Building and Extending IDEs

Vaibhav
Aftab

IDEs

An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development.

The 3 Pillars

1

Structured Representation of Programs

Programs are represented as objects rather than text.

<u>This is key in enabling the environment to support multiple languages.</u>

2

3

IDE continued…

An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs offer Intelligent code completion features.

Expectations from an IDE

- extensible, incrementally improvable, flexible, fast, and efficient.
- Its components must be interoperable.
- It should be easy to use.

Expectations continued…

- It should be able to support effective product and process visibility.
- It should be able to support effective management control.
- It should be proactive.
- It should be able to support multiple users.

Challenges: Expectations

The expectations mentioned earlier are not orthogonal and often conflict

Challenges: Expectations

The expectations mentioned earlier are not orthogonal and often conflict

How?

Challenges: Expectations

Trade offs

- High performance sometimes leads to tighter coupling
- Too many features can lead to higher cognitive burden
- Supporting management aspects can lead to more context switches.

Challenges: Building an IDE

- complexity : adding more features leads to complex system
- reuse: do not code that is available.
- adaptive components: what capabilities are available on the system and adapt the functionality they can provide.
- coupling: do we want different components to be tightly coupled to each other?
- version hell: A-> B.1 , C ->B.2
- lazy loading: Do stuff when asked to do.

RPDE –

Rapid Prototyping for Developing Environments (1990)

A Direct Manipulation Environment that supports

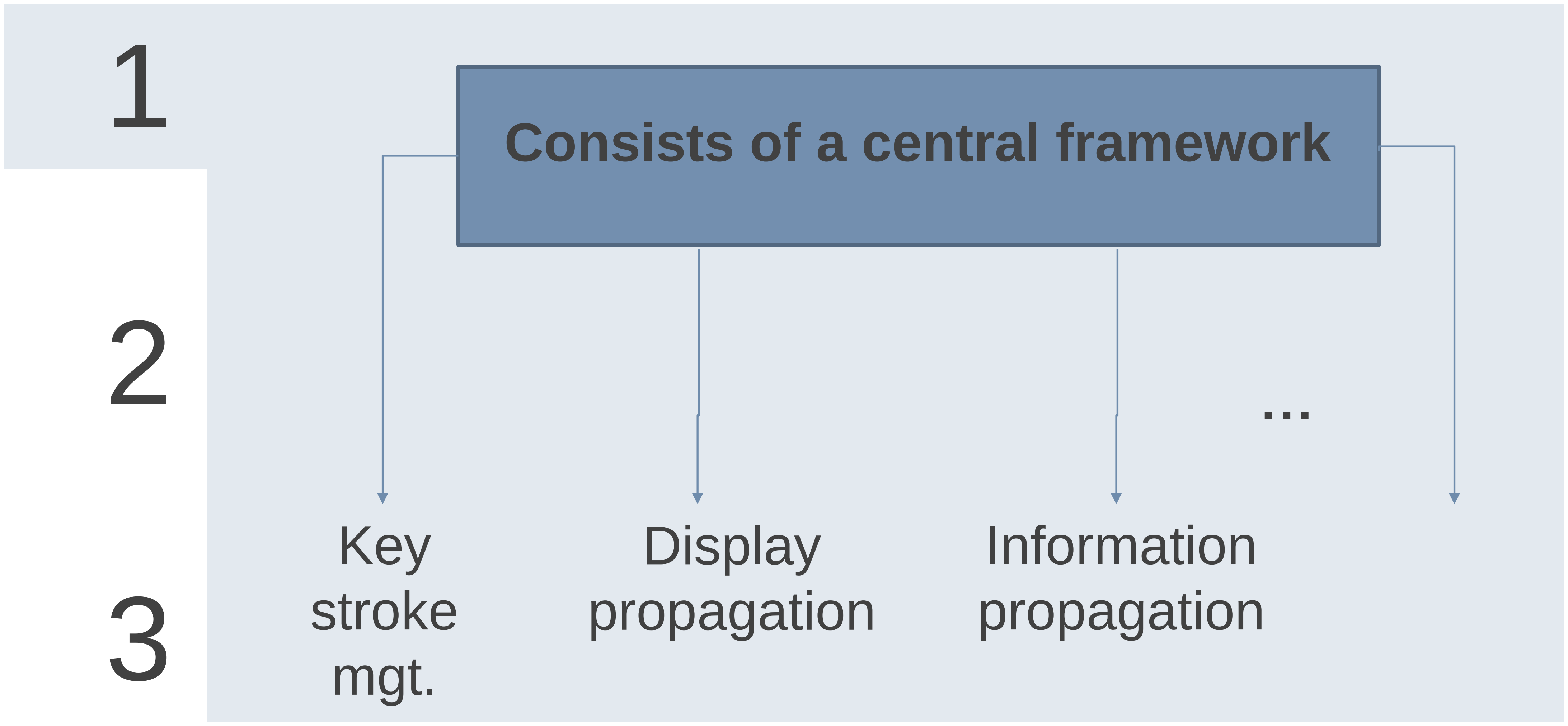
- **Extending an environment**
- **Creation of environments**

Adding a new functionality

Extending a functionality

RPDE

The 3 Pillars



RPDE

The 3 Pillars

1

2

3

These services can be directly used and their implementation details are hidden.

