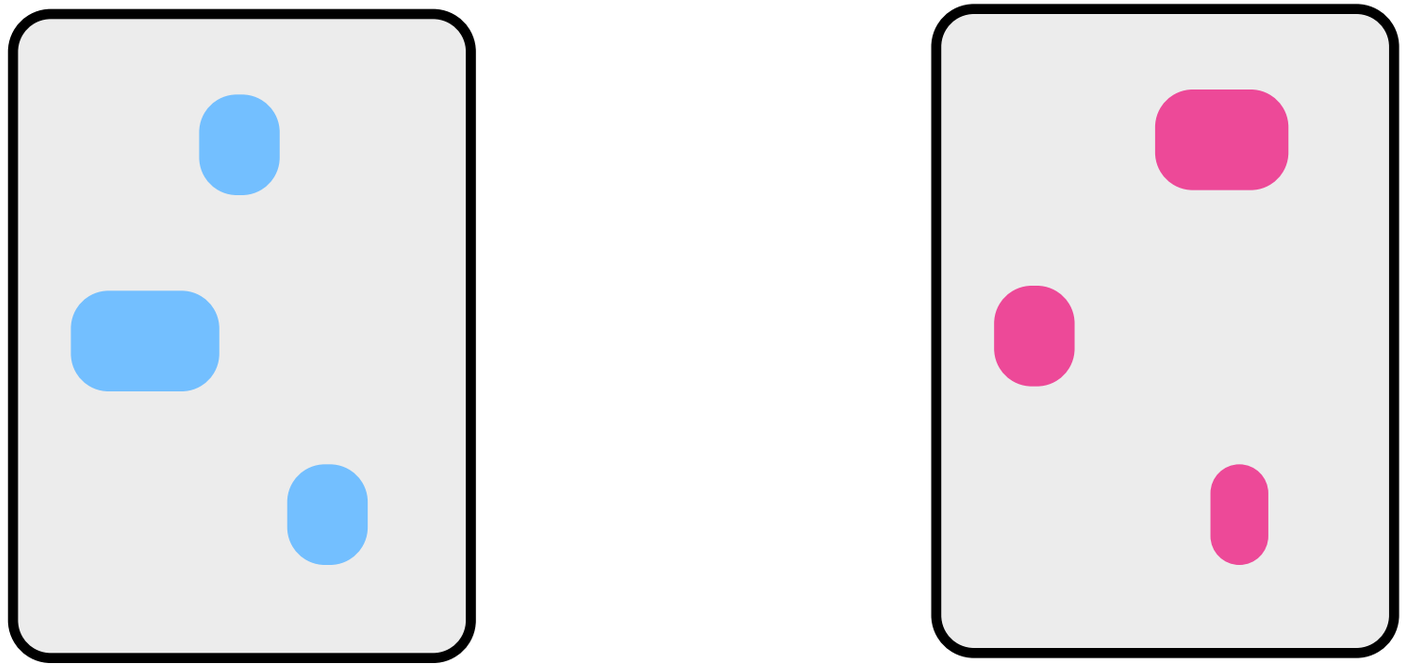
using twinning to adapt programs to alternative apis

