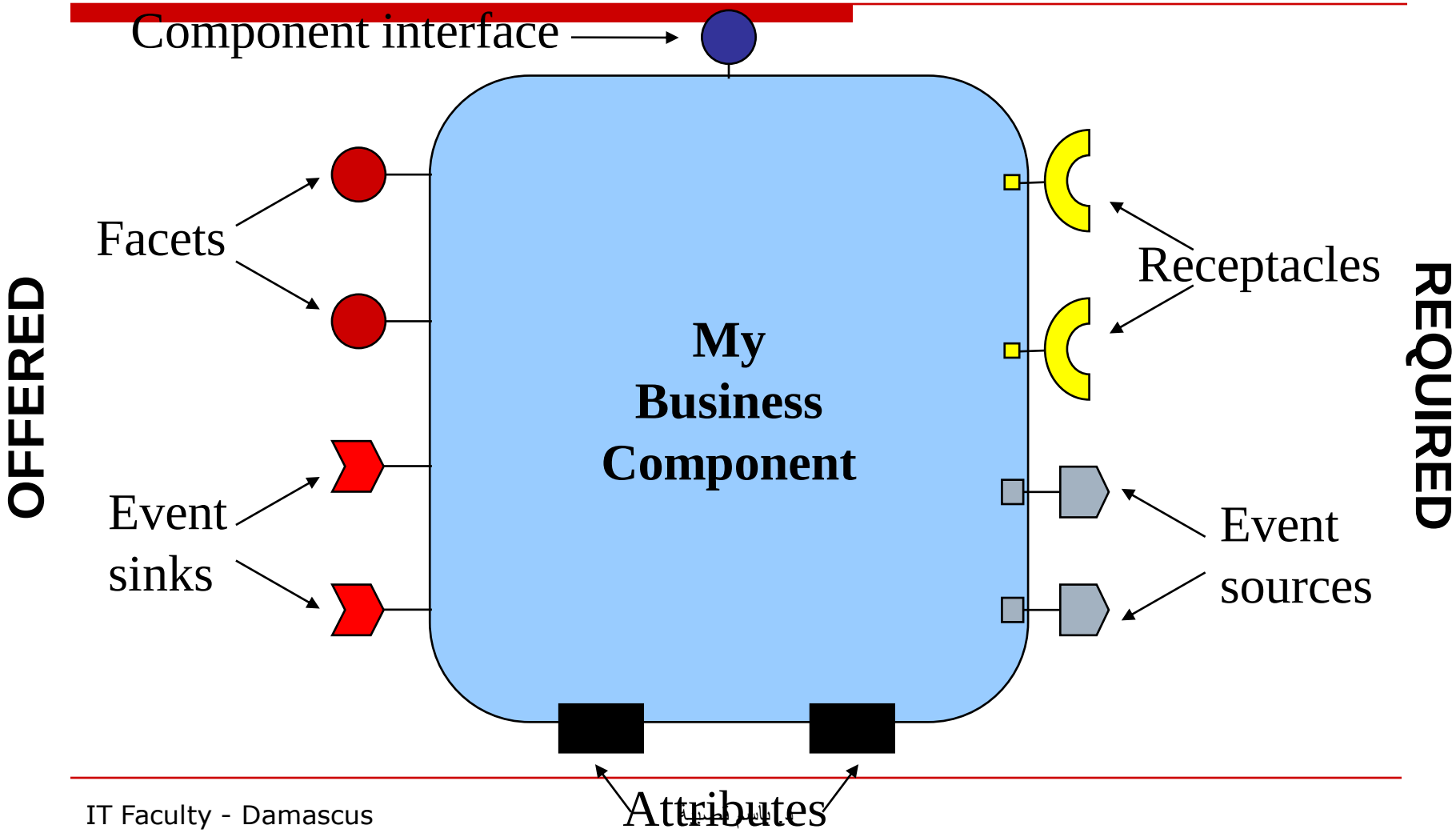


From Component to Java : مثال بسيط

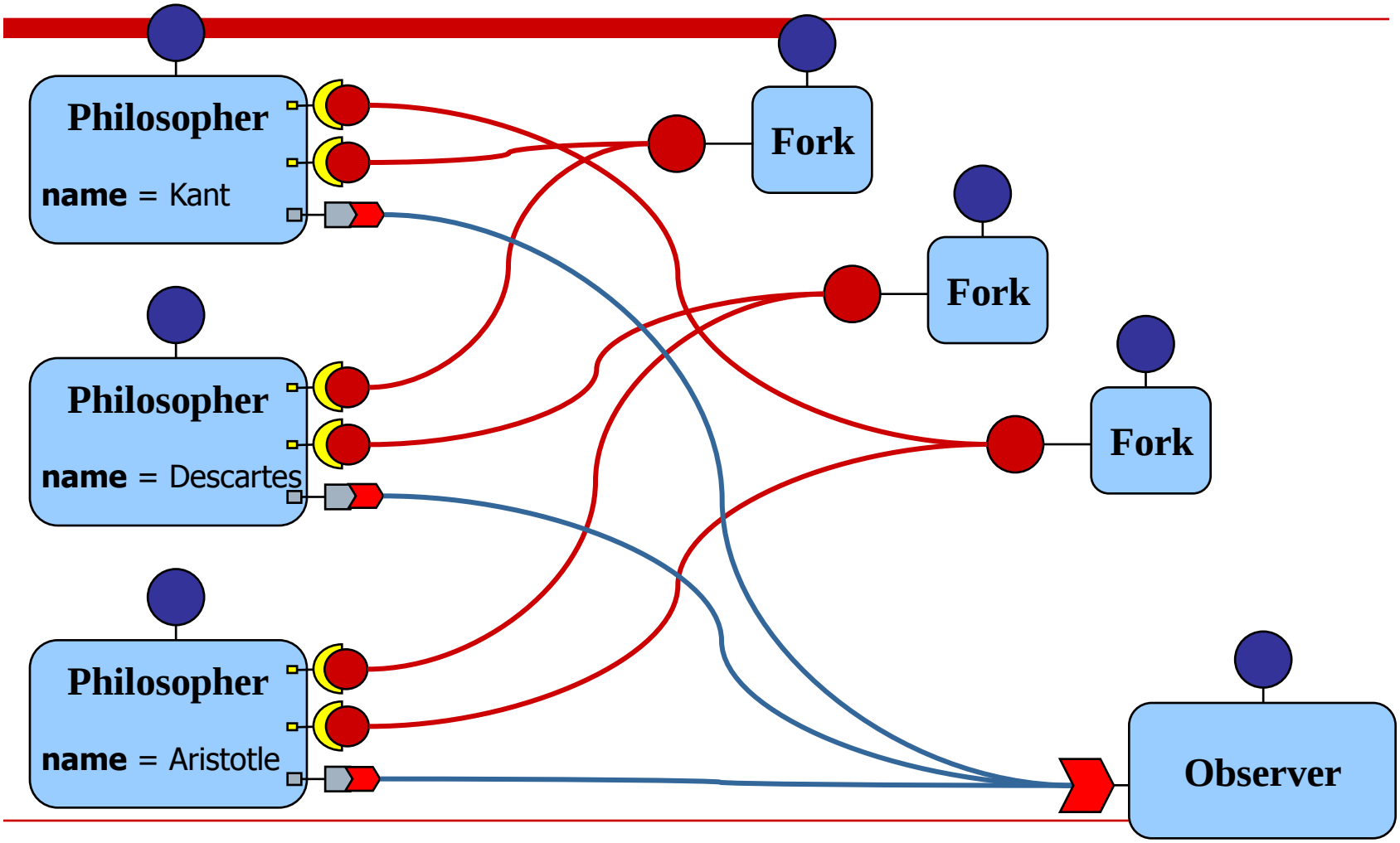
Component M2



From Component to Java : مثال بسيط