This reduces work in building a new environment or extending the environment.

RPDE

The 3 Pillars

1

2

3

Built on an Object Oriented Programming Paradigm

Allows you to change the environment by the addition of small fragments of code.

Inheritance of classes is also supported.

Enhances code reusability.

RPDE

The 3 Pillars

1

2

3

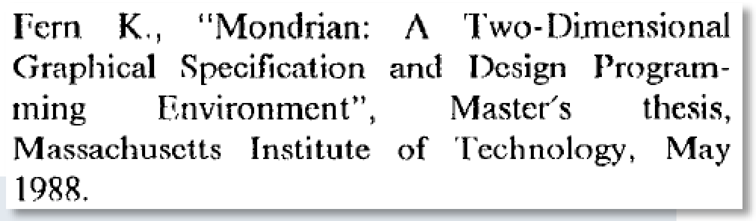
Structured Representation of Programs

Programs are represented as objects rather than text.

<u>This is key in enabling the environment to support multiple languages.</u>

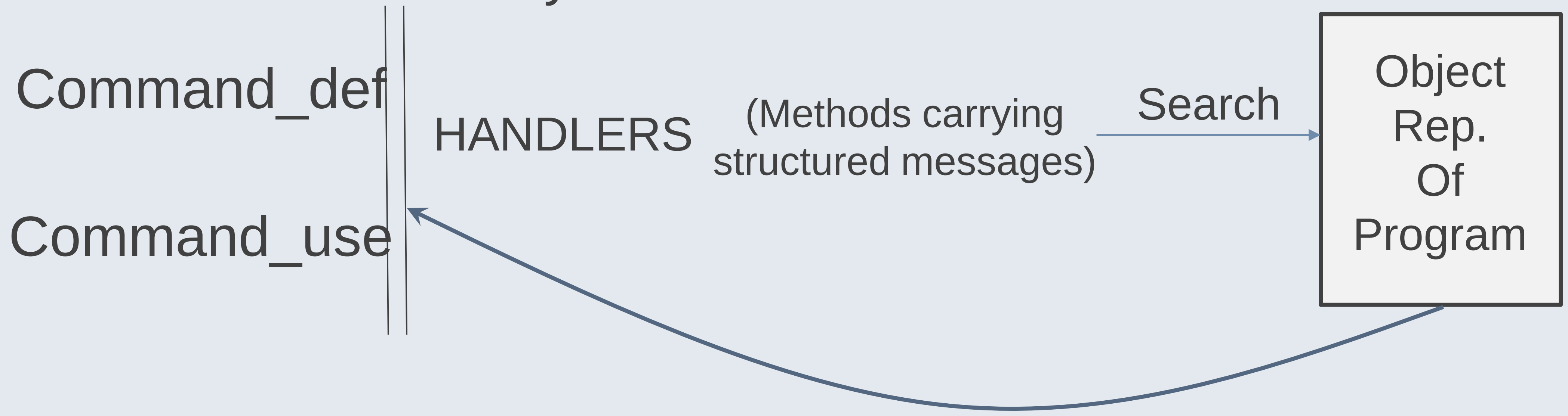
RPDE

Making Extensions in RPDE


Fern K., "Mondrian: A Two-Dimensional Graphical Specification and Design Programming Environment", Master's thesis, Massachusetts Institute of Technology, May 1988.

Adding the "def-use" functionality in a Pascal Environment

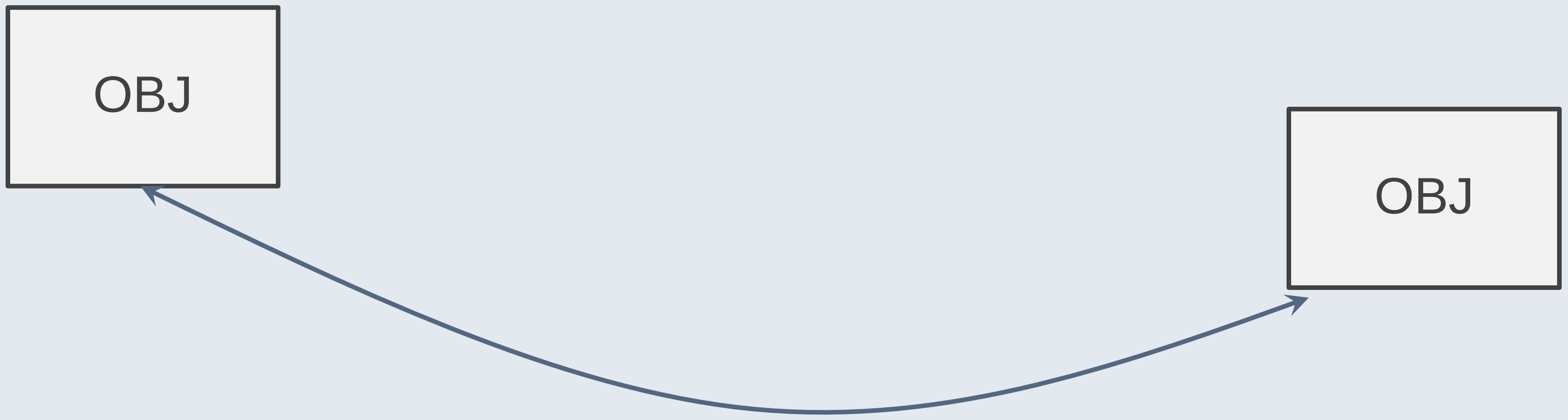
Utility: to know the definition and the next use of a symbol.