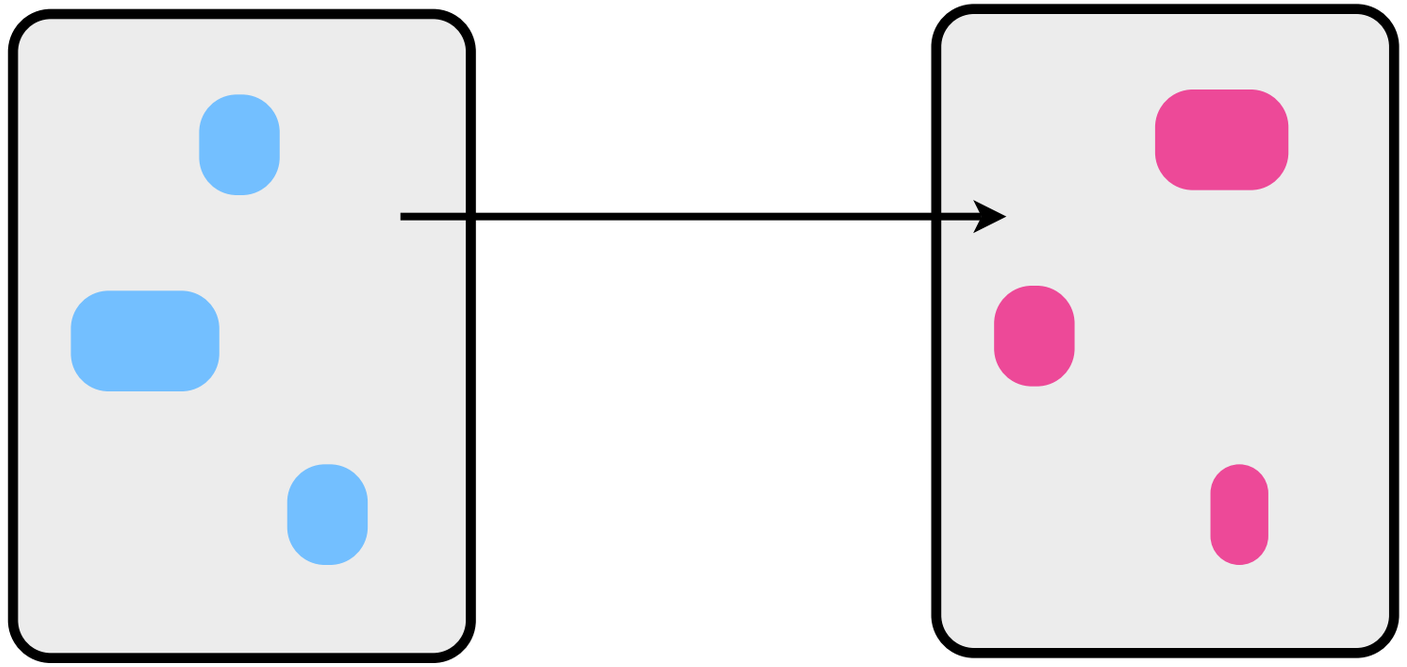
marius nita
david notkin

university of washington
se.cs.washington.edu

related implementations

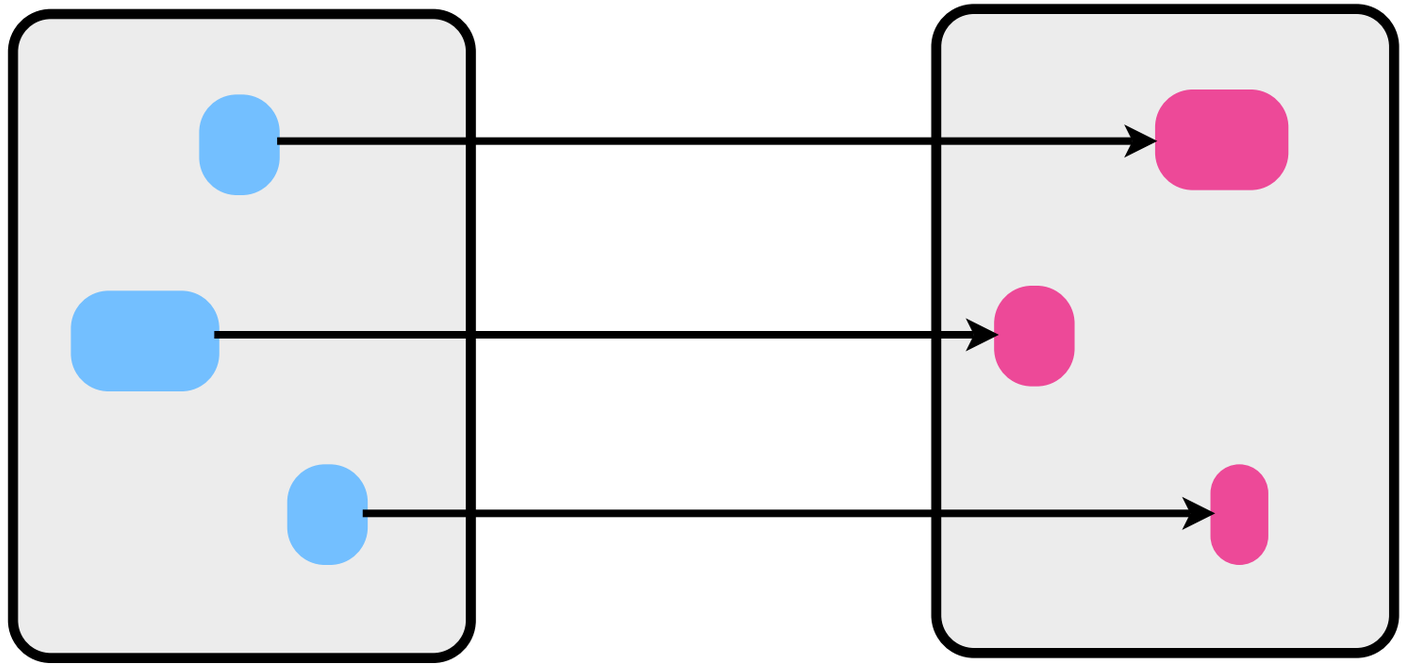


related implementations



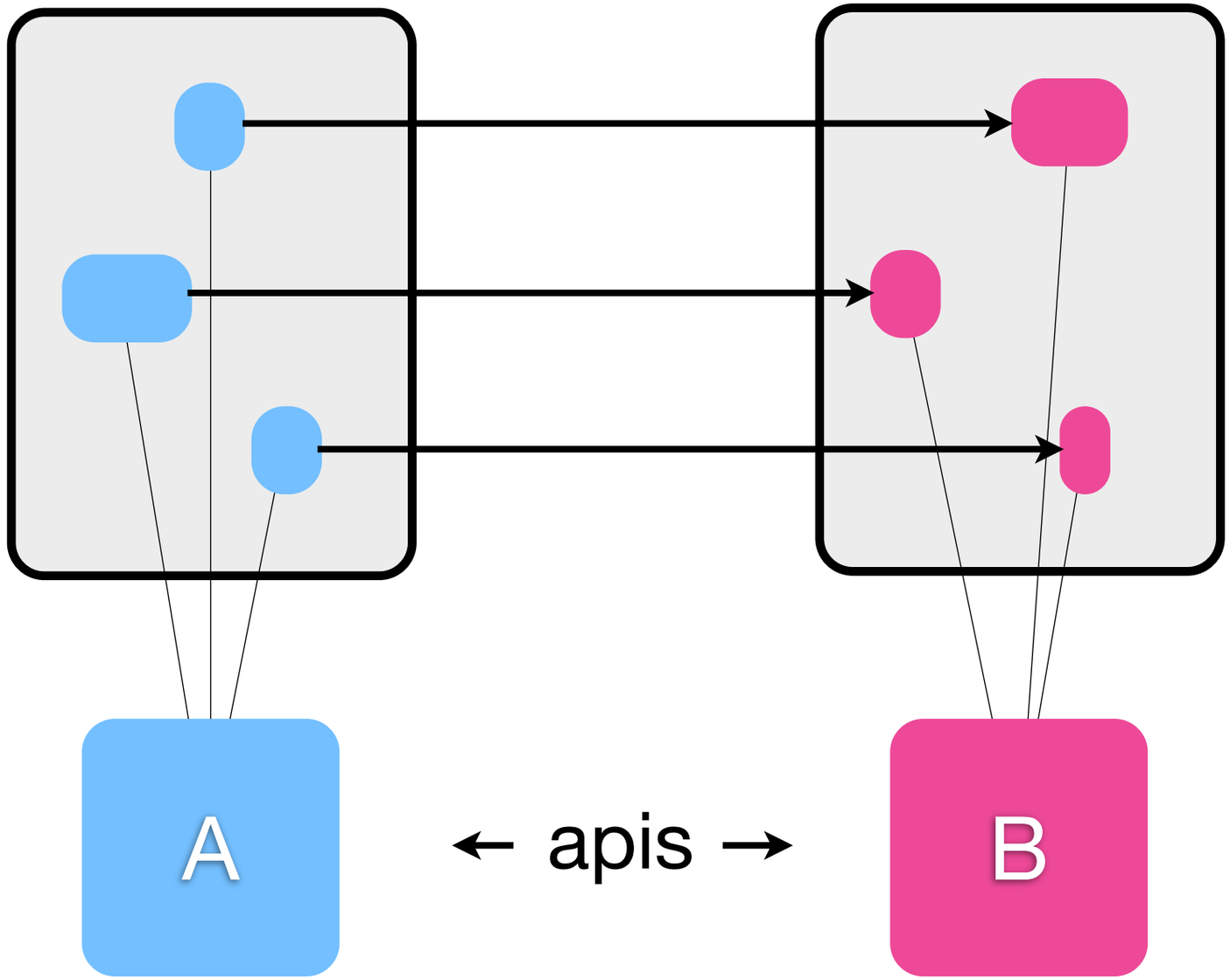
shared code

related implementations



differences

api alternatives



api alternatives

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

A

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

B

api alternatives are common

user choice

portability

structural reuse

api space is large; many tradeoffs

coevolution problem

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

coevolution problem

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

keep shared code in sync

coevolution problem

