GIPC Extensibility

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
GOALS

Customizable Implementation

Unlike RMI runs on Android

Informed by the IPC Design Space

How can we “improve” distributed IPC without language support?
IMPLEMENTATION GOAL

Extendible Implementation

Open Source

Almost no comments

“Layered”

Pluggable

Traceable
public enum PortKind {
    CLIENT_INPUT_PORT,
    SERVER_INPUT_PORT,
    SESSION_PORT,
    MULTI_SERVER_PORT
}

public enum PortMessageKind {
    BUFFER,
    OBJECT,
    RPC
}

public enum PortAccessKind {
    SIMPLEX,
    DUPLEX,
    GROUP
}

public interface PortDescription {
    PortKind getPortKind();
    void setPortKind(PortKind aPortKind);
    PortAccessKind getPortAccessKind();
    setPortAccessKind(PortAccessKind aPortAccessKind);
    PortMessageKind getPortMessageKind();
    setPortMessageKind(PortMessageKind aPortMessageKind);
...
public enum PortMessageKind {
    BUFFER,
    OBJECT,
    RPC
}

Remote procedure call
Object Communication
Byte Communication

Common generic interfaces for object and byte communication
Object hides byte but RPC sees Object
public enum PortAccessKind {
    SIMPLEX,
    DUPLEX,
    GROUP
}
```java
public enum PortKind {
    CLIENT_INPUT_PORT,
    SERVER_INPUT_PORT,
    SESSION_PORT,
    MULTI_SERVER_PORT
}
```
IMPLEMENTATION GOAL

- Extendible Implementation
- Open Source
- No comments
- “Layered”
- Pluggable
- Traceable
Plug in Models

- Add Observer
- Replace existing layer
- Interpose new layer between two existing layers
**Replace Example**

### Channel: Mechanism that delivers and receives messages on the network

<table>
<thead>
<tr>
<th>Application Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIPC Interfaces</td>
</tr>
<tr>
<td>GIPC Port Skeletons and Helper Classes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Port (Server and Client) Reliable Socket Drivers</th>
<th>Input Port (Server and Client) Reliable Socket Driver</th>
<th>Others Channel-Specific Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>NIO</td>
<td>Other</td>
</tr>
</tbody>
</table>

May want to choose which driver to use.
Factory-Based Instantiation

Factory instantiates a set of related types such as ports
- Provides a method for creating each type of object
- Each method takes instantiation parameters

Different implementations can instantiate different classes
- Typically instantiation parameters become constructor parameters

 void create\(^1\)(...)

 implements

 AFactory

 AnotherFactory

 void create\(^2\)(...)

**SIMPLEX INPUT PORT FACTORY INTERFACE**

- **SimplexInputPortFactory**
  - SimplexServerInputPort<MessageType>
    - createSimplexServerInputPort(String aServerId, String aServerName);
  - SimplexClientInputPort<MessageType>
    - createSimplexClientInputPort(String aHost, String aServerId, String aServerName, String aClientName);
**Simplex Input Port Factory**

SimplexInputPortFactory<MessageType> implements

```java
SimplexServerInputPort<MessageType>
createSimplexServerInputPort(String aServerId, String aServerName);

SimplexClientInputPort<MessageType>
createSimplexClientInputPort(String aHost, String aServerId, String aServerName, String aClientName);
```

Specific factories depend on message type
Classes instantiated depend on whether Socket or NIO is used for channel

How to choose between different factories?
GLOBAL ABSTRACT FACTORY

Static Global Registry

- SimplexBufferInputPortFactory
- DuplexBufferInputPortFactory
- GroupBufferInputPortFactory
- SimplexObjectInputPortFactory

Static Properties
- Has link to all factory instances
- Static methods to change and get references
- Can create a specification file to set the factories

Must change registry each time a new port is added
**Multiple GIPC Abstract Factories**

- **Factory Selector**
  - static Factory getFactory()
  - static setFactory(Factory f)
  - static create¹(...)  
  - static create²(...)