RPDE

Making Extensions in RPDE

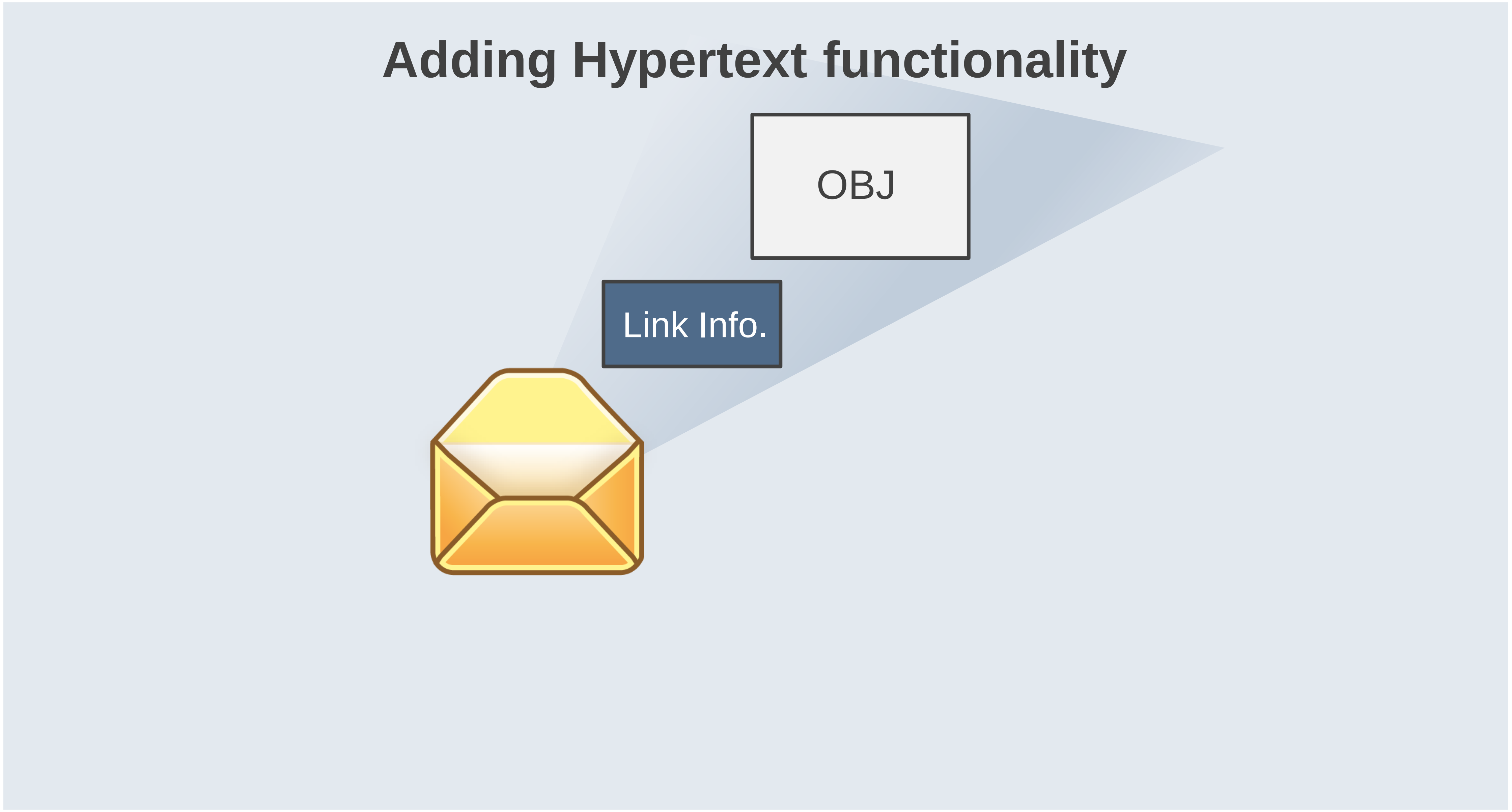
Adding Hypertext functionality



Without making detailed modifications to the objects.
Without affecting the functionality of the environments.