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

maintain the desired correspondence
among the differences

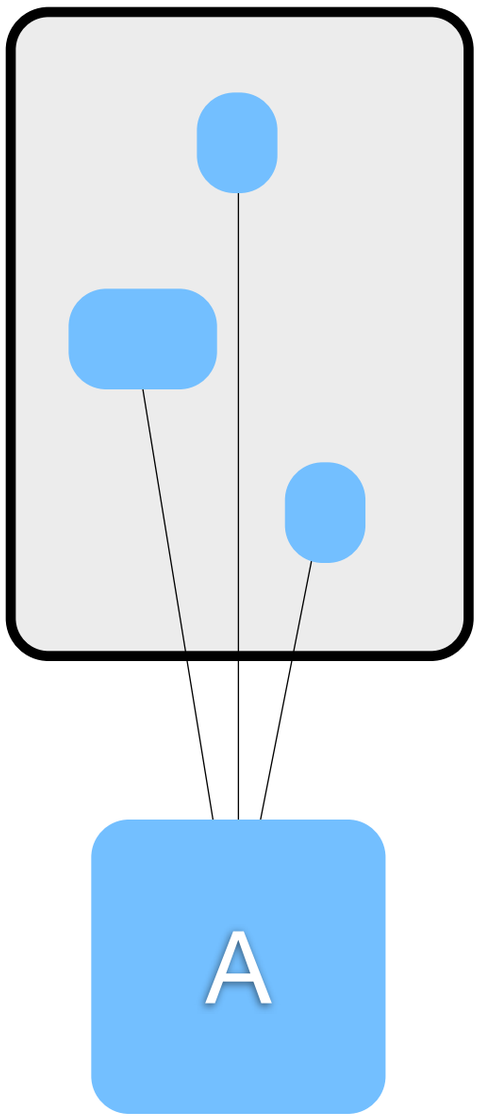
coevolution problem

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

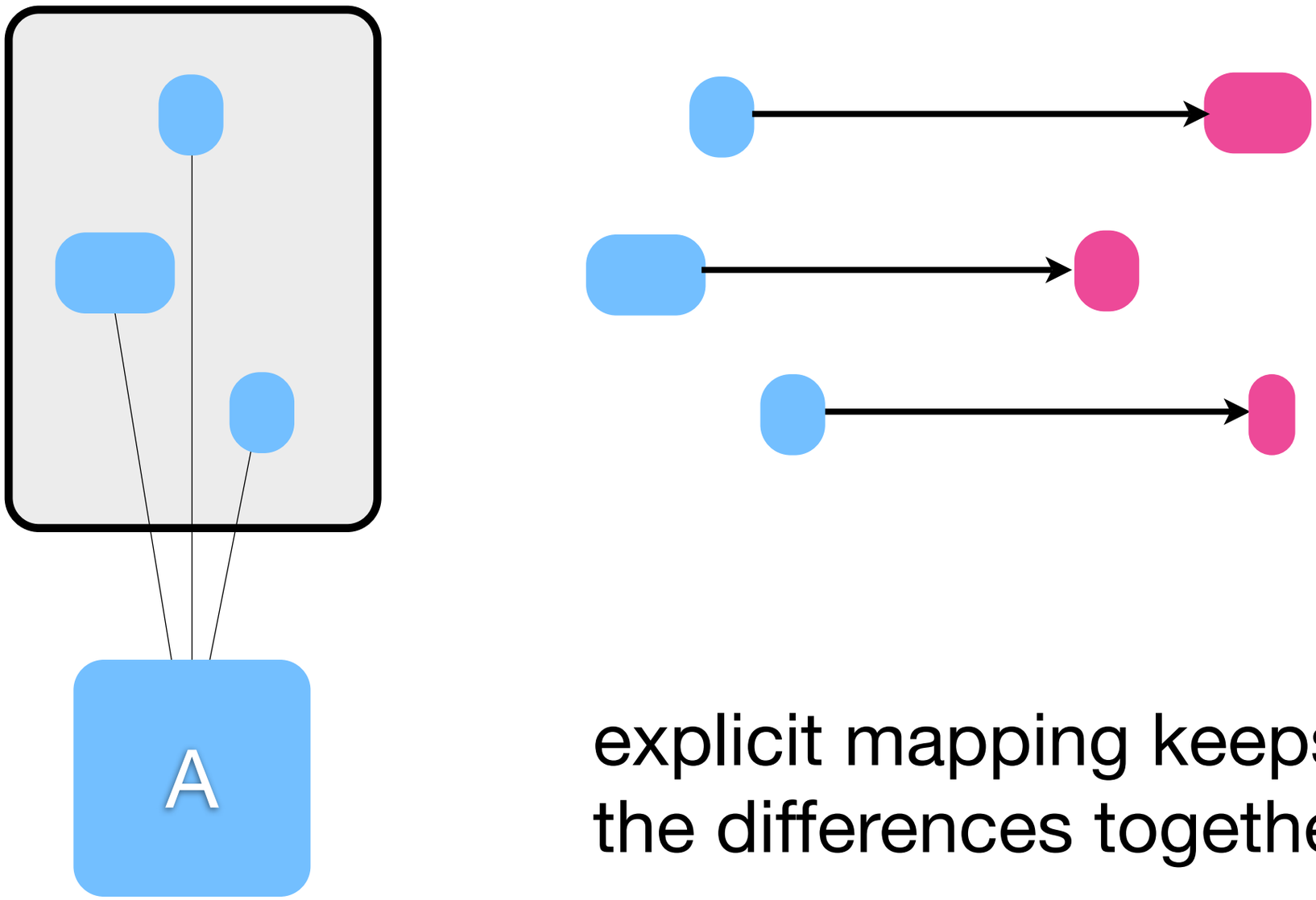
```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

abstraction is a common approach,
but often impossible

our approach: twinning

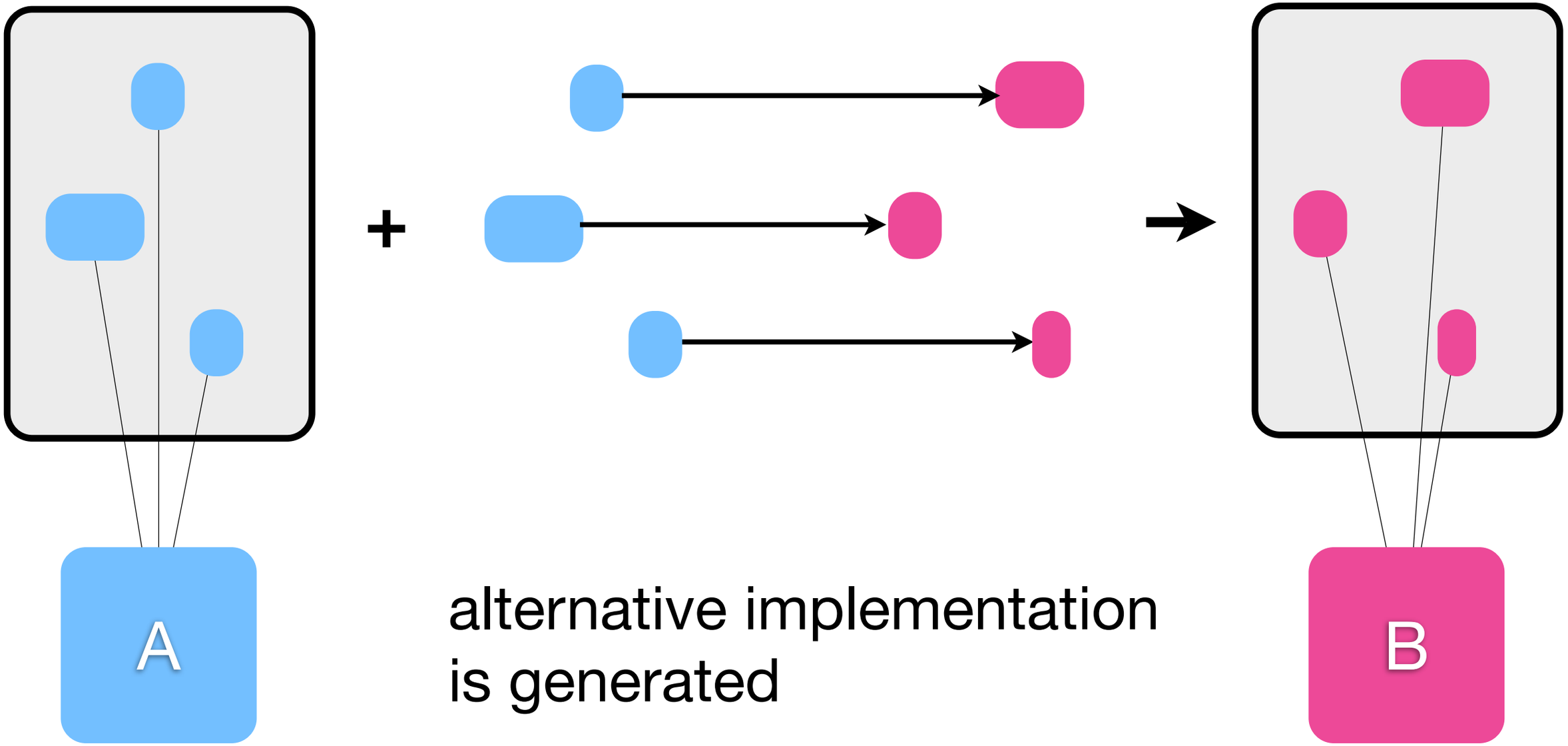


our approach: twinning



explicit mapping keeps
the differences together

our approach: twinning



alternative implementation
is generated

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(emp
  int id = getIntValue(empEl,"Id
  int age = getIntValue(empEl,"A

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name

  return e;
}
```

