

The Eclipse 3.2 Debug Platform: Supporting a community of debuggers

Darin Wright and Samantha Chan
IBM Rational Software

Road Map

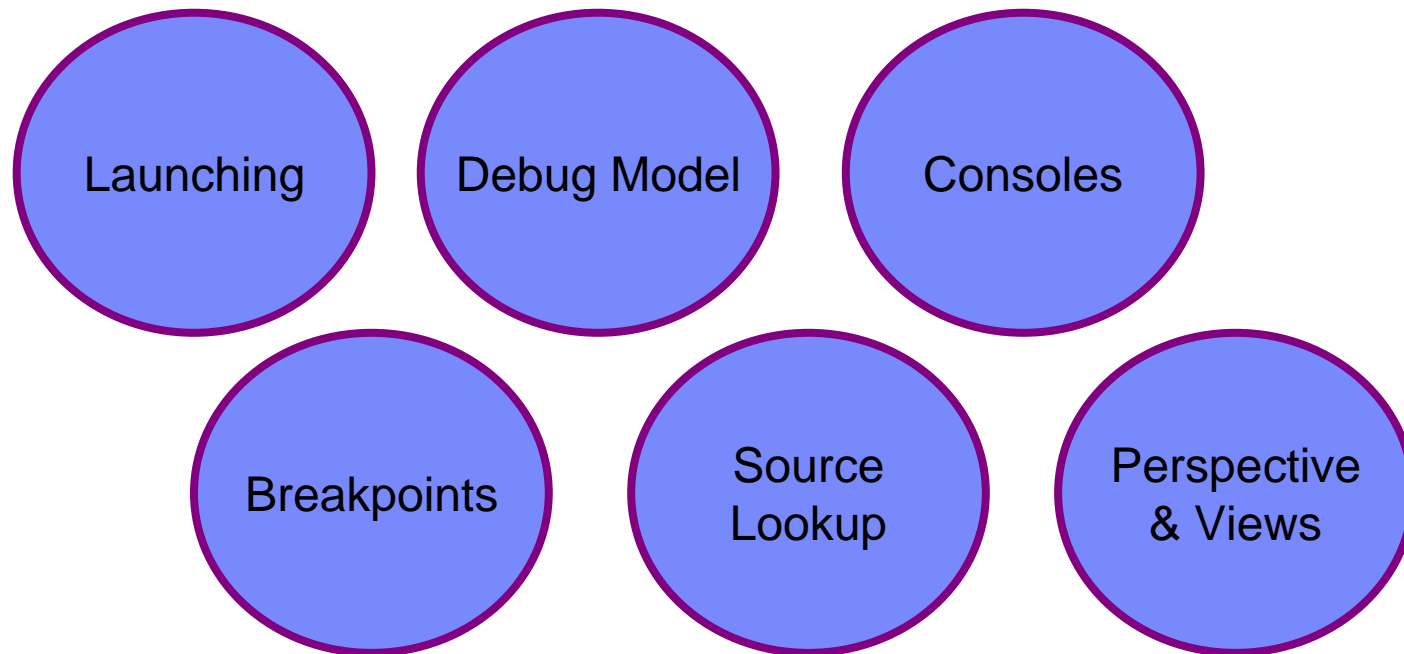
- The Community at Work
 - Evolution of the debug platform
- Being a Flexible Platform
 - Technical challenges and solutions
- A Tour of Other Enhancements
 - Memory view
 - Launching

The Community at Work

Evolution of the debug platform

A Long Time Ago...