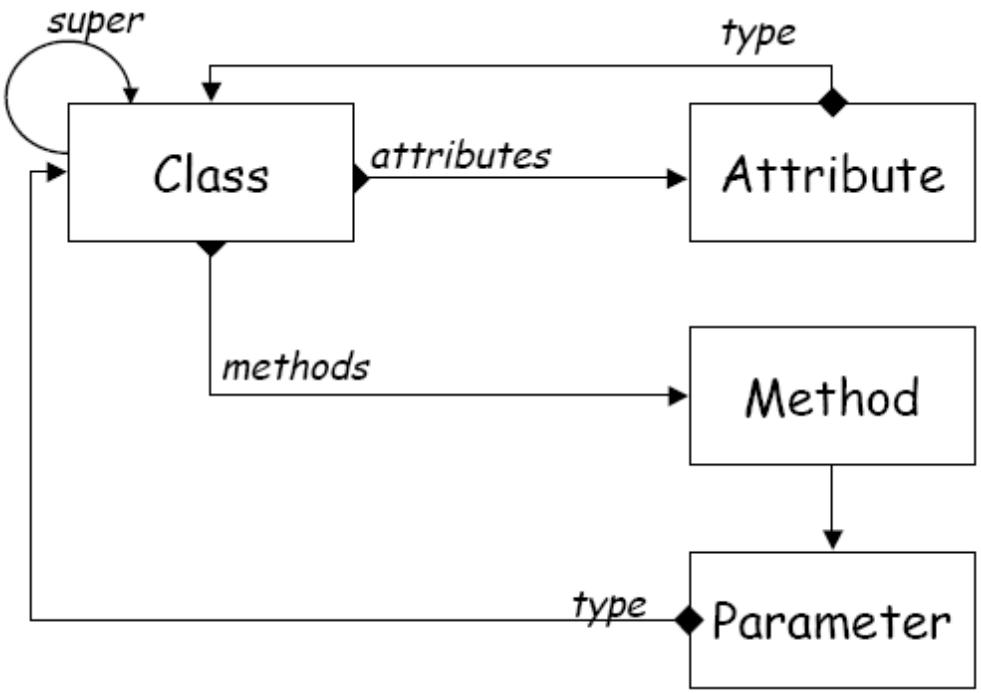


From Component to Java : مثال بسيط

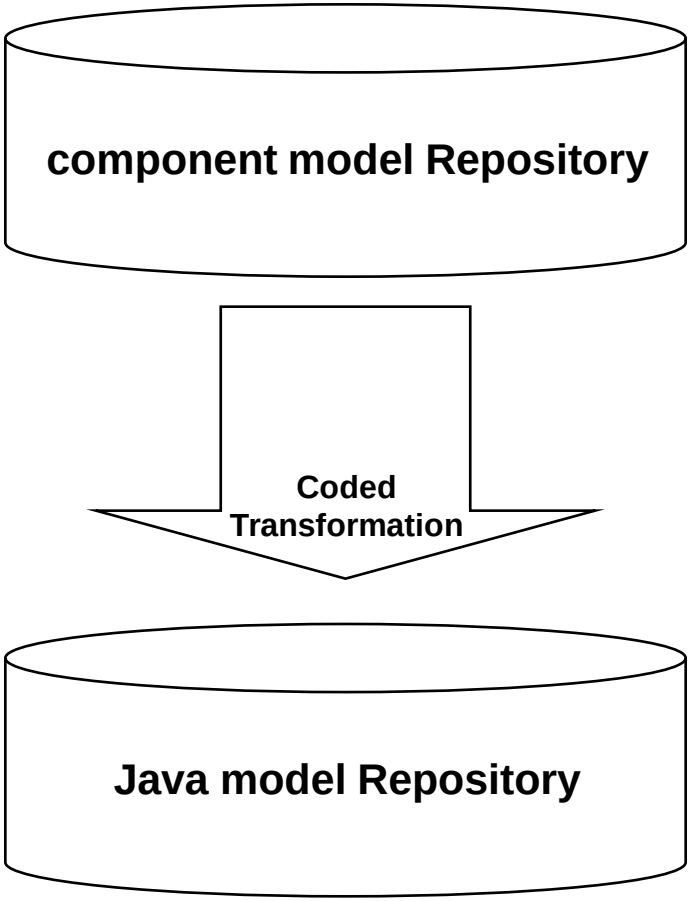


From Component to Java : مثال بسيط

Java M2