- **General Idea**
  - Helps choose between factories

- **GIPC**
  - Has a link to a Factory instance
  - Static methods to change and get reference
  - Can get reference to factory and invoke factory methods
  - Abstract factory provides static versions of factory creation methods for convenience
GIPC Simplex Abstract Factories

BufferSimplexInputPortSelector

- static setSimplexBufferInputPortFactory(SimplexInputPortFactory<ByteBuffer> factory)
- static SimplexInputPortFactory<ByteBuffer> getSimplexBufferInputPortFactory()
- static SimplexServerInputPort<ByteBuffer> createSimplexServerInputPort(String aServerId, String aServerName);
- static SimplexClientInputPort<ByteBuffer> createSimplexClientInputPort(String aHost, String aServerId, String aServerName, String aClientName);

Has-A

SimplexInputPortFactory<ByteBuffer>

Getter and setters for ByteBuffer simplex factories

Static versions of Factory operations bound to byte buffer
SimplexServerInputPort<ByteBuffer> aServerInputPort = BufferSimplexInputPortSelector.createServerSimplexInputPort(
  SERVER_PORT, SERVER_NAME);
aServerInputPort.connect();

SimplexClientInputPort<ByteBuffer> clientInputPort = BufferSimplexInputPortSelector.createClientSimplexInputPort(
  SERVER_HOST, SERVER_PORT, SERVER_NAME, clientName);
clientInputPort.connect();

How to assign factories to selectors?

How to reduce overhead of setting related factories for each configuration?
AnAbstract PortLauncher Support

- setTracing()
- setFactories()
- init()

extends *

ADuplexRPC InputPort Launcher Support

- setFactories()

ASyncReceive DuplexRPC InputPort Launcher Support

- setFactories()

Port launcher support sets factories for a set of related selectors

Can do additional actions such as set tracing for objects created by these selectors

Before a port is created, the init() call is expected to be called on one or more PortLauncherSupport objects, which are expected to set the factories needed to instantiate that port and set constructor parameters of the objects created by the factories.
LAUNCHER VS. LAUNCHER SUPPORT

AbstractPort Launcher

- launch()
- get(Auxiliary, Tertiary)PortLauncherSupport()
- get(Auxiliary, Tertiary)Port()
- createProxies()
- registerRemoteObjects()
- getConnectionListener()
- doPostConnectAsyncOperations()
- createUI()

gets port launcher support for main, auxiliary and tertiary ports) and initializes them (to create factories)
Plug in Models

- Add Observer
- Replace existing layer
- Interpose new layer between two existing layers
## Interposition Motivation

<table>
<thead>
<tr>
<th>Application Code</th>
<th>GIPC Interfaces</th>
<th>GIPC Port Skeletons and Helper Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Port (Server and Client) Reliable Socket Drivers</td>
<td>Input Port (Server and Client) Reliable Socket Driver</td>
<td>Others Channel-Specific Drivers</td>
</tr>
<tr>
<td>Socket</td>
<td>NIO</td>
<td>Other</td>
</tr>
</tbody>
</table>

- What if we want to add delays with messages?
- Do not want to duplicate code in different drivers
- Can change or subclass skeletons
- Need to worry about several unrelated concepts such as connection
- Sometimes all we want to do is trap receives and sends
Pipe-Based Extensibility

Application Code

GIPC Interfaces

GIPC Port Skeletons and Helper Classes

<table>
<thead>
<tr>
<th>Input Port (Server and Client) Reliable Socket Drivers</th>
<th>Input Port (Server and Client) Reliable NIO Driver</th>
<th>Others Channel-Specific Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>NIO</td>
<td>Other</td>
</tr>
</tbody>
</table>

Send

Receive

Producer

Consumer

Consumer

Producer

Consumer

Producer

Consumer

Producer

Can add arbitrary long chains of application message trappers between a producer/consumer pair
PROXIES VS. TRANSLATORS

