

# Myx Tutorial



Hazel Asuncion

For Inf 221

February 24, 2009

Slides taken and adapted from Eric Dashofy's  
C2-style & Implementing your system in Myx

# Implementing Lunar Lander



- Design, Myx Style
- Implement Myx Components
- Assign Myx Components to Implementations

# Designing, Myx Style



- Architectural style – set of constraints put on development to elicit beneficial properties
- Myx style
  - Constraints:
    - Components – loci of computation; provide services to other connectors; transform data
    - Connectors – loci of communication; move data between bricks

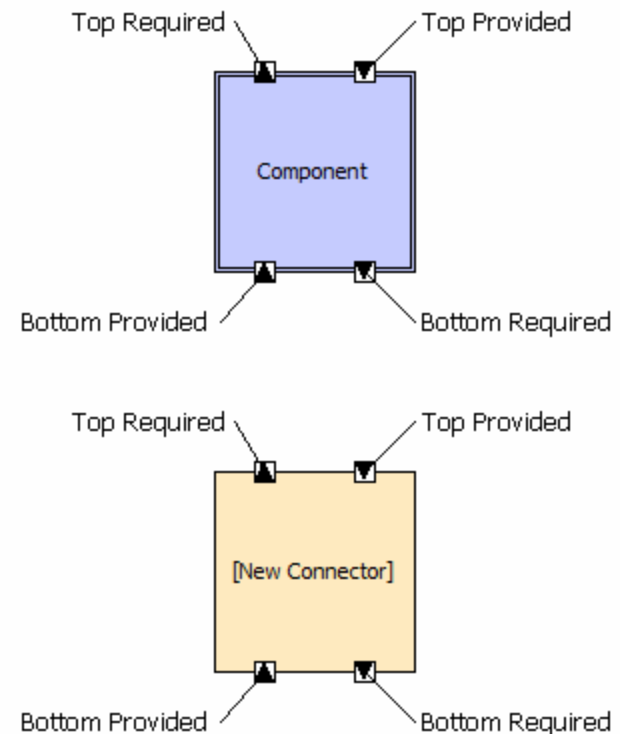
***Components & connectors are collectively called “bricks”***

# Designing, Myx Style

## ■ Myx style

### – Constraints:

- Interfaces – “portals” through which bricks interact
  - Service Types:
    - » Provided (Provided by the component)
    - » Required (Required by the component)
  - Direction: in, out, or inout (flow of control)