From Component to Java : مثال بسيط



From Component to Java : مثال بسيط

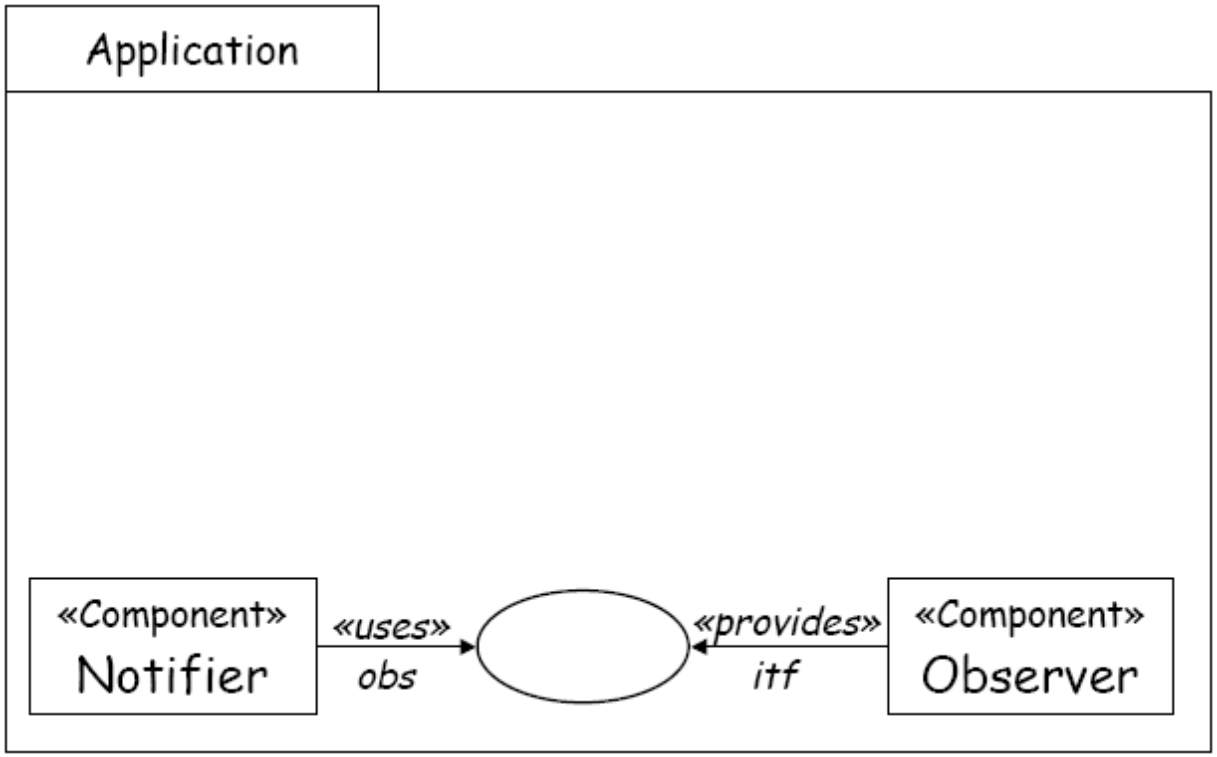
- الإسقاط من Component نحو java
- Port interface **>>>** Java class
- Component **>>>** Java class
  - Provide **>>>** private attribute
  - Provide **>>>** public method *provides_(itf)*
  - Uses **>>>**
  - Attribute **>>>**

