The Story Begins
The Setting

- RBC Capital Markets
- Global Fixed Income Division

- Global development and user base
  - New York, London, Toronto

- Main focus on front office trading applications
  - Real-time display of fast-moving market data

- Numerous trading products
  - Note instruments
  - FRNs
  - Cash instruments (bonds, futures)
  - Derivatives
A Little History

- **Application development historically product-focused**
  - Trading localized to product types
  - Multiple product-specific application systems
  - Customized to needs of individual products
  - Varying levels of functionality across product application systems

**Note Instruments**
- Pricing
- Risk
- Trade Capture
eCommerce

**FRNs**
- Pricing
- Risk
- Trade Capture
eCommerce

**Cash Instruments**
- Pricing
- Risk
- Trade Capture
eCommerce

**Derivatives**
- Pricing
- Risk
- Trade Capture
eCommerce
A Little History, continued

- **Shift to functional focus**
  - Cross-asset trading
  - Any product, any region, any currency
  - Increased need for functional integration
  - *New development; Multiple function-specific applications*

### Diagram

- **Pricing**
- **Risk**
- **Trade Capture**
- **eCommerce**

#### Instruments
- Note Instruments
- FRNs
- Cash Instruments
- Derivatives
User Experience – What We Have

- Product-specific applications
- Function-specific applications
- Lots of user context switching across application UIs
- Reduced efficiency, increased cost
- Big problem

<table>
<thead>
<tr>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
<th>Image 4</th>
<th>Image 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image 1" /></td>
<td><img src="image2.png" alt="Image 2" /></td>
<td><img src="image3.png" alt="Image 3" /></td>
<td><img src="image4.png" alt="Image 4" /></td>
<td><img src="image5.png" alt="Image 5" /></td>
</tr>
</tbody>
</table>
User Experience – What We Want

- One integrated UI where users can access relevant product and functional application components
- Extremely high value to users
- It can be done!
  - Eclipse demonstrates this
  - Eclipse solves many of the hard problems
Things That Are In Common

- Bootstrap installer
- Unified user authentication (SSO)
- Automatic application component provisioning
- Inter-component communication mechanisms
- Master application UI window
- UI extension points
- Common look and feel for components
- User preference management
- Component model
Things That Are Different

- Many different application components
- Multiple independent application development teams
- Different development, testing and release schedules
- Different users (e.g. traders, salespeople, support staff) will need different sets of application components
➢ Eclipse RCP-based UI container
➢ Hosts multiple application plugins
➢ Provides common services, mainly through plugins
  • Eclipse, Spring, etc
➢ Insulates component developers from some degree of complexity
  • Pre-built, pre-configured RCP application
  • Encapsulates common environment configuration, e.g. connection to RBC Active Directory LDAP
A little less than a year old, 1 developer for most of that time

Lots of functionality comes for free with Eclipse, Spring plugins

Had to build some things; had to configure and integrate a lot of things
Installation and Provisioning

- **Installer**
  - One installer for everyone
  - Simple, standard installation for all workstations

- **Executable**
  - One executable for everyone
  - Application components are automatically downloaded and installed when the user logs in

- **Provisioning**
  - Users are associated with provisioning groups
  - Provisioning groups have associated product configurations which specify what components to install for members of the group
  - P2-based automatic provisioning
Authentication and Authorization

- Login dialog is the first thing the user sees
- Uses Spring Security
- Authenticates against RBC Active Directory LDAP
- Users can be associated with one or more security roles
- User security role info is installed in the SecurityContext on successful login
- The SecurityContext can then be queried for role membership to drive authorization behavior, e.g.:
  ```java
  if (AuthUtil.hasAuthority("ROLE_ADMINISTRATOR")) {
    // do something
  }
  ```
- The SecurityContext is available to any component hosted in Helios for authentication/authorization
Master Application Window
Common Look And Feel

- Standard set of UI controls
  - Mostly stock SWT controls
  - A few custom controls (date picker, data grid)
- Eclipse Forms Framework
- Built UI framework on top of above to encapsulate higher-order behavior and associations
  - Labels associated with form fields
  - Required field rendering
  - Etc
- Looking to also provide declarative UI specification
  - Eclipse modeling tools
Inter-Component Communication

➢ OSGi services
   • Can easily expose classes as OSGi services or get references to OSGi services via Spring DM
   • `<osgi:service ref="application" interface="org.eclipse.equinox.app.IApplication"/>
   • `<osgi:reference id="ldapAuthoritiesPopulator" interface="org.acegisecurity.providers.ldap.LdapAuthoritiesPopulator"/>

➢ Eclipse extension points

➢ Spring Remoting
• Eclipse automatically saves user preferences on application shutdown and restores UI state when the application is restarted

• Eclipse normally stores user preferences in the workspace folder

• We want to move preferences to the server side
  • Allow preferences to be shared between users
  • Allow support to retrieve and locally apply a user’s preferences for troubleshooting purposes
The Wider World
Think Different

- Major shift from monolithic to component-oriented development
  - Developing parts rather than entire applications
- Much more reliant on common/shared stuff
  - Technology stack, component model, development infrastructure
- Requires coordination across development teams
  - Shared vision of macro application functionality
  - Need to make sure parts can work together

- The most crucial issue in making component-oriented development work is not really a technical one – it is figuring out how to work together as part of a larger development community

- Not just interested in Eclipse technologies, but also Eclipse development model
Moniker for shared development at RBC
Emulating Open Source development model

Core
- Core Technology team
- Coordinate and are 100% allocated to Fusion projects

Contributors
- Individuals from other teams that develop for Fusion
- Allocated part time
- Represent application requirements in Fusion development and communicate Fusion knowledge back to application team

Community
- People and teams that use Fusion products
Development Process Infrastructure

- Version control – source code repository, release tags, maintenance branches
- Automated build – build from source, continuous integration
  - Custom maven plugin wrapper for PDE Build
    - Encapsulates build conventions (directory locations, etc)
    - Reduces number of required configuration parameters
- Artifact repository – well-known location to find released artifacts
- Issue tracking system – plan and track releases
- Wiki – knowledge repository, reference information
- Forum – public discussions, support

- Favor lots of transparency, accessibility
- Always looking to improve our process and tooling
Rules Of Thumb

- Make sure you’re building something people need, with the quality that they want.
- Make others part of the solution and then you won’t have to worry so much about adoption.
- Get support from the top (management) to make resource and schedule accommodations for shared technology projects.
Epilogue
So Here We Are

- Helios currently used by 2 FI applications in standalone mode
- Those and several other applications will be starting to migrate onto Helios in shared mode in early 2009

- Strategic application integration platform for RBC Fixed Income applications
- Impacting how applications are being developed and composed
Parting Shots

- Combination of Eclipse and Spring is extremely powerful
- Technology alone is not sufficient for success
- Need to pay a lot of attention to how the technology will be used and who will be using it

- Eclipse allowed us to address the really high value problem first
- This is just the beginning, there’s a long way to go from here
  - Adding additional shared functionality to Helios
  - Other middle and backend integration infrastructure