Sender <Object> (Object Port)

Send Forwarder <Object, Object>

Serializer <Object, ByteBuffer>

Sender<ByteBuffer> (Buffer Skeleton)

Receive Notifier<Object>

ReceiveForwarder <Object, Object>

Deserializer <ByteBuffer, Object>

Receive Listener <ByteBuffer> (Buffer Skeleton)

Proxy

Translator
**Shared Trappers and Synchronizations**

Sender <Object> (RPC Port) -> Call Message and Return Message SendTrapper <Object, Object> -> Synchronizing Shared Data Structures

Receiver <Object> (Object Port) -> Call Message and Return Message ReceiveTrapper <Object, Object> -> Receive Listener <Object> (RPC Port)

Send trapper may need to block sender to implement synchronous operation

Send and receive trapper may need to share information about remote references sent and received
Send Trapper

SendTrapper <InMessageType, OutMessageType>

extends

UniNamingSender <OutMessageType>

send(String aRemoteEnd, MessageType aMessage)

has-a

implements

Implementation of SendTrapper <InMessageType, OutMessageType>

UniNamingSender <OutMessageType> Destination

Property with getter and setter

void setDestination(UniNamingSender <OutMessageType> d)

UniNamingSender <OutMessageType> getDestination()
Receive Trapper

ReceiveNotifier <InMessageType>

notifyPortReceive(String aRemoteEnd, MessageType aMessage);

has-a

extends

Receive Trapper <InMessageType, OutMessageType>

ReceiveNotifier <OutMessageType> destination

Implementation of AReceive Trapper <InMessageType, OutMessageType>
**ATTACHING TRAPPERS**

Receive Trapper
<InMessageType, OutMessageType>

GIPC Datacomm Port
<MessageType>

Send Trapper
<InMessageType, OutMessageType>

Application-defined trappers can be statically/dynamically attached to a data communication port

Factory selector used by port to initialize trapper (chain)

Port provides setters/getters for replacing or changing trapper chain

Predefined trappers for fault tolerance, causality, delays, and (de)serialization
IMPLEMENTATION GOAL

- Extendible Implementation
- Open Source
- No comments
- “Layered”
- Pluggable
- Traceable
TRACING WITH DEBUGGER

- Debugger makes it difficult to test race conditions
- All threads and processes mapped to a single code window
- Cannot see the history of actions taken by a thread
- Break points do not transfer to another computer
- Cannot use a mechanism to set multiple related debug points from a program
 TRACE

Please enter message:
The woods are lovely
I*** (inputport.datacomm.simplex.buffer) Forwarding message to send trapper:inputport
I*** (inputport.datacomm.simplex) Forwarding sent message java.nio.HeapByteBuffer
I*** (inputport.datacomm.simplex.buffer.nio) Sending message: java.nio.HeapByteBuffer
I*** (inputport.datacomm.simplex.buffer.nio) ABufferedWrite with id: 0 contents: java.
I*** (inputport.datacomm.simplex.buffer.nio) Started storing of buffered write with
I*** (inputport.datacomm.simplex.buffer.nio) Store operation op for: java.nio.channels.SocketChannel
I*** (inputport.datacomm.simplex.buffer.nio) Reverse cachedSelector select unblocks
I*** (inputport.datacomm.simplex.buffer.nio) Store operation op for: java.nio.channels.SocketChannel
I*** (inputport.datacomm.simplex.buffer.nio) Selector registering read as no pending
I*** (inputport.datacomm.simplex.buffer.nio) channel not connected or no pending write
I*** (inputport.datacomm.simplex.buffer.nio) Selector calls select

Tracer.showInfo(true);

May not want to to know about nio
Filtering: Selecting which events to print?
ONLY INPUTPORT.DATACOMM.SIMPLEX.BUFFER