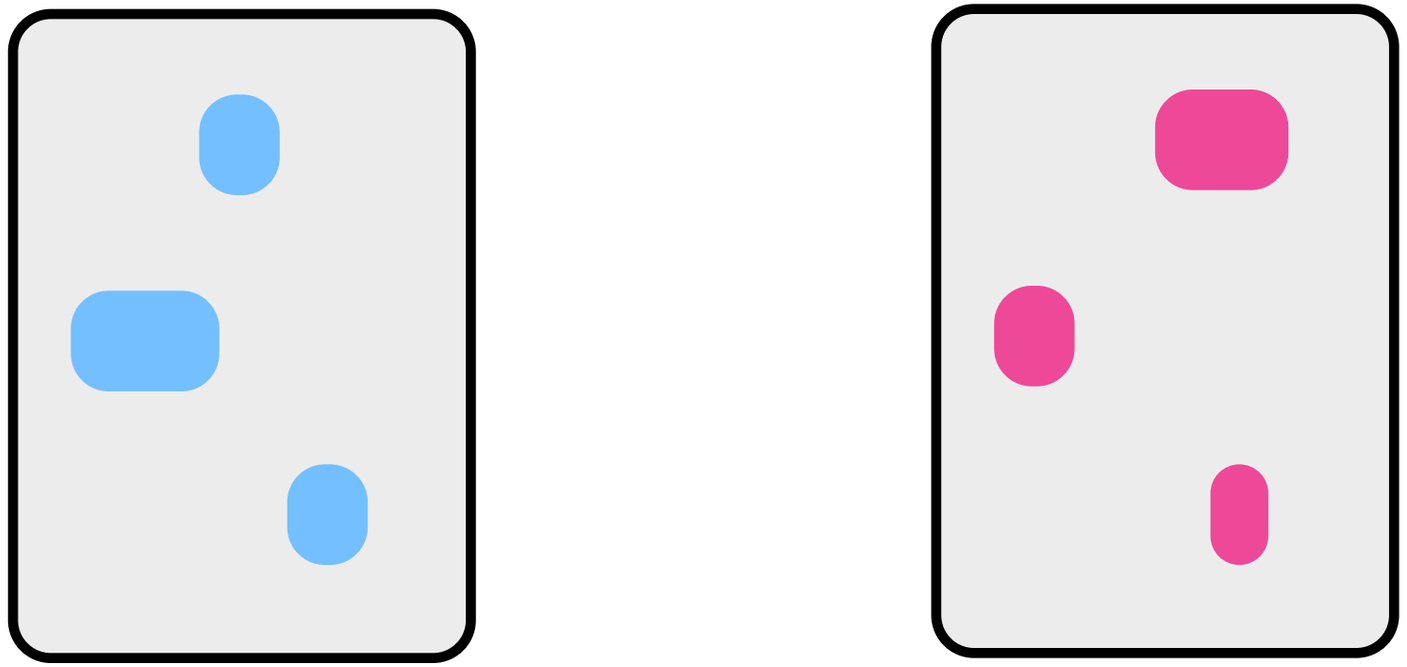
==>

```
String (Elem e, String attr) {
  return e.getAttribute(attr);
}
```

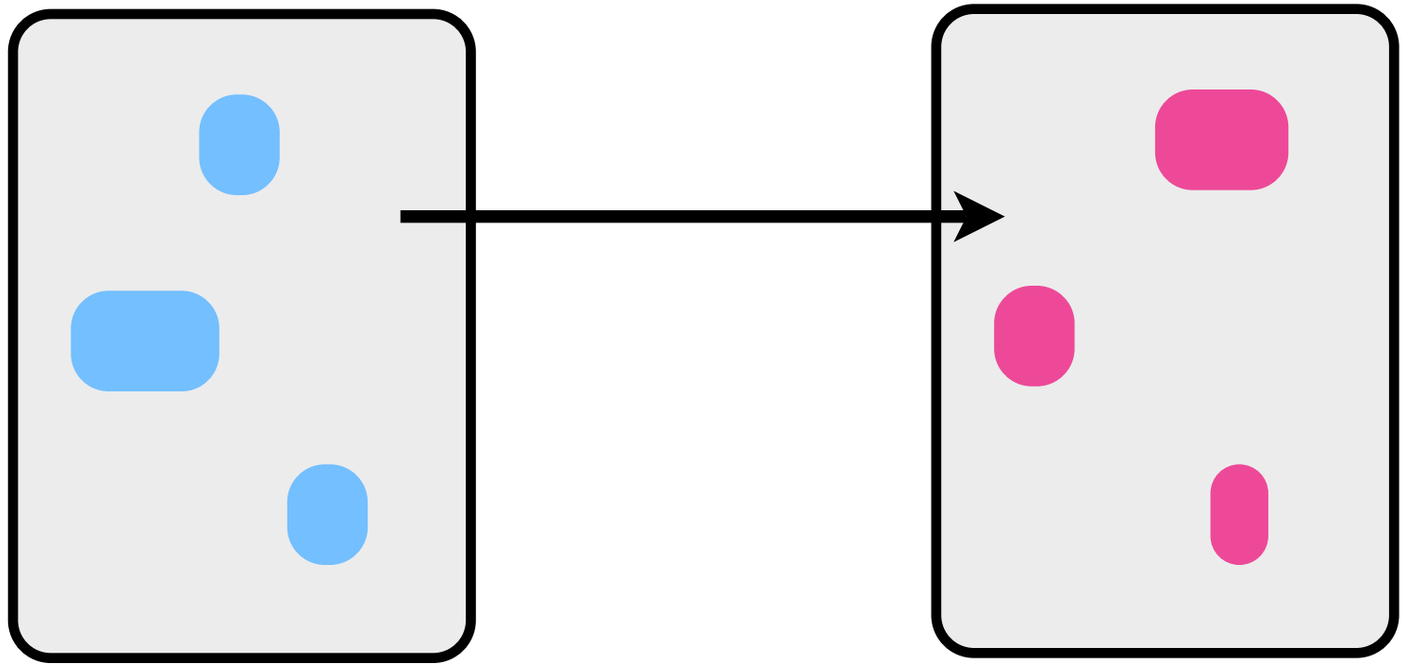
```
Employee getEmployee(Element empEl)

  getTextValue(empEl,"Name");
  ntValue(empEl,"Id");
  IntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  new Employee(name,id,age,type);

}
```

```
String (Element e, String attr) {
  return e.attributeValue(attr);
}
```