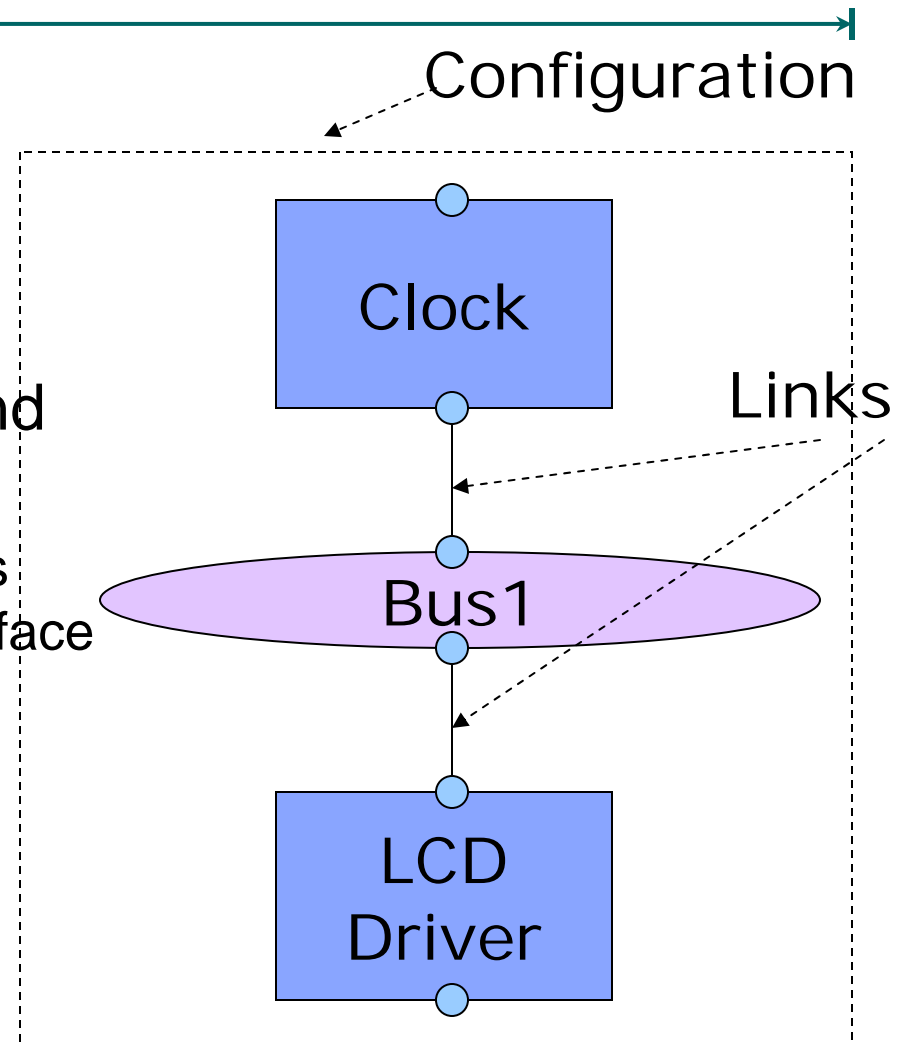


# Designing, Myx Style

## ■ Myx style

### – Constraints:

- Links – associations between provided and required interfaces
  - Direction of control is provided by the interface endpoints

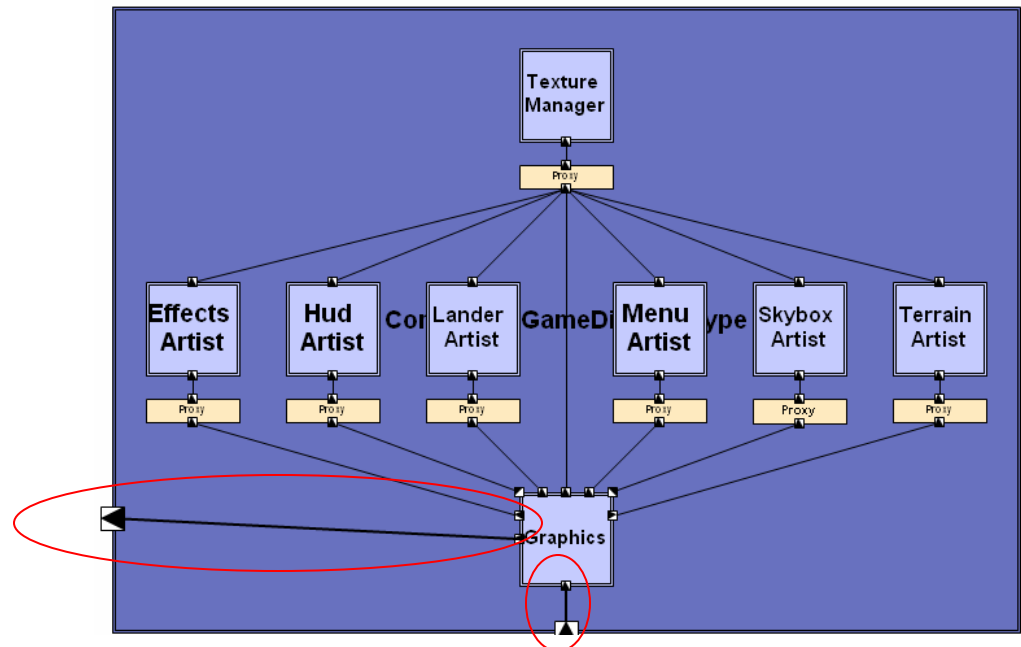


# Designing, Myx Style

## ■ Myx style

### – Constraints:

- Interface mappings – similar to links but used to associate an interface of the outer brick with the interface of the inner brick