AServerSimplexBufferInputPortLauncher [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Oct 17, 2011 4:22:27)
I*** (inputport.datacomm.simplex.buffer) Retrieved from global state receive trap;
I*** (inputport.datacomm.simplex.buffer) Changing connection status and asking driver;
I*** (inputport.datacomm.simplex.buffer) Received message: java.nio.HeapByteBuffer;
I*** (inputport.datacomm.simplex.buffer) Associating Alice with java.nio.channels.
I*** (inputport.datacomm.simplex.buffer) ServerInputPort connected to: java.nio.channels.
Echo Server<--->Alice (Opened)
I*** (inputport.datacomm.simplex.buffer) Received message: java.nio.HeapByteBuffer;
I*** (inputport.datacomm.simplex.buffer) ServerInputPort received message java.nio.cha;

AnAliceSimplexBufferInputPortLauncher [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Oct 17, 2011 4:22:33)
I*** (inputport.datacomm.simplex.buffer) Retrieved from global state send trapper;
I*** (inputport.datacomm.simplex.buffer) Set my send trapper to: inputport.datacomm.
I*** (inputport.datacomm.simplex.buffer) Adding send listener: inputport.datacomm.
I*** (inputport.datacomm.simplex.buffer) Asking driver to connect and changing status.
I*** (inputport.datacomm.simplex.buffer) Received connected notification from driver.
I*** (inputport.datacomm.simplex.buffer) Sending to server my name: Alice
I*** (inputport.datacomm.simplex.buffer) Forwarding message to send trapper: inputp
Alice<--->Echo Server (Opened)
I*** (inputport.datacomm.simplex.buffer) Received sent notification from driver
I*** (inputport.datacomm.simplex.buffer) Notifying to send listeners message: java.
Alice<--->Echo Server: (0) java.nio.HeapByteBuffer[pos=0 lim=5 cap=5]
Please enter message:
The woods are lovely
I*** (inputport.datacomm.simplex.buffer) Forwarding message to send trapper: inputp
Please enter message:
I*** (inputport.datacomm.simplex.buffer) Received sent notification from driver
I*** (inputport.datacomm.simplex.buffer) Notifying to send listeners message: java.
Alice<--->Echo Server: (1) java.nio.HeapByteBuffer[pos=0 lim=30 cap=30]
**Explicit Keywords**

```java
public static void info(String keyWord, String info);
Tracer.info("inputport.datacomm.simplex.buffer", "Asking driver to connect and changing status");
```

```java
public static void setKeywordPrintStatus(
    String keyWord,
    Boolean status);
Tracer.setKeywordPrintStatus("inputport.datacomm.simplex.buffer", true)
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
```

- Have to specify package name each time
- Package name can change
Implicit KeyWords

public static void info(Object object, String info);

Tracer.info(this, "Asking driver to connect and changing status");

public static void setKeywordPrintStatus(Class c, Boolean status);

Tracer.setKeywordPrintStatus(
    AGenericSimplexBufferClientInputPort.class,
    false);
**SHOWING PACKAGE NAMES**

Cannot identify classes of the objects printing out messages

Need to not only specify which events to display but what information to display about each event

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setKeywordPrintStatus(AGenericSimplexBufferClientInputPort.class, true);
Tracer.setMessagePrefixKind(MessagePrefixKind.PACKAGE_NAME);

Received sent notification from driver
Alice-->Echo Server:(0) java.nio.HeapByteBuffer[pos=0 lim=5 cap=5]
Please enter message:
The woods are lovely
Forwarding message to send trapper:inputp
Please enter message:
Received sent notification from driver
Alice-->Echo Server:(1) java.nio.HeapByteBuffer[pos=0 lim=30 cap=30]
SHOWING SHORT CLASS NAMES

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setKeywordPrintStatus("AGenericSimplexBufferClientInputPort.class", true);
Tracer.setMessagePrefixKind(MessagePrefixKind.SHORT_CLASS_NAME)