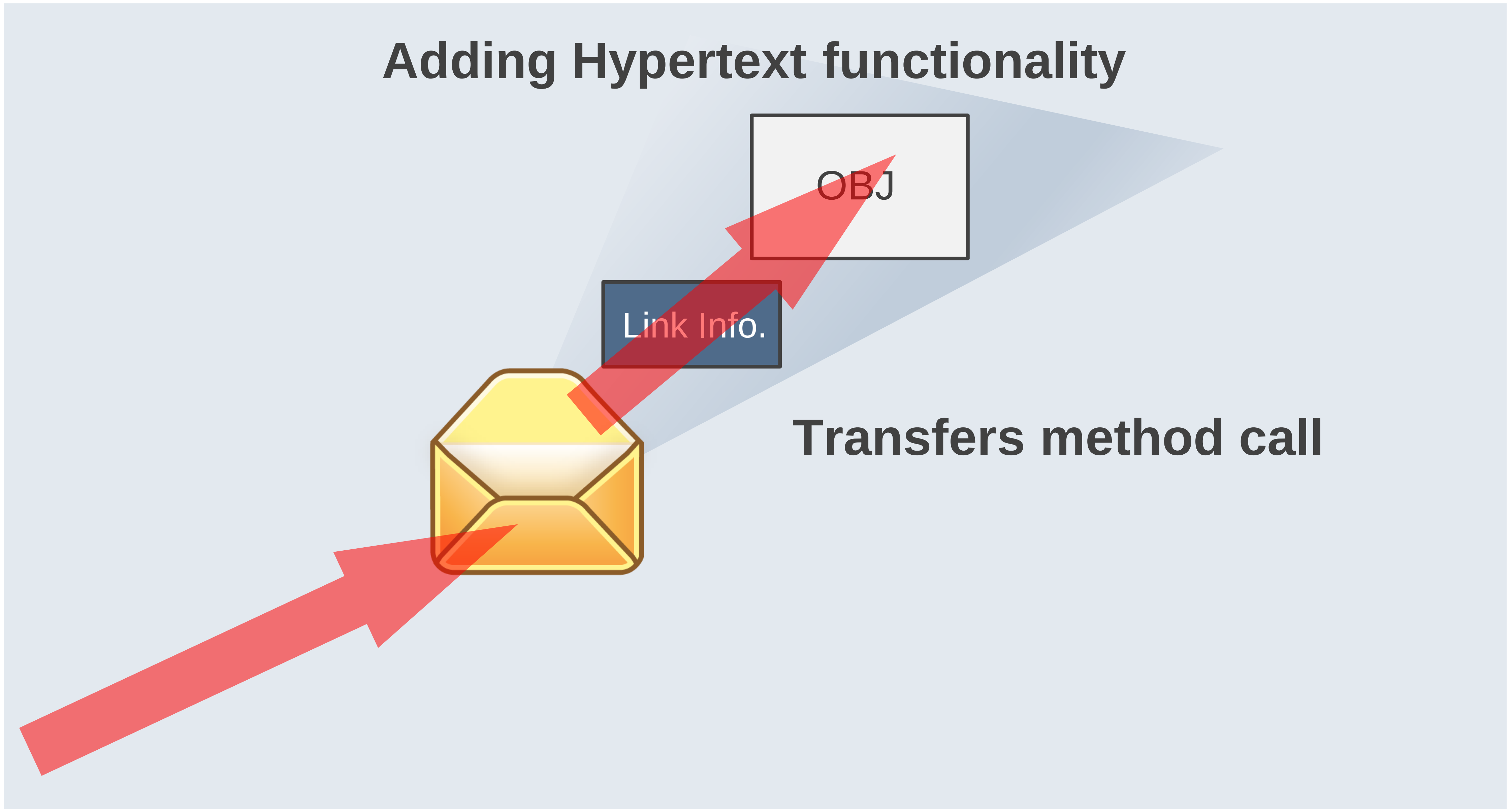
RPDE

Making Extensions in RPDE



RPDE

Making Extensions in RPDE



RPDE

Making Extensions in RPDE

Adapting a Pascal Environment for Programming in C

Since Pascal and C have the same language constructs the same object types were used.

Generate C source codes.

Exceptions: the **switch** in C and **case** in Pascal are different.

A new object type for C's switch was declared.

Environment Generators

- Enable the generation of new environments by modifying and recompiling the environment definition.
- The environment definition constitutes the code.
- E.g. Cornell Synthesizer ’84
  Gandalf ’86
- The approach is declarative and seems appealing, **but it has drawbacks.**