# Designing, Myx Style



## ■ Constraints

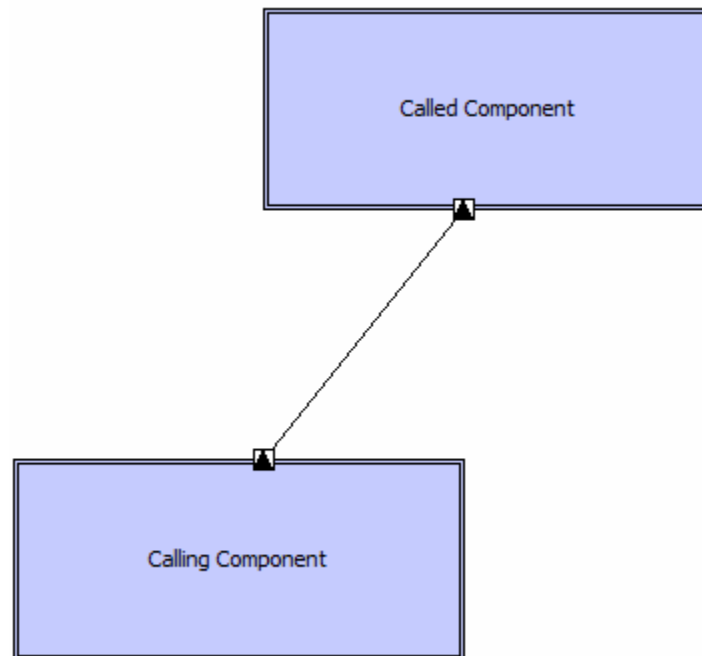
- Links have exactly two endpoints
  - One required interface to one provided interface
- Substrate independence
  - A brick may make assumptions about services provided by other bricks *above* it, but not services provided by other bricks *below* it
- Communication only through interfaces

# Designing, Myx Style



## ■ Communication

- Synchronous invocation – permitted upward only
  - Achieved through function call