prior work: clones

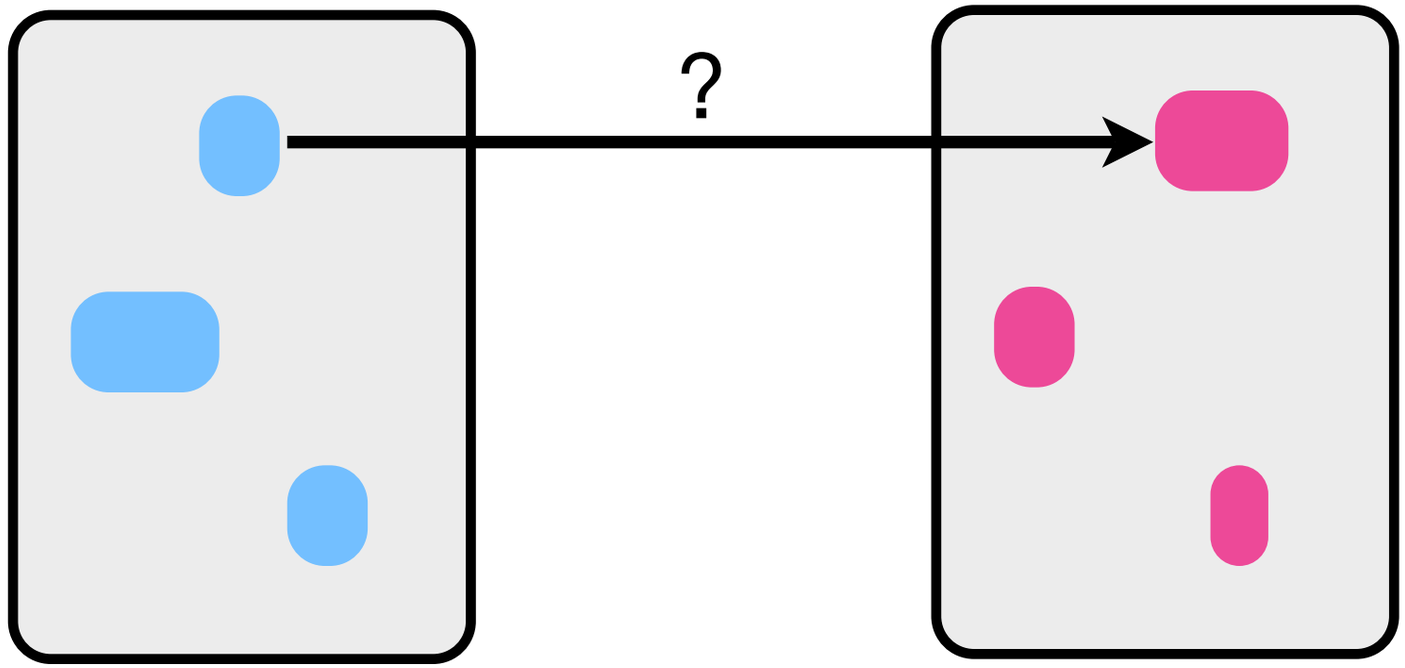


prior work: clones



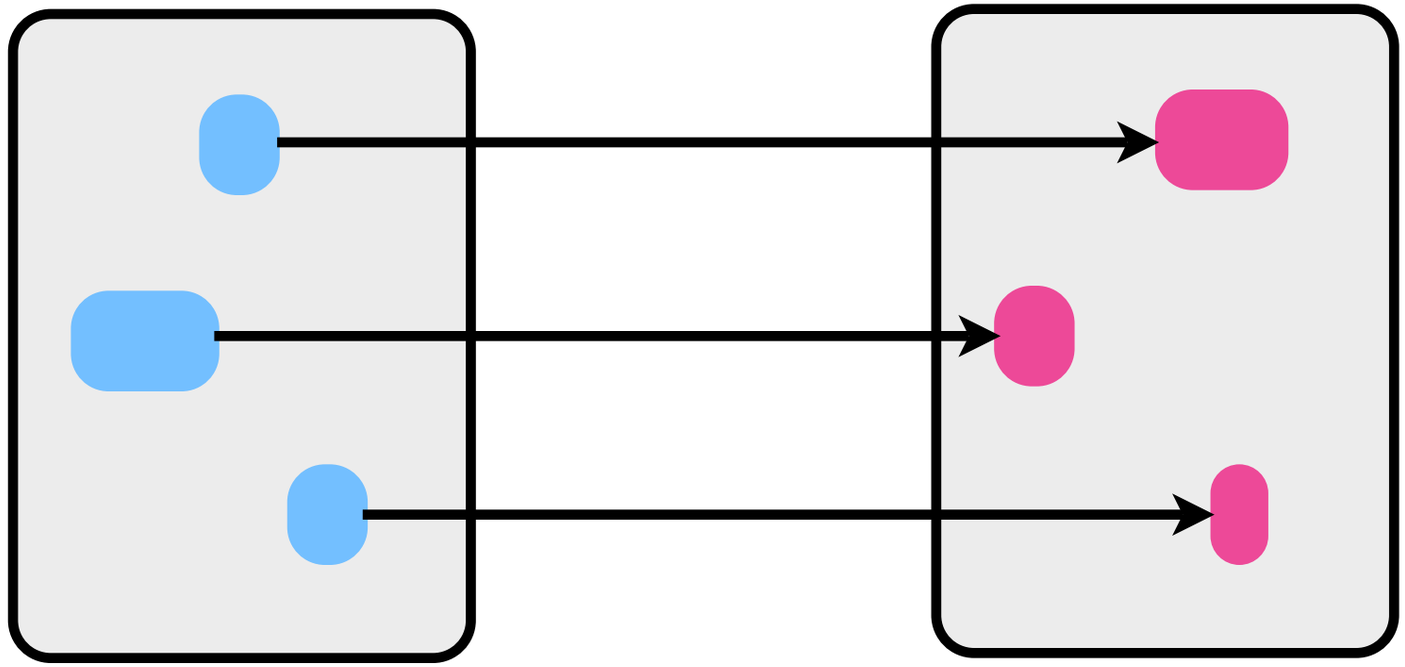
simultaneous editing, linked editing, cren, clonetracker

prior work: clones



do not track relationships
among differences

prior work: explicit mappings



prior work: explicit mappings

class library migration [oopsla'95]

arcum [esec/fse'07]

refactoring playback: catchup! [icse'05],
refactoringcrawler [ecoop'06]

program transformation: txl [comp.lang.'91],
tawk [wpc'96], etc.