I*** (AGenericSimplexBufferClientInputPort) Received sent notification from driver
I*** (ASendRegistrarAndNotifier) Notifying to send listeners message: java.nio.ByteBuffer
   Alice --> Echo Server: (0) java.nio.ByteBuffer[pos=0 lim=5 cap=5]
   Please enter message:
   The woods are lovely
I*** (AGenericSimplexBufferClientInputPort) Forwarding message to send trapper:inputpc
   Please enter message:
I*** (AGenericSimplexBufferClientInputPort) Received sent notification from driver
I*** (ASendRegistrarAndNotifier) Notifying to send listeners message: java.nio.ByteBuffer
   Alice --> Echo Server: (1) java.nio.ByteBuffer[pos=0 lim=20 cap=20]
`public static void setMessagePrefixKind(
    MessagePrefixKind newValue)`

`public enum MessagePrefixKind {
    NONE,
    OBJECT_TO_STRING,
    SHORT_CLASS_NAME,
    FULL_CLASS_NAME,
    PACKAGE_NAME
}`
Displaying All Classes in Package

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setKeywordPrintStatus(AGenericSimplexBufferClientInputPort.class, true);
Tracer.setMessagePrefixKind(MessagePrefixKind.SHORT_CLASS_NAME)

What if we want to focus on one class?
Narrow down which events
CONTROLLING IMPLICIT KEYWORD

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setImplicitKeywordKind(ImplicitKeywordKind.OBJECT_CLASS_NAME);
Tracer.setKeywordPrintStatus(
    AGenericSimplexBufferClientInputPort.class, true);
Tracer.setMessagePrefixKind(MessagePrefixKind.SHORT_CLASS_NAME);
**Implicit Keyword**

```java
public static void setImplicitKeywordKind(ImplicitKeywordKind newValue)
```

```java
public enum ImplicitKeywordKind {
    OBJECT_TO_STRING,
    OBJECT_CLASS_NAME,
    OBJECT_PACKAGE_NAME
}
```
setKeywordPrintStatus(String keyword, Boolean status);

info(String keyword, String info);

setImplicitKeywordKind(ImplicitKeywordKind newValue);

info(Object obj, String info);

public enum ImplicitKeywordKind {
    OBJECT_TO_STRING,
    OBJECT_CLASS_NAME,
    OBJECT_PACKAGE_NAME
}

setMessagePrefixKind(MessagePrefixKind newValue);

public enum MessagePrefixKind {
    NONE,
    OBJECT_TO_STRING,
    SHORT_CLASS_NAME,
    FULL_CLASS_NAME,
    PACKAGE_NAME
}
DEBUGGING CAPABILITIES IN TRACER?

- Blocking but separate windows for different processes
- See state at traced actions (with and without blocking)
- Separate state for different threads (with and without blocking)
- Have application-specific code learn about traced calls (perhaps in different processes)
TRACING STRINGS \rightarrow OBJECTS

```java
setKeywordPrintStatus(String keyWord, Boolean status);

info(String keyWord, String info);

setImplicitKeywordKind(ImplicitKeywordKind newValue);

info(Object obj, String info);

setKeywordPrintStatus(Class cls, Boolean status);

public enum ImplicitKeywordKind {
    OBJECT_TO_STRING,
    OBJECT_CLASS_NAME,
    OBJECT_PACKAGE_NAME
}

setMessagePrefixKind(MessagePrefixKind newValue);

public enum MessagePrefixKind {
    NONE,
    OBJECT_TO_STRING,
    SHORT_CLASS_NAME,
    FULL_CLASS_NAME,
    PACKAGE_NAME
}
```
public class MVCTraceableInfo extends TraceableInfo{
    public MVCTraceableInfo(String aMessage, Object anAnnouncer) {
        super(aMessage, anAnnouncer);
    }
    public static MVCTraceableInfo newInfo(String aMessage, Object aFinder) {
        MVCTraceableInfo retVal = new MVCTraceableInfo(aMessage, aFinder);
        retVal.announce();
        return retVal;
    }
}