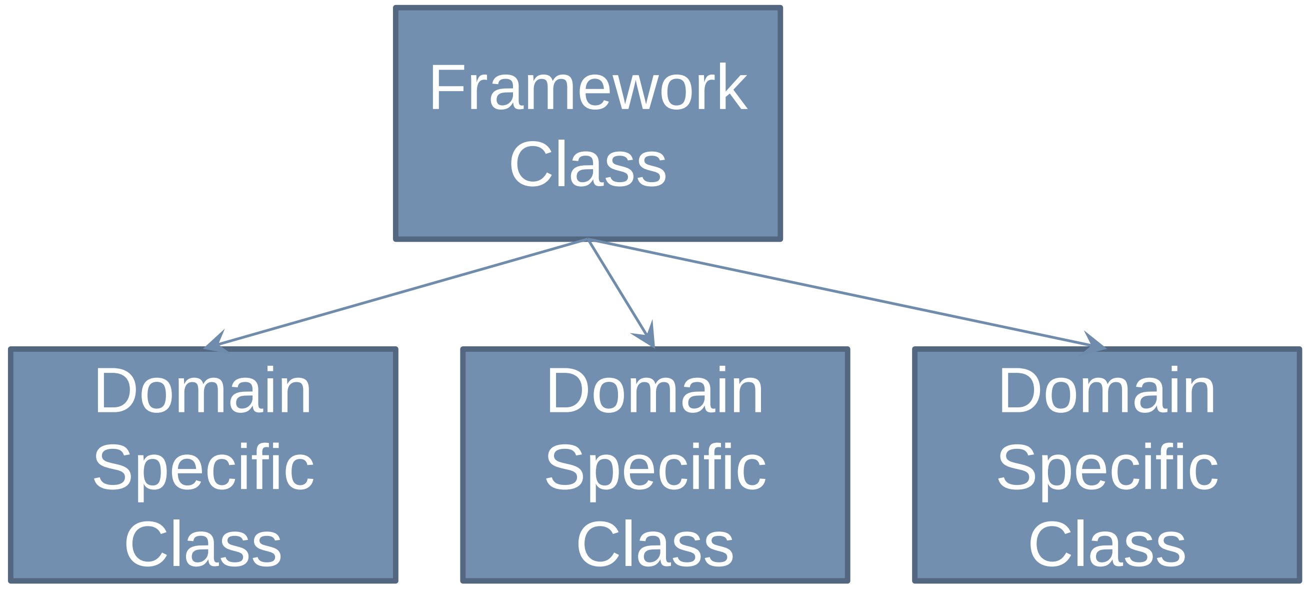
Environment Generators

- Environment Code is **HUGE**, **COMPLEX** AND **INTERRELATED**. Making a change requires complete understanding of this code.
- Changing a functionality will require you to find all places where the functionality was used, in order to ensure your change is deployed in a consistent manner.
- For e.g. you want to remove Select All functionality from Edit Menu. You will want this change to reflect in all views of your environment.

OO Application Frameworks

Provide Environment Extension facilities by allowing the developer to extend subclasses from high-level classes.

E.g. Small Model-View-Controller 1983.

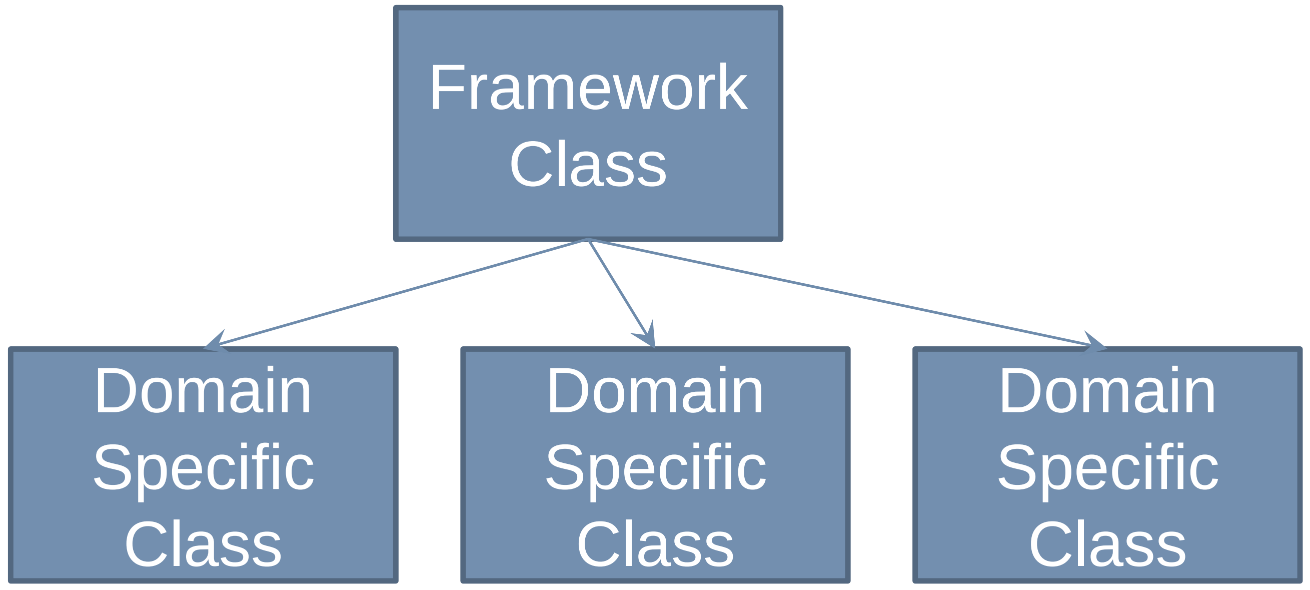


- The framework specifies extendable classes: a framework class and several domain specific subclasses.
- The domain specific classes override methods of the framework class.
- Environments can be generated by extending these classes.

OO Application Frameworks

Provide Environment Extension facilities by allowing the developer to extend subclasses from high-level classes.

E.g. Small Model-View-Controller 1983.



- Although this enhances flexibility, the separation between the framework class and the DS-Classes **are not clear**
- The 2 functionalities are mixed up and you need to be cautious when inheriting, otherwise your environment will lose uniformity.
- E.g. Redo After Undo functionality across different domains

RPDE beats OOAFs and EGs

Framework specifies more granular classes for inheritance.

RPDE

Extend a set of central services – extensions via addition of small code fragments

v/s

Framework only specifies Higher level classes for inheritance.

OOAF

EG

Modify Complex Environment code to make extensions.

Modular Approach: OSGI

The OSGi technology is a set of specifications that define a dynamic component system for Java. These specifications enable a development model where applications are (dynamically) composed of many different (reusable) components.