prior work: explicit mappings

specific:

succinct, specialized languages

may require extensive domain knowledge

general:

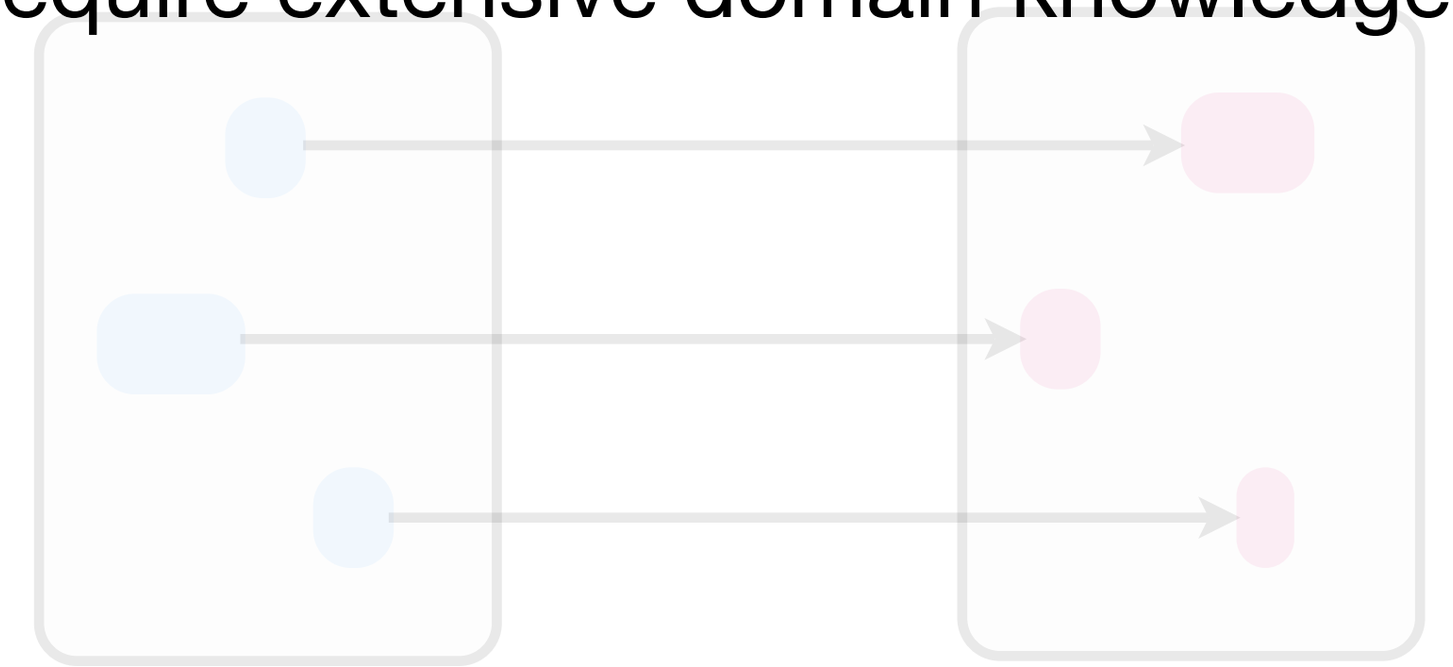
broad, expressive transformation languages

may require knowledge on par with the compiler

specific:

succinct, specialized transformation languages

may require extensive domain knowledge



general:

broad, expressive transformation languages

may require knowledge on par with the compiler

specific:

succinct, specialized transformation languages

may require extensive domain knowledge

twinning:

aimed at end-programmers

restricted to translation of statements

enables a fairly general class of customizations

general:

broad, expressive transformation languages

may require knowledge on par with the compiler

mapping syntax

mapping syntax

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

mapping syntax

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

Elem ==> Element

empEl.getAttribute(“type”) ==> empEl.attributeValue(“type”)

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

mapping syntax

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

Elem ==> Element

empEl.getAttribute(“type”) ==> empEl.attributeValue(“type”)

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

mapping syntax

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

Elem ==> Element

e.getAttribute(attr) ==> e.attributeValue(attr)

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

mapping syntax

```
Elem
```

```
==>
```

```
Element
```

```
Elem e, String attr:
  e.getAttribute(attr)
```

```
==>
```

```
Element e, String attr:
  e.attributeValue(attr)
```

mapping syntax

```
Elem          ==>   Element
```

```
Elem e, String attr:
  e.getAttribute(attr)   ==>   Element e, String attr:
                                 e.attributeValue(attr)
```

mapping syntax

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

Elem e, String attr:
e.getAttribute(attr) **==>** **Element e, String attr:**
e.attributeValue(attr)

mapping syntax

```
String (Elem e, String attr) {
  return e.getAttribute(attr);
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr);
}
```

mapping syntax

```
String (Elem e, String attr) {
  return e.getAttribute(attr);
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr);
}
```

pair of nameless Java methods

can match and replace Java statements

matching & replacement

matching & replacement

```
String (Elem e, String attr) {
  return e.getAttribute(attr)
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr)
}
```

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

matching & replacement

```
String (Elem e, String attr) {
  return e.getAttribute(attr)
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr)
}
```

```
Employee getEmployee(Elem empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

matching & replacement

```
String (Elem e, String attr) {
  return e.getAttribute(attr)
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr)
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

matching & replacement

```
String (Elem e, String attr) {
  return e.getAttribute(attr)
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr)
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

matching & replacement

```
String (Elem e, String attr) {
  return e.getAttribute(attr)
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr)
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

matching & replacement

```
String (Elem e, String attr) {
  return e.getAttribute(attr);
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr);
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.getAttribute("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

matching & replacement

```
String (Elem e, String attr) {
  return e.getAttribute(attr)
}
```

==>

```
String (Element e, String attr) {
  return e.attributeValue(attr)
}
```

```
Employee getEmployee(Element empEl)
{
  String name = getTextValue(empEl,"Name");
  int id = getIntValue(empEl,"Id");
  int age = getIntValue(empEl,"Age");

  String type = empEl.attributeValue("type");