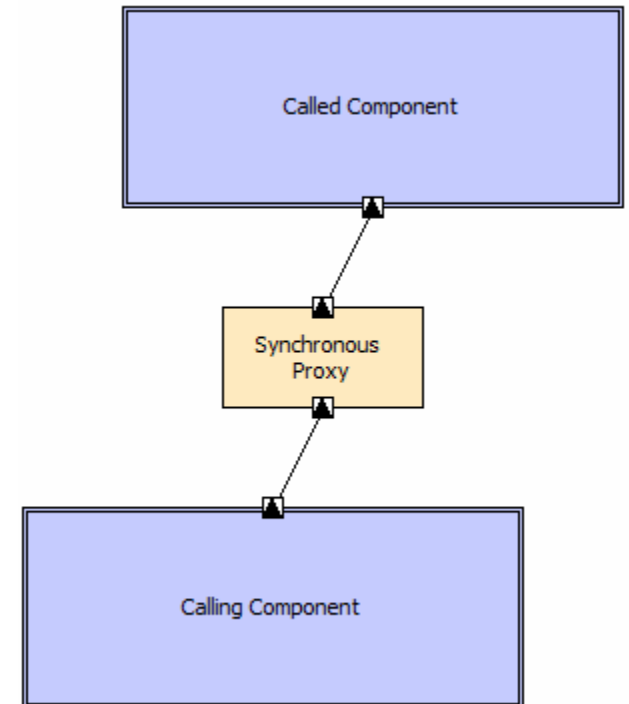
# Designing, Myx Style

## ■ Communication

– Synchronous invocation – with proxy

- Proxy enables

- Dynamic linking
- Data format transformation
- Logging
- Debugging

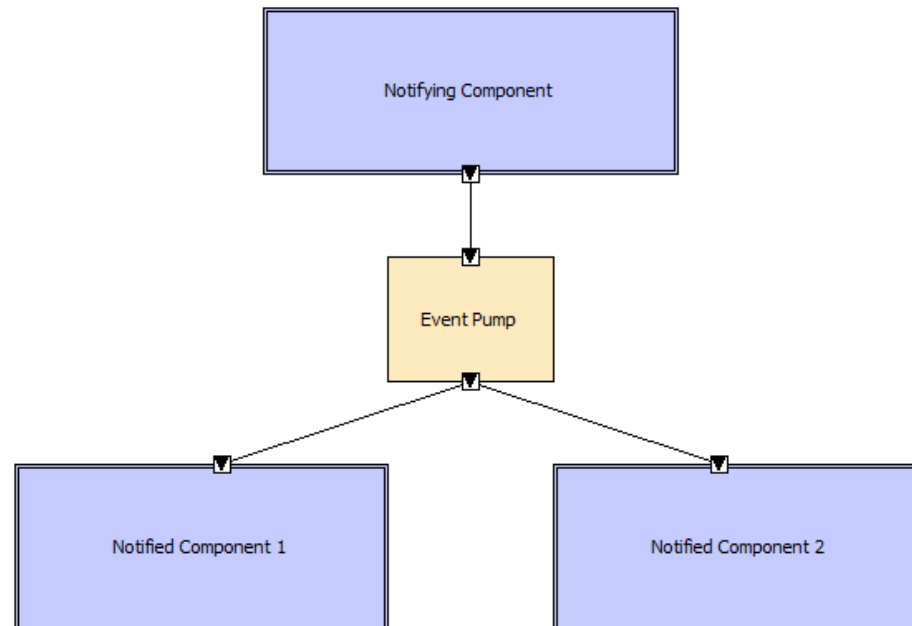


# Designing, Myx Style

## ■ Communication

### – Asynchronous notification

- Event pump – receives the message and forwards the message to all the bricks connected to its interface



# Designing, Myx Style

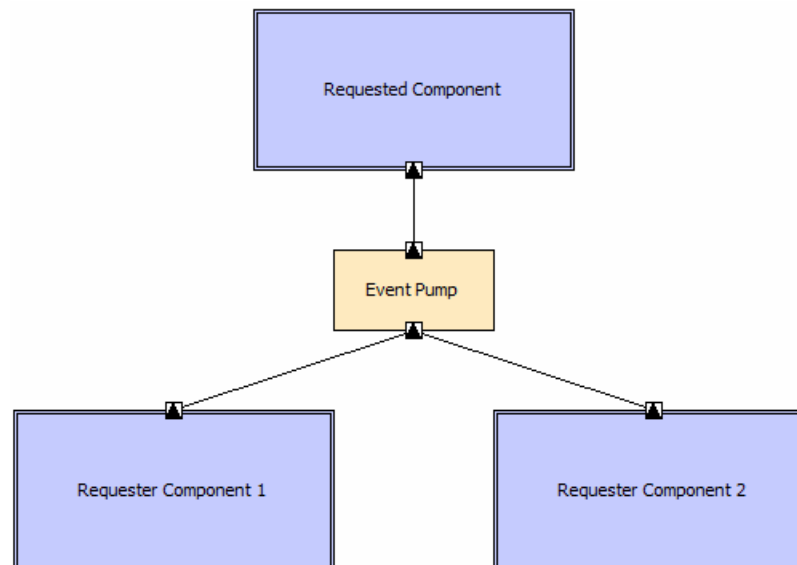
## ■ Communication

### – Asynchronous requests

– Event pump – similar to synchronous notification

» Adds the requests to the queue

» Flow of control immediately given back to the requester

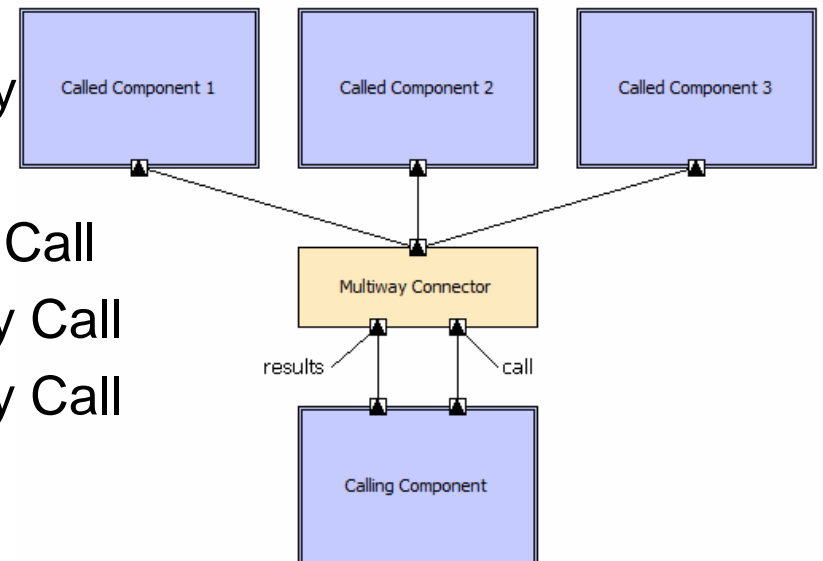


# Designing, Myx Style


## ■ Communication

### – Multiway Call Pattern


- Multiway Connector – more advanced
  - Use when the calling component want to invoke the same operation on all three components simultaneously
  - May be implemented
    - » Synchronous Multiway Call
    - » Asynchronous Multiway Call
    - » Asynchronous Multiway Call with Asynchronous Notification