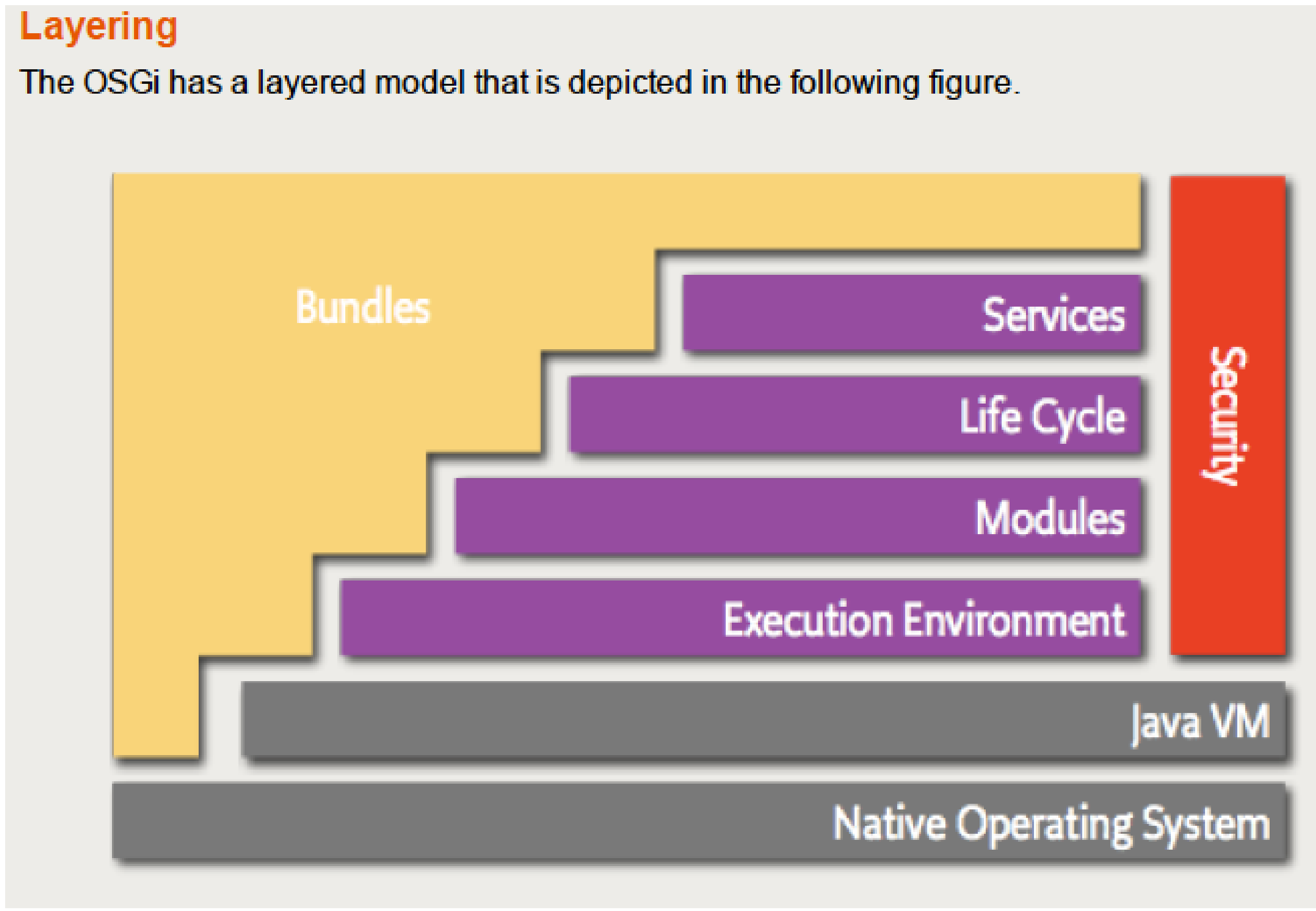
Modular Approach: OSGI

The OSGi specifications enable components to hide their implementations from other components while communicating through services, which are objects that are specifically shared between components

Layered Model: OSGI

The following list contains a short definition of the terms:

- Bundles - Bundles are the OSGi components made by the developers.
- Services - The services layer connects bundles in a dynamic way by offering a publish-find-bind model for plain old Java objects.
- Life-Cycle - The API to install, start, stop, update, and uninstall bundles.
- Modules - The layer that defines how a bundle can import and export code.
- Security - The layer that handles the security aspects.
- Execution Environment - Defines what methods and classes are available in a specific platform.



Layering

The OSGi has a layered model that is depicted in the following figure.