Tracer.info(this, "MVC structure built")

MVCTraceableInfo( "MVC structure built", this);
ObjectEditor.edit(
    TraceableDisplayAndWaitManagerFactory.getTraceableDisplayAndPrintManager());

EventAnnouncerThread

newEvent()

newEvent()
MESSAGE BUS

Like an observable it has registration method

Has an announce method

Does not generate events – simply communicates them to observers
RUN FUNCTIONALITY
public ADuplexRPCClientRelayingCollaborativeMVCLauncher(
    String aClientName, String aServerHost,
    String aServerId, String aServerName) {
    super(aClientName, aServerHost, aServerId, aServerName);
    ObjectEditor.edit
        (TraceableDisplayAndWaitManagerFactory.
            getTraceableDisplayAndPrintManager());
    Tracer.setKeywordDisplayStatus(this, true);
}
Each process can have separate trace window.

Server

Alice

Bob
Each Trace Window has Separate Thread Area

Thread interacting with underlying communication channel

Thread invoking remote calls
Thread display

Selecting Thread
port.trace.CallInitiated: Queued: inputport.rpc.g

1: Forward Back Proceed

Asynchronous Received Call Invoker
inputport.rpc.group.mvc.collaborative.relaying.e

2: Forward Back Proceed

Thread name

Other info
**GOALS**

Debugger allows programmer to set breakpoint, resume, and at each breakpoint, pause and see stack trace, variable values, and console message when some line of code is executed.

Tracer should provide equivalent.
CONSOLE OUTPUT EQUIVALENT: TYPED TRACING MESSAGE

GIPC-Defined Call Initiated Type

Instance specific message

Port trace Call Initiated: Queued: inputport.rpc.group.mvc.collaborative.relaying.example.ARelayingCollaborativeUpperCaserRelayToOthers[Alice said: WOODS(1, 0)]

Programmer-defined MVCTraceable Type

Instance specific message

asyncronous received call invoker

Inputport.rpc.group.mvc.collaborative.relaying.example.MVCTraceableInfo: Server Relaying Output