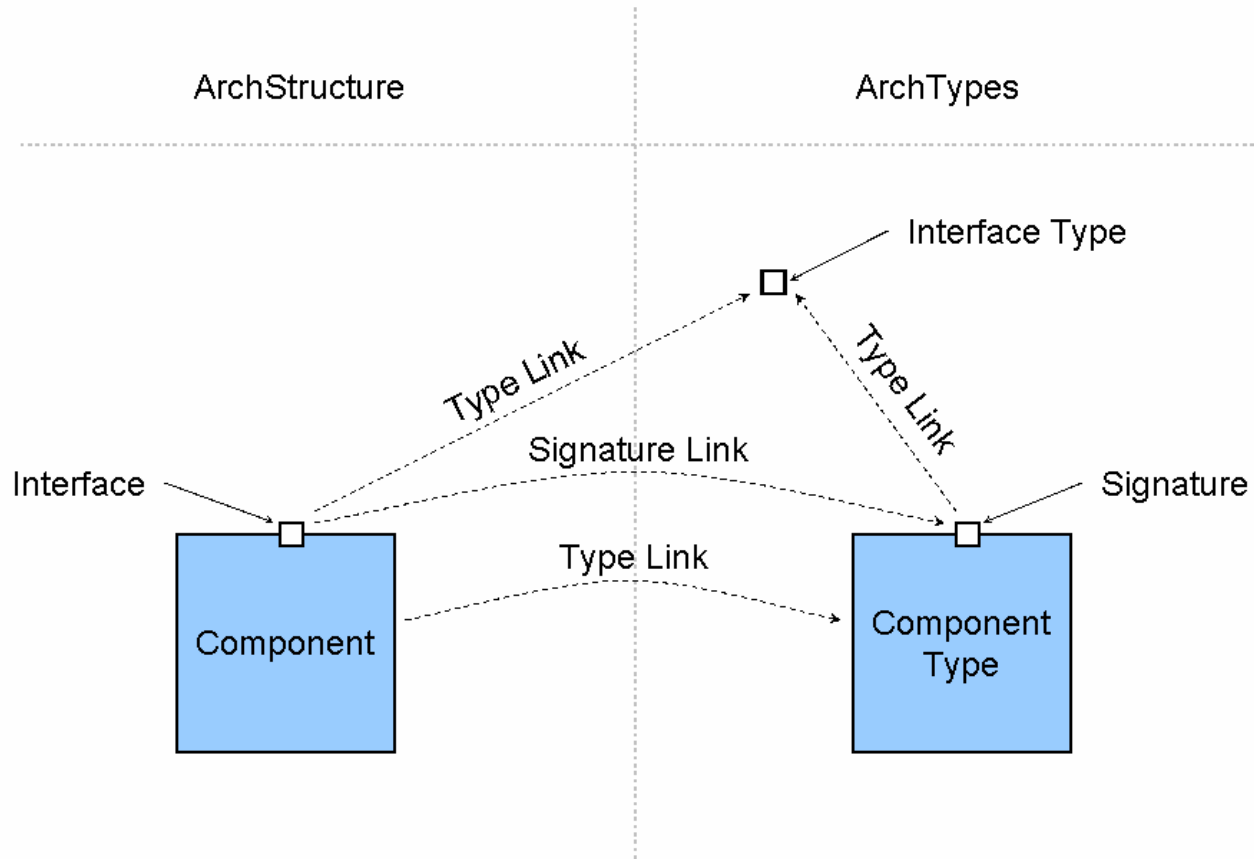
# Myx Style - Benefits

- 
- Performance – no runtime overhead, if use the synchronous procedure calls
  - Reusability – component/connector can be reused
  - Flexibility – substrate independence – can replace a layer of components/connectors
  - Concurrency – asynchronous communication
  - Reduced deadlock – bricks are in separate memory locations
  - Distributability – easy to split an application to run on different processes or machines