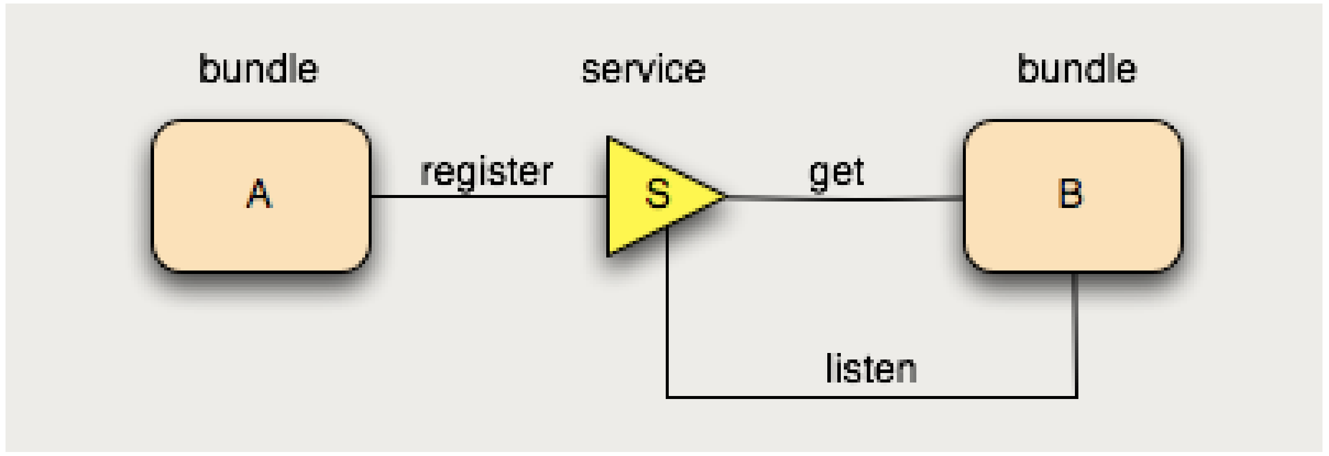
Bundles

modularity is at the core of the OSGi specifications and embodied in the bundle concept. In Java terms, a bundle is a plain old JAR file. However, where in standard Java everything in a JAR is completely visible to all other JARs, OSGi hides everything in that JAR unless explicitly exported. A bundle that wants to use another JAR must explicitly import the parts it needs. By default, there is no sharing

Bundles

modularity is at the core of the OSGi specifications and embodied in the bundle concept. In Java terms, a bundle is a plain old JAR file. However, where in standard Java everything in a JAR is completely visible to all other JARs, OSGi hides everything in that JAR unless explicitly exported. A bundle that wants to use another JAR must explicitly import the parts it needs. By default, there is no sharing

What can bundles do?



A bundle can

- register a service
- it can get a service,
- and it can listen for a service to appear or disappear.

Any number of bundles can register the same service type, and any number of bundles can get the same service

Why Are We Discussing OSGI?

More Questions ;)

- Can an application emerge from putting together different reusable components that had no a-priori knowledge of each other?
- Can an application emerge from dynamically assembling a set of components?

YES

Eclipse: IDE build using OSGI

Since 2003, the highly popular Eclipse Integrated Development Environment runs on OSGi technology and provides extensive support for bundle development

Plugins

In Eclipse the smallest unit of modularization is a plug-in. The terms plug-in and bundle are (almost) interchangeable. An Eclipse plug-in is also an OSGi bundle and vice versa.

Plugin:FLUORITE

a publicly available event logging plug-in for Eclipse which captures all of the low-level events when using the Eclipse code editor.

Plugin:FLUORITE

- FLUORITE can be used for not only evaluating existing tools, but also for discovering issues that motivate new tools.
- There are three different types of events that FLUORITE logs: commands, document changes, and annotations.
- A command is an event directly invoked by a user’s action. This includes typing new text, moving the cursor position or selecting text by keyboard or mouse, along with all editor commands such as copying, pasting, and undoing.
- A document change event is logged whenever the active file is changed by any executed command. Each document change event contains the actual deleted or inserted text. This is needed because we cannot correctly reproduce the snapshots of the files by capturing only the commands.
- An annotation is logged when the developer wants to add an annotation at a given time to provide information to the investigator about the current activity.

Can we extend plugins?

Extension

The process of adding some processing element or elements to a plug-in is known as an extension.

- This process is not restricted to UI elements.
- Any plug-in may allow other plug-ins to extend it by adding processing elements.
- An extension is defined by an extender plug-in and causes a host plug-in to modify its behavior.
- e.g., the addition of new menu items to the Eclipse workbench