  Employee e = new Employee(name,id,age,type);

  return e;
}
```

benefits of the mapping

crystallizes the relationship between the implementations

enables maintenance of the api correspondence concern separately from the program logic

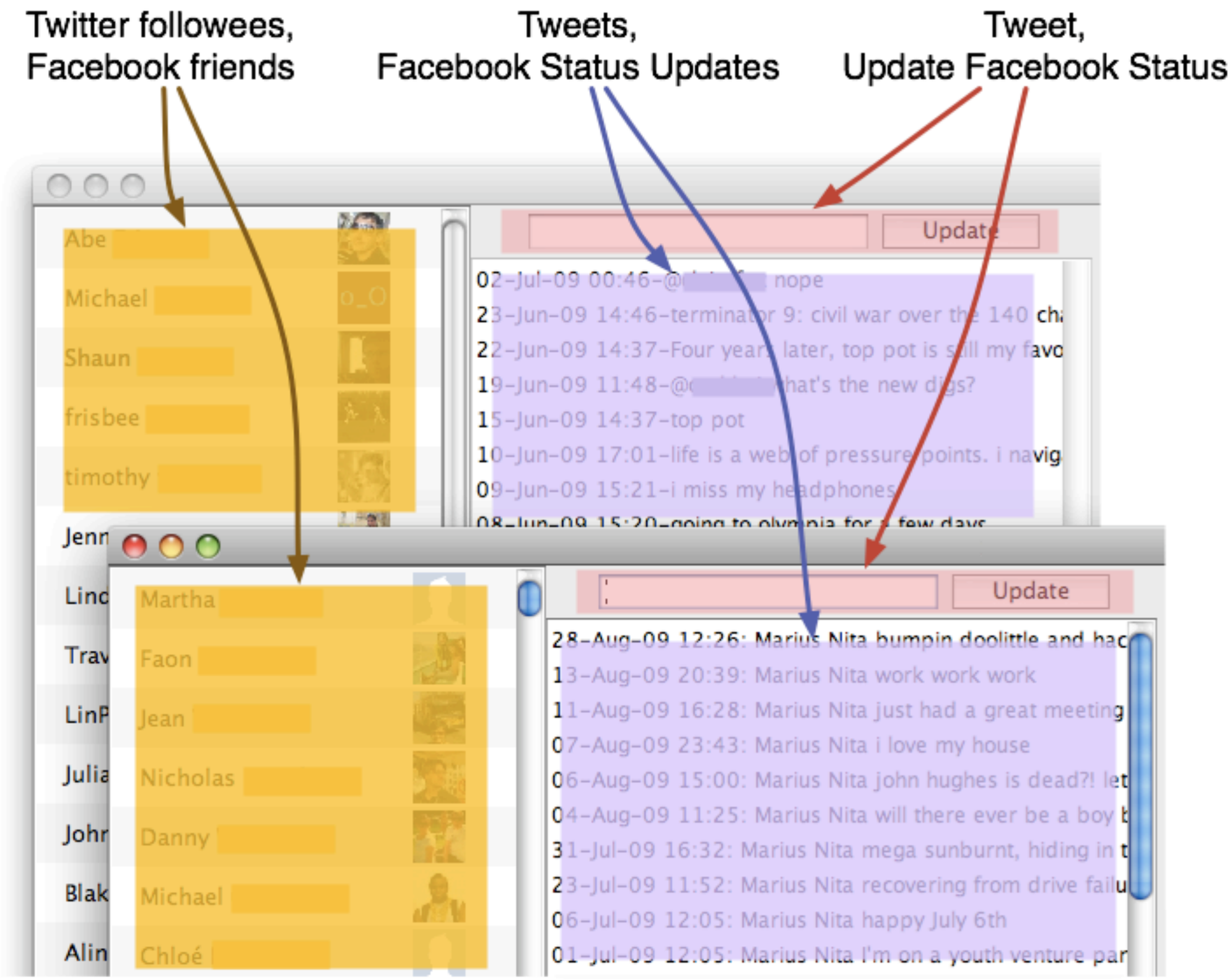
can be applied over time

experience

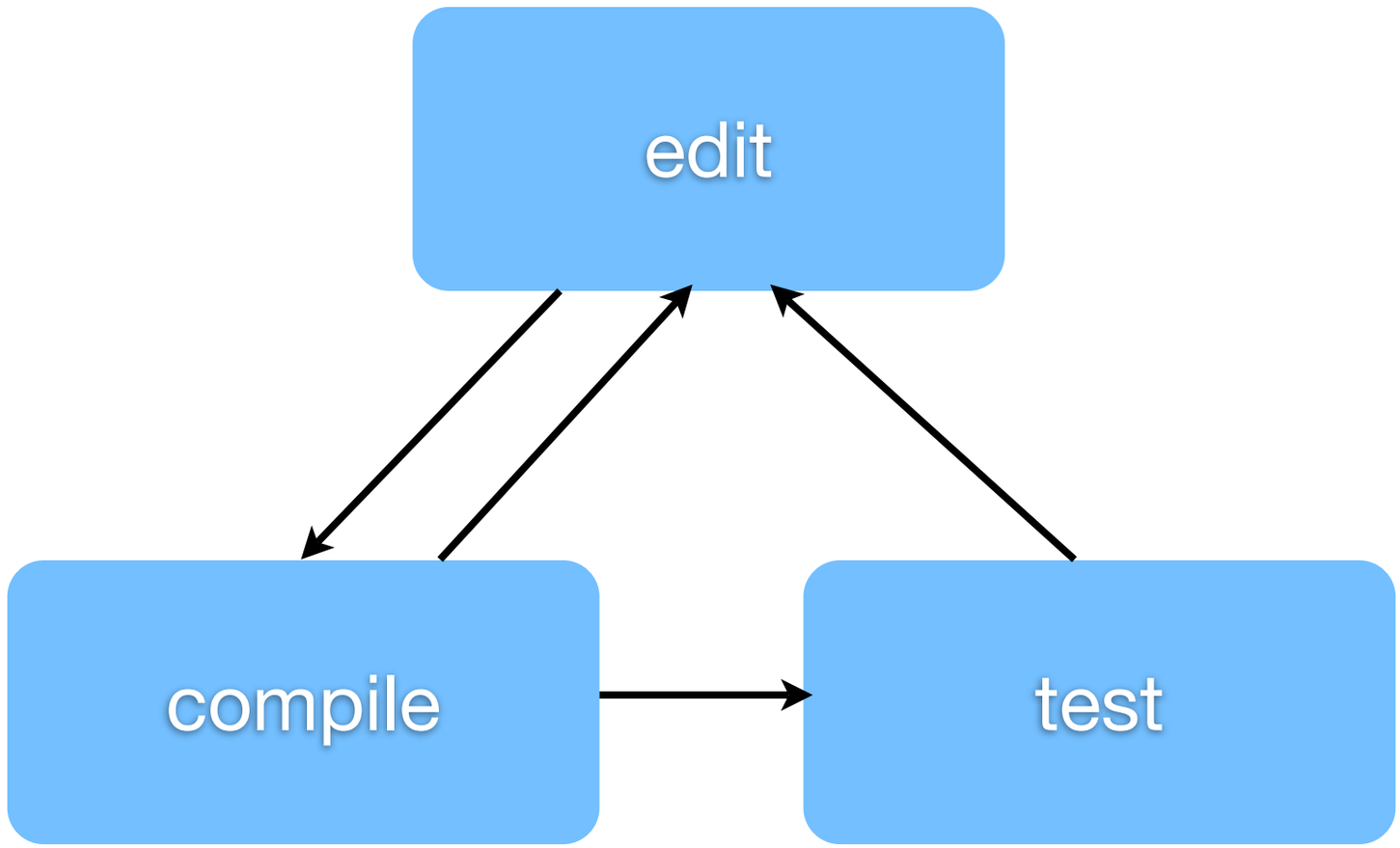
we implemented a prototype and applied it in two separate cases:

xml parsers: crimson vs. dom4j apis
(dom-style subsets)

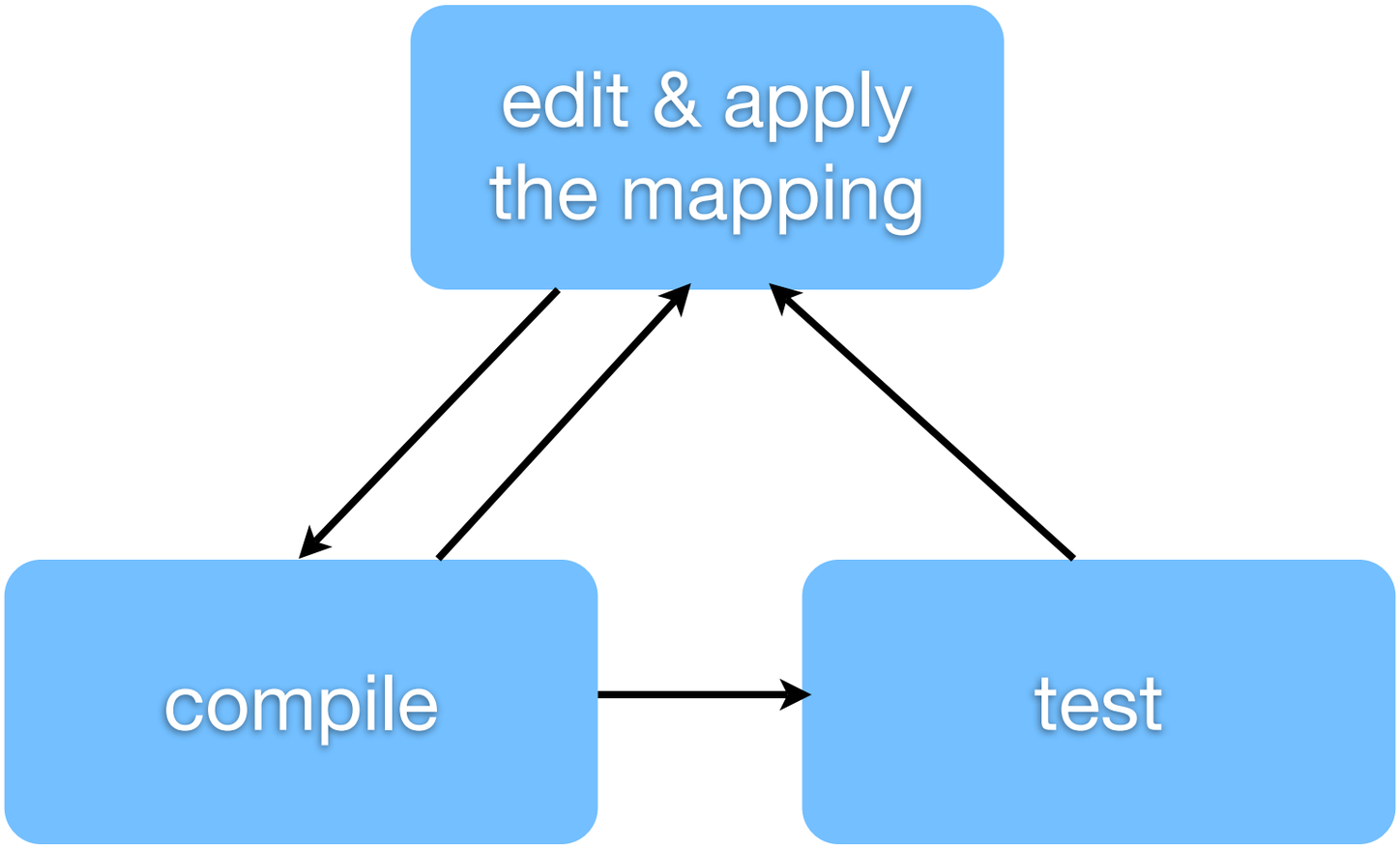
social services: twitter vs. facebook apis


Twitter followees,
Facebook friends
Tweets,
Facebook Status Updates
Tweet,
Update Facebook Status
Abe
Michael
Shaun
frisbee
timothy
Update
02-Jul-09 00:46-@ nope
23-Jun-09 14:46-terminator 9: civil war over the 140 ch
22-Jun-09 14:37-Four years later, top pot is still my favo
19-Jun-09 11:48-@ that's the new digs?
15-Jun-09 14:37-top pot
10-Jun-09 17:01-life is a web of pressure points. i navig
09-Jun-09 15:21-i miss my headphones
Martha
Faon
Jean
Nicholas
Danny
Michael
Chloé
Update
28-Aug-09 12:26: Marius Nita bumpin doolittle and hac
13-Aug-09 20:39: Marius Nita work work work
11-Aug-09 16:28: Marius Nita just had a great meeting
07-Aug-09 23:43: Marius Nita i love my house
06-Aug-09 15:00: Marius Nita john hughes is dead?! let
04-Aug-09 11:25: Marius Nita will there ever be a boy b
31-Jul-09 16:32: Marius Nita mega sunburnt, hiding in t
23-Jul-09 11:52: Marius Nita recovering from drive failu
06-Jul-09 12:05: Marius Nita happy July 6th
01-Jul-09 12:05: Marius Nita I'm on a youth venture par

twinning process



twinning process



most entries are simple