Visitor pattern = قواعد الإسقاط

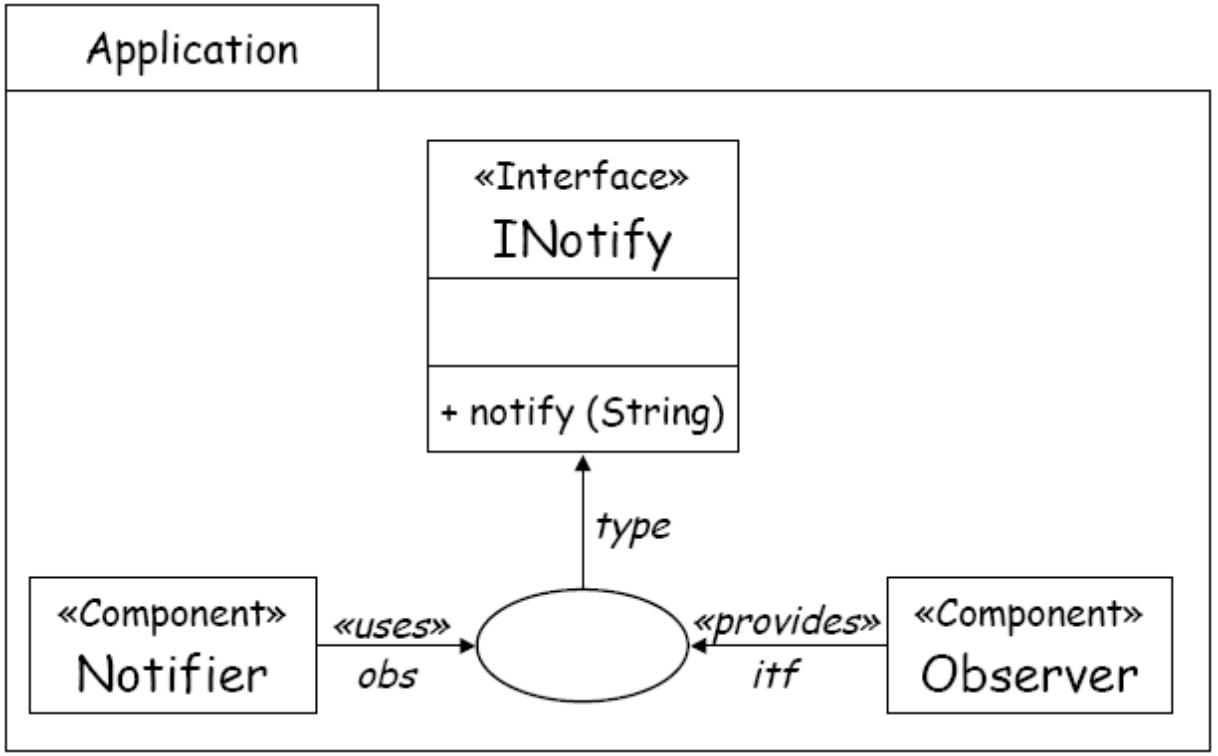
```
transformation comp2java
    rule comp2clss
        forall Component cc
        make Class c, jc.name = cc.name
        linking cc to jc by comp2clss
    rule itf2clss
        forall Interface it
        make Class cl, cl.name = it.name
        linking it to cl by citf2jclss
    rule op2meth
        forall Operation op
        where itf2class links op.itf to Y
        make Method me, me.name = op.name,
            Y.methods += me
        linking op to me by cop2jmeth
```

```
rule provides2java
   forall Port p
   where p.kind == "provides" &&
           citf2jclss links p.type to XX &&
           comp2clss links p.owner to YY
   make Attribut a, a.name = "_" + p.name,
          Argument a1, a1.type = XX,
          Method m,
          m.name = "provide_" + p.name,
          m.returntype = a1,
          YY.method += m, YY.fields += a
   linking p to a  by port2attr
   linking p to a1 by port2arg
   linking p to m  by port2meth
```