# Designing...in a Nutshell

- 
1. Create a new component
  2. Add component interfaces
  3. Create new component types (usually one for each new component added)
  4. Assign components to component types
  5. Add signatures to component types
  6. Add links
  7. Create interface types
  8. Assign interfaces and signatures to interface types (Type Wranger)

# Using xADL



Taken from Scott Hendrickson's tutorial

# Implementation



- You have designed an architecture for a Lunar Lander application
  - This could be termed the *prescriptive* architecture—it is what you *intend* to construct
- Now, you have to go implement the system
  - This will test the assumptions you made during design
  - You will be challenged to do so in a way that plausibly connects your architecture to your implementation
  - Your architecture may end up slightly different—this is a common phenomenon where *descriptive architecture*  $\neq$  *prescriptive architecture*



# Implementation Challenges



- Although your architectures are going to be different, they have some commonalities
  - Components
  - Interconnections (links, possibly explicit connectors)
  - Explicit provided and required interfaces
- Your target platform (Java) is mismatched
  - Objects
  - Pointers/references everywhere
  - Optional provided interfaces
- How do we bridge the gap?

# An Architecture Framework

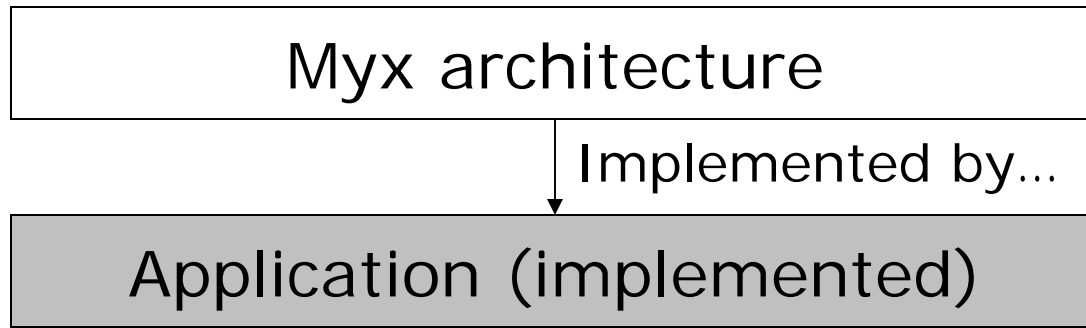


- An architecture framework is software that bridges the gap between the concepts of an architectural style and the capabilities of a given platform (programming language + operating system + runtime)