getMessage() declare

setMessage() declare

Traceable

announce() declare
STACK TRACE EQUIVALENT: STACK TRACE

[Diagram showing stack trace options and actions]
## Stack Trace Display

<table>
<thead>
<tr>
<th>Class Name</th>
<th>File Name</th>
<th>Line Number</th>
<th>Method Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>port.trace.CallInitiated</td>
<td>CallInitiated.java</td>
<td>21</td>
<td>newCase</td>
</tr>
<tr>
<td>input.rpc.simplex.ASim</td>
<td>ASimplexCallReceiveTrap</td>
<td>38</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.rpc.duplex.ADuplex</td>
<td>ADuplexCallReceiveTrap</td>
<td>28</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.rpc.simplex.ASim</td>
<td>ASimplexRPCServerInput</td>
<td>122</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm.AReceive</td>
<td>AReceiveRegistrarAndNoti</td>
<td>28</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.AReceive</td>
<td>AReceiveMessageForward</td>
<td>20</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.group</td>
<td>AnAbstractGroupInputPort</td>
<td>243</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm.groups</td>
<td>AReceiveRegistrarAndNoti</td>
<td>28</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.groups</td>
<td>AReceiveMessageForward</td>
<td>20</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>ADeserializingForwarder</td>
<td>27</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>ADeserializingForwarder</td>
<td>1</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>ASimpleXObjectServerInput</td>
<td>54</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>ASimpleXObjectServerInput</td>
<td>1</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm.AReceive</td>
<td>AReceiveRegistrarAndNoti</td>
<td>28</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.AReceive</td>
<td>AReceiveMessageForward</td>
<td>20</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.group</td>
<td>AnAbstractGroupInputPort</td>
<td>243</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm.groups</td>
<td>AReceiveRegistrarAndNoti</td>
<td>28</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.groups</td>
<td>AReceiveMessageForward</td>
<td>20</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>AGenericSimplexBufferServer</td>
<td>156</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>AGenericSimplexBufferServer</td>
<td>87</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>AnNIOSimplexBufferServer</td>
<td>92</td>
<td>socketChannelRead</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>AReadCommand.java</td>
<td>80</td>
<td>notifyRead</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>AScatterGatherReadComm</td>
<td>99</td>
<td>maybeExtractMessageAnd</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>AScatterGatherReadComm</td>
<td>35</td>
<td>processRead</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>AReadCommand.java</td>
<td>43</td>
<td>execute</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>ASelectionReadManager</td>
<td>45</td>
<td>processRead</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>ASelectionManager.java</td>
<td>180</td>
<td>processSelectedOperation</td>
</tr>
<tr>
<td>input.datacomm.simple</td>
<td>ASelectionManager.java</td>
<td>207</td>
<td>run</td>
</tr>
</tbody>
</table>
INHERITANCE HIERARCHY

```
I
NHERITANCE
HIERARCHY
```

```
Runnable
    - start()

Thread
    - run()

Object
    - getClass()

Throwable
    - toString()
    - getMessage()
    - getStackTrace()

Exception
    - getLocalizedMessage()
    - printStackTrace()

RuntimeException
    - cause()
    - cause()
```

```
Traceable
    - announce()
    - getMessage()
    - setMessage()
    - getStackTrace()

IS-A

getStackTrace()

setMessage()

announce()

getStackTrace()
```
EXPLORING VARIABLES: EVENT PROPERTIES

Can see properties of event by calling getters
Thread Has History of Typed Trace Info
BROWSING: LAST TO LAST CALL
Resume: Proceed Button
SET BREAK POINT: PAUSE/DISPLAY ALL ANNOUNCED MESSAGES?

May want to pause only some of the announcements

May want to display only some of the announcements

How to specify a set of related announcements that should be displayed or paused?
**Using Announcer Object Attributes**

```java
public enum ImplicitKeywordKind {
    OBJECT_TO_STRING,
    OBJECT_CLASS_NAME,
    OBJECT_PACKAGE_NAME
}
```

- `setKeywordDisplay(Wait)Status(announcer, true(false))` says that if an event is announced by an object whose `toString()`/class/package attribute is that of the announcer then it should be displayed(wait).

- `setImplicitDisplay(Wait)KeywordKind` determines if `toString()`, class or package attribute is used for all events.

- Print, Display and Wait are three different things you can do with traced information.
USING EVENT OBJECT

Traceable

getMessage() announce()

setMessage() setWait()

setDisplay()
Debugger Issues Resolved

Debugger makes it difficult to test race conditions

All threads and processes mapped to a single code window

Cannot see the history of actions taken by a thread

Break points do not transfer to another computer

Cannot use a mechanism to set multiple related debug points
A Message bus allows communication of events among a set of related objects such as debugger and editor.

- An event announced by an object are sent to all objects listening to it.

A Message bus can use a semi-wait free algorithm to avoid conflicting concurrent announcements.
SUMMARY

- A Traceable bus is a message bus that allows one to simultaneously see the values of Traceable objects announced by different threads of a process.
- One of the listeners of these objects is a single ATraceable DisplayAndWaitManager.
- The notification callback called in this object in response to the announcement of a Traceable:
  - Displays information about the Traceable if its Display property is true.
  - Blocks the thread until the user unblocks it if the Wait property of the Traceable is true.
- Primitives are provided to set the display and wait property of all Traceables announced by
  - all code in a class,
  - all code in a package.
- A Message bus can use a semi-wait free algorithm to avoid conflicting concurrent announcements.