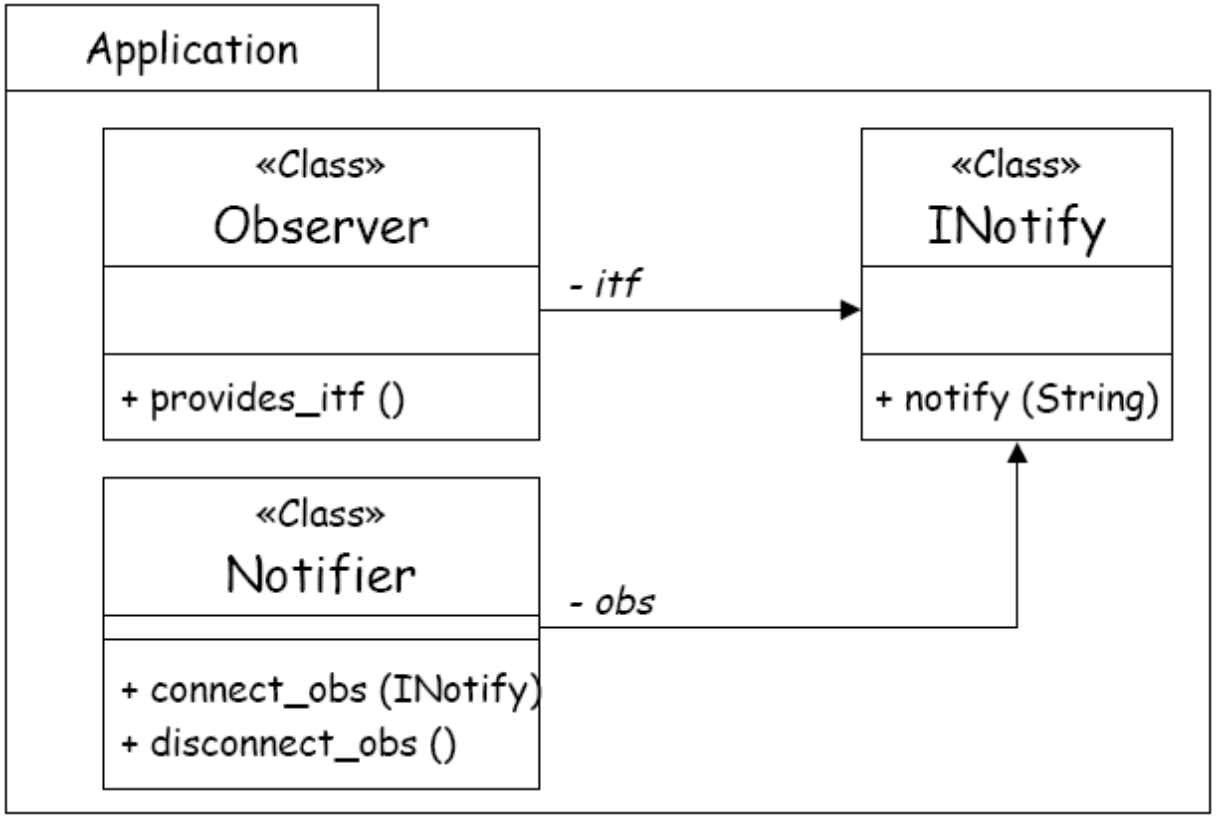
From Component to Java : مثال بسيط



From Component to Java : مثال بسيط



From Component to Java : مثال بسيط



From Component to Java : مثال بسيط

```
class INotify {
  void notify (String s) {
    // write code
  }
}
```

```
class Observer {
  private INotify _itf ;
  INotify provides_itf () {
    // return this._itf ;
  }
}
```

```
class Notifier {
  private INotify _obs ;
  void connect_obs (INotify o) {
    // this._obs = o ;
  }
  void disconnect_obs () {
    // this._obs = null ;
  }
}
```

From Component to Java : مثال بسيط (component model 1/2)

```
<?xml version="1.0"?>
<compmodel>

  <basetype>
    <bname>String</bname>
  </basetype>

  <basetype>
    <bname>void</bname>
  </basetype>

  <port>
    <name>INotify</name>
    <operation>
      <return>void</return>
      <name>notify</name>
      <parameter>
        <name>msg</name>
        <type>String</type>
      </parameter>
    </operation>
  </port>
```

From Component to Java : مثال بسيط (component model 2/2)

```
<component>
  <name>Observer</name>
  <provides>
    <type>INotify</type>
    <name>itf</name>
  </provides>
</component>
```

```
<component>
  <name>Notifier</name>
  <uses>
    <type>INotify</type>
    <name>obs</name>
  </uses>
</component>

</compmodel>
```

From Component to Java : مثال بسيط (Java model ½)

```
<?xml version="1.0" encoding="UTF-8"?>
<javamodel>

 <class>
  <name>INotify</name>
  <method>
   <return>void</return>
   <name>notify</name>
   <parameter>
    <name>msg</name>
    <type>String</type>
   </parameter>
  </method>
 </class>
```

```
<class>
 <name>Observer</name>
 <attribute>
  <type>INotify</type>
  <name>_itf</name>
 </attribute>
 <method>
  <name>provide_itf</name>
  <return>INotify</return>
 </method>
</class>
```

From Component to Java : مثال بسيط (Java model 2/2)

```
<class>
  <name>Notifier</name>
  <attribute>
    <type>INotify</type>
    <name>_obs</name>
  </attribute>
  <method>
    <name>connect_obs</name>
    <parameter>
      <type>INotify</type>
      <name>obs_</name>
    </parameter>
    <return>void</return>
  </method>

  <method>
    <name>disconnect_obs</name>
    <return>void</return>
  </method>
</class>

</javamodel>
```

From Component to Java : مثال بسيط