# Implementing a Myx architecture



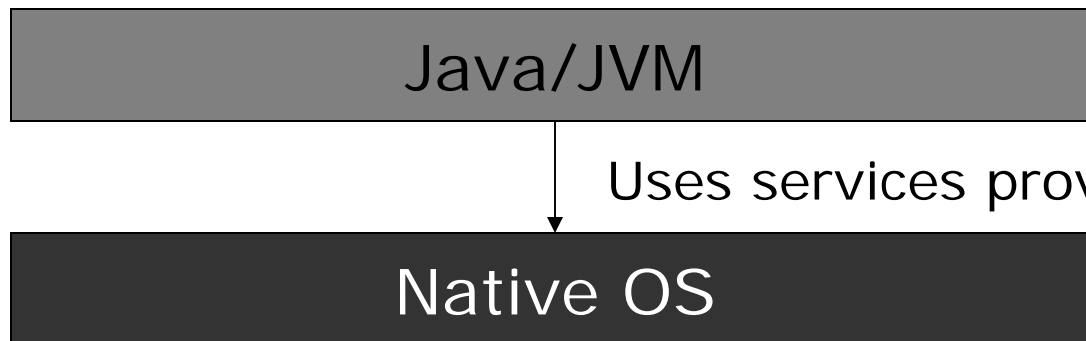
Myx  
World



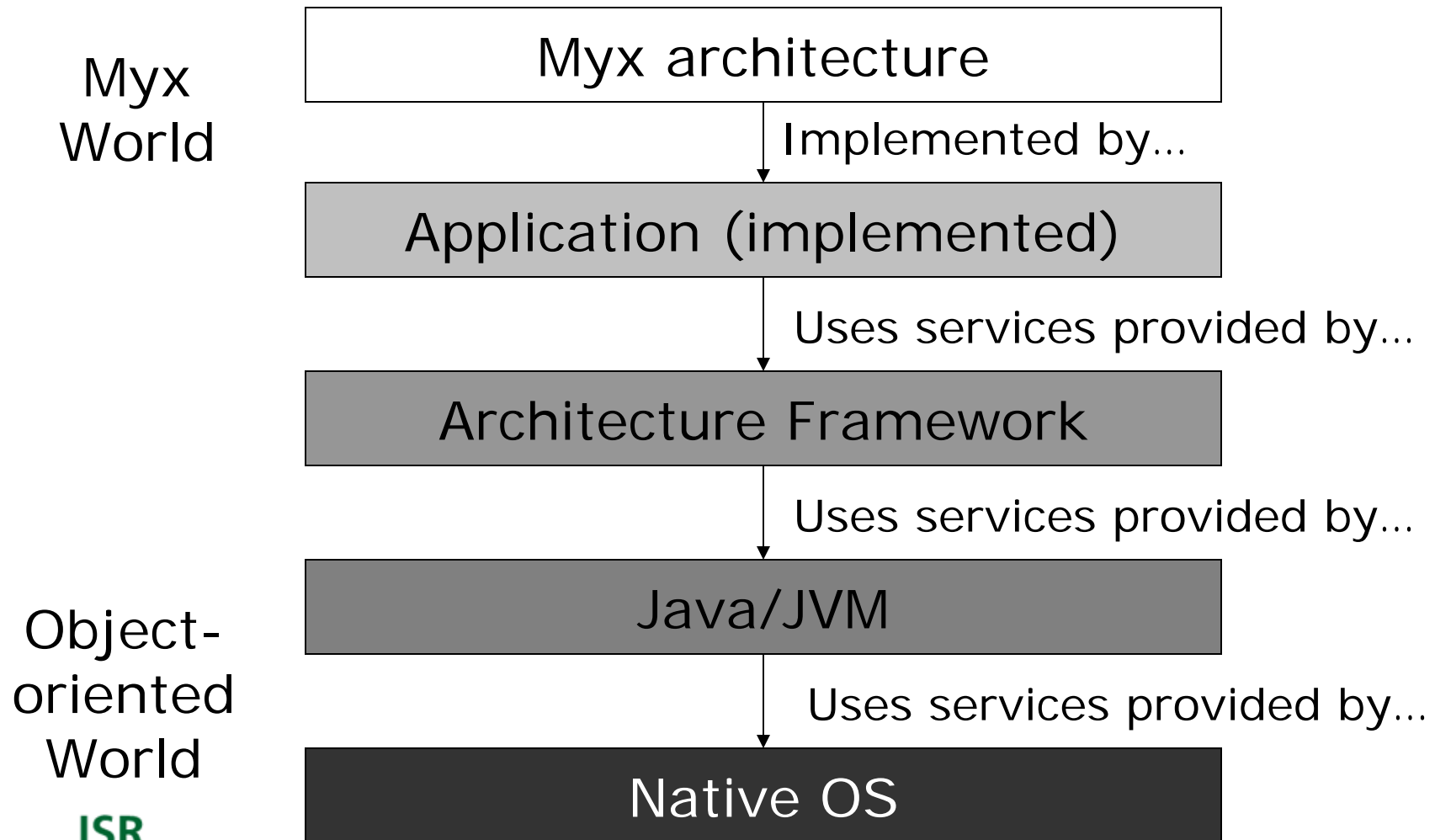
???

How do we bridge this gap?

Object-  
oriented  
World



# Implementing a Myx architecture



# Example



- Myx World

- Components
- Connectors
- Events
- Links
- Many threads
- Synchronous and Asynchronous communication

- Java World


- Objects
- Method calls
- Parameters
- References
- Few threads
- Synchronous communication

# Enter myx.fw



- myx.fw is an architecture framework for the Myx style built in Java, providing:
  - Abstract base classes for components, connectors
  - Reusable connectors (local & network)
  - (Pluggable) topology management
  - (Pluggable) threading policies
- Essentially, myx.fw does the hard work, components and connectors simply implement behaviors.