Extension

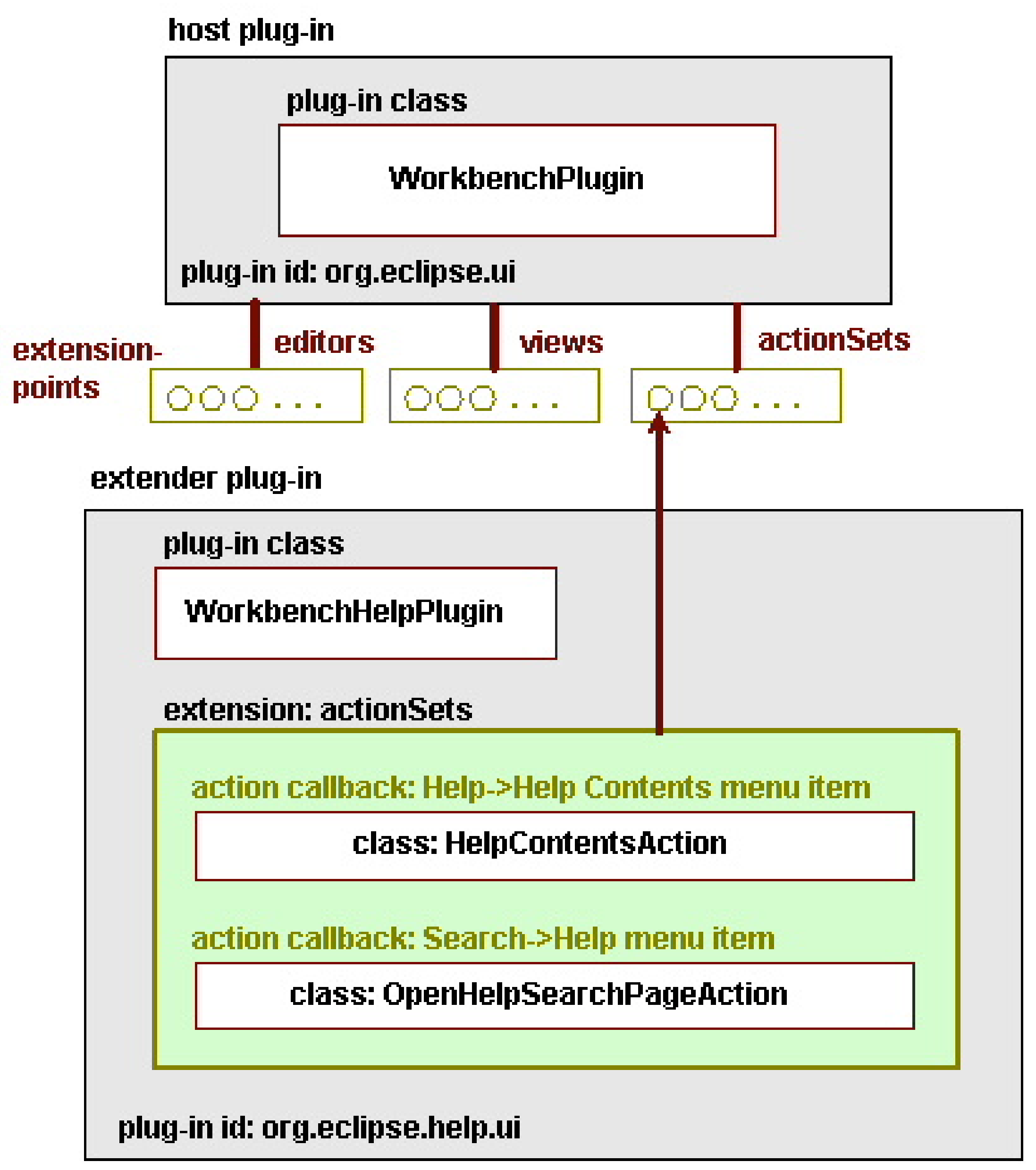

host plug-in
plug-in class
WorkbenchPlugin
plug-in id: org.eclipse.ui
extension-points
editors
views
actionSets
extender plug-in
plug-in class
WorkbenchHelpPlugin
extension: actionSets
action callback: Help->Help Contents menu item
class: HelpContentsAction
action callback: Search->Help menu item
class: OpenHelpSearchPageAction
plug-in id: org.eclipse.help.ui

Figure 1. Participants of a plug-in extension. The workbench UI plug-in is extended by the workbench help plug-in via an *actionSets* extension that defines specific help-related menu items.

Lazy Extension Processing

When a host plug-in is activated, an eager processing of its extensions would cause the activation of all of its extender plug-ins, and, recursively, their extender plug-ins, down the plug-in hierarchy As a result, an eager extension processing regime can considerably slow down plug-in activation, and therefore system startup.

Lazy Extension Processing

Using Virtual Proxies in Lazy Extension Processing can delay the creation of extender-specific callback objects, until such objects are actually required to perform some action.

Challenges: Building an IDE

- complexity : adding more features leads to complex system
- reuse: do not code that is available.
- adaptive components: what capabilities are available on the system and adapt the functionality they can provide.
- coupling: do we want different components to be tightly coupled to each other?
- version hell: A-> B.1 , C ->B.2
- lazy loading: Do stuff when asked to do.

What about Design Tasks?

Designing a complex software system is a cognitively challenging task; thus, designers need cognitive support to create good designs.

Argo (DODE)

Argo, a domain-oriented design environment for software architecture.

supports designers by providing

- external memory
- hiding non-essential details
- checking for inconsistencies or potential design flaws
- and providing design guidance, analysis, and visualization capabilities

Cognitive Theories

- **Reflection in Action:** design environments must provide design feedback to support decision making in the context of partial designs, i.e. while designs are being manipulated.
- **Opportunistic Design:** designers do not follow their own plans in order, but choose steps that are mentally least expensive among alternatives.

Argo features

- Visibility: What has been done, whats in the TODO?
- Flexibility: Allow them to deviate
- Guidance: suggests which of the many possible tasks the designer should perform next.
- Reminding: helps designers revisit incomplete tasks or overlooked alternatives.
- Timeliness: delivery of information to designers.

References

- https://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html
- http://www.osgi.org/Technology/WhatIsOSGi
- http://www.osgi.org/Technology/WhyOSGi
- http://www.vogella.com/tutorials/OSGi/article.html
- Robbins, J. E., Hilbert, D. M., & Redmiles, D. F. (1997). Extending Design Environments to Software Architecture Design.
- Kadia, P. R. (1992). Issues Encountered in Building a Flexible Software Development, 169–180.
- Yoon & Myers: Capturing and analyzing events low level event from the code editor (PLATEAU 2011)
- Ossher & Harrison: Support for Change in RPDE3 (SDE 1990)