```
String (twitter.Status status) {
   return status.getText();
}
```

==>

```
String (fb.Status status) {
   return status.getMessage();
}
```

code outside the mapping

```
List (twitter.Twitter svc) {
  return svc.getFollowing();
}
```

==>

```
List (fb.FacebookSession svc) {
  return FbUtil.getFriends(svc);
}
```

code outside the mapping

```
List (twitter.Twitter svc) {
  return svc.getFollowing();
}
```

==>

```
List (fb.FacebookSession svc) {
  return FbUtil.getFriends(svc);
}
```

conversions

```
List (twitter.Twitter svc) {
  return svc.getUserTimeline();
}
```

==>

```
List (fb.FacebookSession svc) {
  return Arrays.asList(svc.getStatuses());
}
```

```
twitter.Twitter () {
  LoginDialog login = new LoginDialog();
  String u = login.getUserName();
  String p = login.getPassword();
  return new twitter.Twitter(u, p);
}
```

==>

```
fb.FacebookSession () {
  String API_KEY = "...";
  String SECRET = "...";
  fb.DesktopApplication app =
    new fb.DesktopApplication(API_KEY, SECRET);
  String token = app.requestToken();
  String url = app.getLoginUrl();
  // launch url in browser
  return app.requestSession(token);
}
```

```
twitter.Twitter () {
  LoginDialog login = new LoginDialog();
  String u = login.getUserName();
  String p = login.getPassword();
  return new twitter.Twitter(u, p);
}
```

==>

```
fb.FacebookSession () {
  String API_KEY = "...";
  String SECRET = "...";
  fb.DesktopApplication app =
    new fb.DesktopApplication(API_KEY, SECRET);
  String token = app.requestToken();
  String url = app.getLoginUrl();
  // launch url in browser
  return app.requestSession(token);
}
```

current limitations

no support for subtyping yet

no contextual matching

types correspond one to one

all the limitations of source-to-source transforms

more in the paper

generating abstract interfaces and adapters from the mapping

dealing with exceptions

recap

crystallizes the relationship between the implementations

enables maintenance of the api correspondence concern separately from the program logic

can be applied over time

http://cs.washington.edu/homes/marius