# Your basic myx.fw component...



```
package edu.uci.ics.mypackage;

//standard imports
import edu.uci.isr.myx.fw.AbstractMyxSimpleBrick;
import edu.uci.isr.myx.fw.IMyxName;
import edu.uci.isr.myx.fw.MyxUtils;

//May not need this import
//This can be used in looking up a specific component
import edu.uci.isr.myx.fw.MyxRegistry;
```

# Your basic myx.fw component...

```
public class MyMyxComponent extends AbstractMyxSimpleBrick
{
}

```

Implements lots of boilerplate  
functionality from the IMyxBrick,  
IMyxLifecycleProcessor,  
IMyxProvidedServiceProvider



# Your basic myx.fw component...

```
public class MyMyxComponent extends AbstractMyxSimpleBrick
{
    private IView viewer;

    //declare interfaces
    public static final IMyxName INTERFACE_NAME_OUT =
        MyxUtils.createName("out_interface");
}
```

Name must exactly match the  
interface name on the component

# ← Lifecycle Methods →

```
public void init(){
    //called when component/connector is created
    //but component not guaranteed to be hooked up
}

public void begin(){
    //called when component/connector is hooked up
    //in the architecture, should send initial messages
}

public void end(){
    //called when component/connector is about to be
    //unhooked, should send final messages
}

public void destroy(){
    //called when component/connector is about to be
    //destroyed
}
```

# A boilerplate myx.fw component...

```
public class MyMyxComponent extends AbstractMyxSimpleBrick
{
    private IView viewer;

    //declare interfaces
    public static final IMyxName INTERFACE_NAME_OUT =
        MyxUtils.createName("out_interface");

    public static final IMyxName INTERFACE_NAME_IN =
        MyxUtils.createName("in_interface");

    public void begin(){ //called automatically by fw.
        //send out initial events
        //may get required objects here

        viewer =
            (IView).MyxUtils.getFirstRequiredServiceObject(this,
                INTERFACE_NAME_OUT)
    }
}
```

# A boilerplate myx.fw component...

```
public class MyMyxComponent extends AbstractMyxSimpleBrick
{
    :
    public static final IMyxName INTERFACE_NAME_IN =
        MyxUtils.createName("in_interface");
    :
    public Object getServiceObject(IMyxName name) {
        //if no interfaces are going in, always return null
        //In this case, we have an interface coming in
        if(name.equals(INTERFACE_NAME_IN)) {
            return this;
        }
        return null;
    }
}
```

# ArchStudio Tour



# Further references



- Myx Architecture Style
  - <http://www.isr.uci.edu/projects/archstudio/myx.html>
- Intro to ArchStudio
  - [http://tps.ics.uci.edu/svn/projects/archstudio4/documents/INF123\\_IntroToArchStudio.pdf](http://tps.ics.uci.edu/svn/projects/archstudio4/documents/INF123_IntroToArchStudio.pdf)
- Hello World Tutorials
  - <http://www.ics.uci.edu/~hasuncio/classes/in4matx119/HelloWorldTutorial.pdf>
  - [http://tps.ics.uci.edu/svn/projects/archstudio4/documents/INF123\\_HelloWorld\\_OneComp\\_Mac.pdf](http://tps.ics.uci.edu/svn/projects/archstudio4/documents/INF123_HelloWorld_OneComp_Mac.pdf)
  - [http://tps.ics.uci.edu/svn/projects/archstudio4/documents/HelloWorld\\_TwoComp.pdf](http://tps.ics.uci.edu/svn/projects/archstudio4/documents/HelloWorld_TwoComp.pdf)
- Environment setup for current Lunar Lander implementation:
  - [http://tps.ics.uci.edu/svn/projects/archstudio4/documents/INF123\\_LL\\_EnvSetup.pdf](http://tps.ics.uci.edu/svn/projects/archstudio4/documents/INF123_LL_EnvSetup.pdf)
  - [http://tps.ics.uci.edu/svn/projects/archstudio4/documents/INF123\\_LL\\_EnvSetup2.pdf](http://tps.ics.uci.edu/svn/projects/archstudio4/documents/INF123_LL_EnvSetup2.pdf)