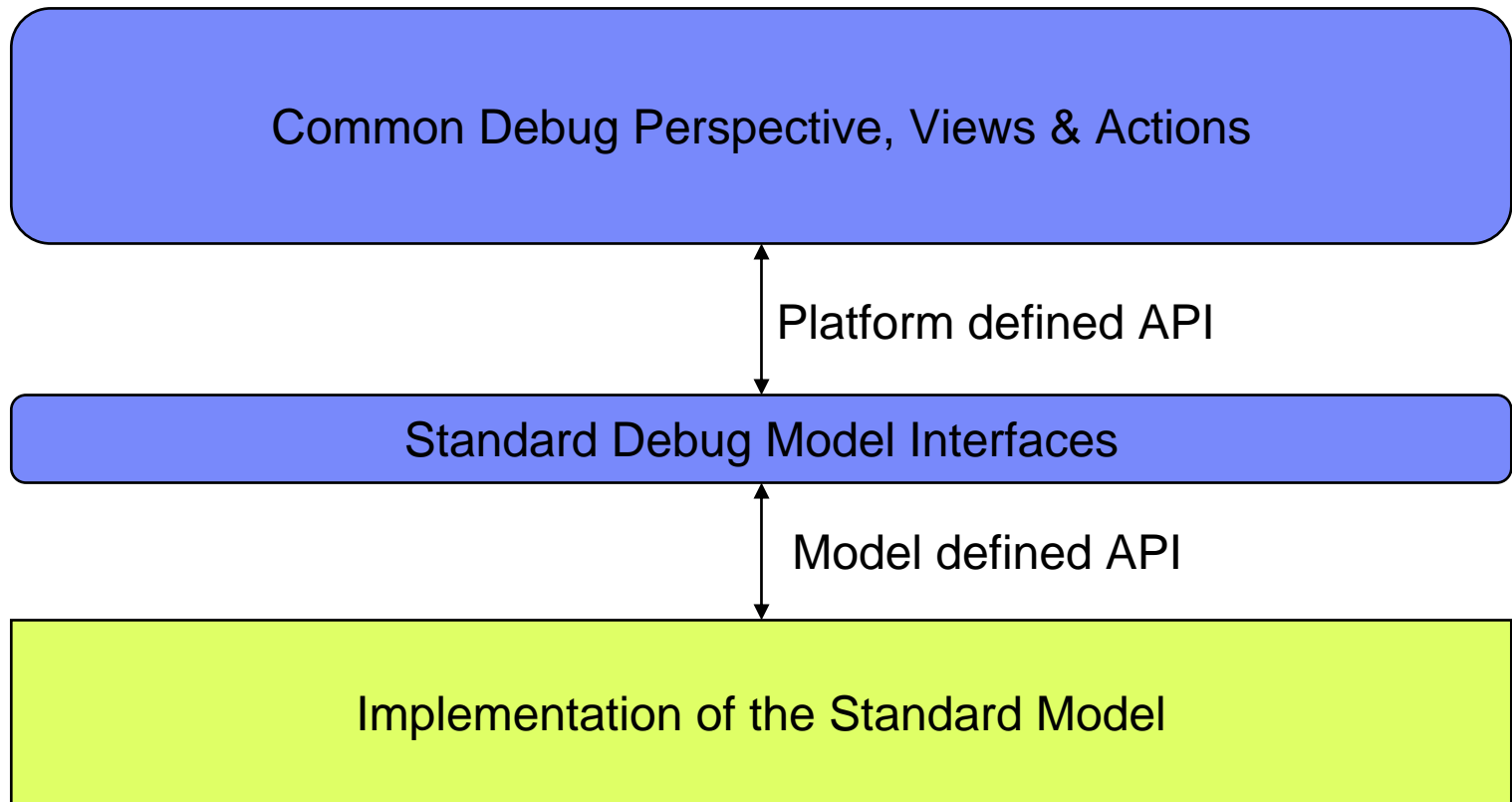
Debug Platform Facilities and Frameworks



A Platform Was Born

- Hypothesis
 - Most debuggers are very similar
 - A debuggable process made up of threads, stack frames, variables, etc., with common functions like stepping, terminating, suspending, etc.
 - The platform should provide
 - Abstractions for the common entities and functions (interfaces that will be implemented by each debugger)
 - A user interface that works against the abstractions and functions (debug, variables, breakpoint views with standard debug toolbar)
- Sales pitch
 - All you have to do is provide an implementation of the core interfaces specific to your debug architecture, and you get basic a debugger without writing any UI code

Standard Debug Platform



And Lived Happily Ever After

- Many debuggers were built on the platform
 - JDT, CDT
 - IBM:
 - Compiled Language Debug
 - Websphere Application Server Debug
 - JSP™ Debug
 - JavaScript™ Debug
 - DB2 Stored Procedure Debug
 - EGL Debug
 - Others

Good Things Don't Last Forever

- Caveat: a weak, general solution
 - Defense: a debugger can extend the generic user interface with custom views and actions to expose its special features
- General unrest
 - Over the last few years an increasing number of developers found the platform did not meet their needs
 - DSDP – Device Software Development Platform
 - PTP – Parallel Tools Platform

A Peaceful Revolution

- Debug community gathered to discuss shortcomings in the platform and gather requirements
 - Disguised intervention to get the platform to admit shortcomings
 - Participating companies included:
 - Accelerated Technologies (Mentor Graphics), AMI Semiconductor, Antenna, Apogee Software, EclipseME, Freescale, Digi, HP, IBM, Intel, MontaVista, Nokia, PalmSource, QNX, ShareME Technologies, SonyEricsson, Texas Instruments, Timesys, Wind River Systems, and Wirelexsoft

Key Issues

1. The standard debug model doesn't represent all architectures
 - Embedded hardware models are different – often there are multiple processors in multi-core, possibly with DSP configurations
2. Standard debug views and actions place high demands on the underlying target and enforce a synchronous interaction
 - View content and labels are retrieved synchronously in UI thread

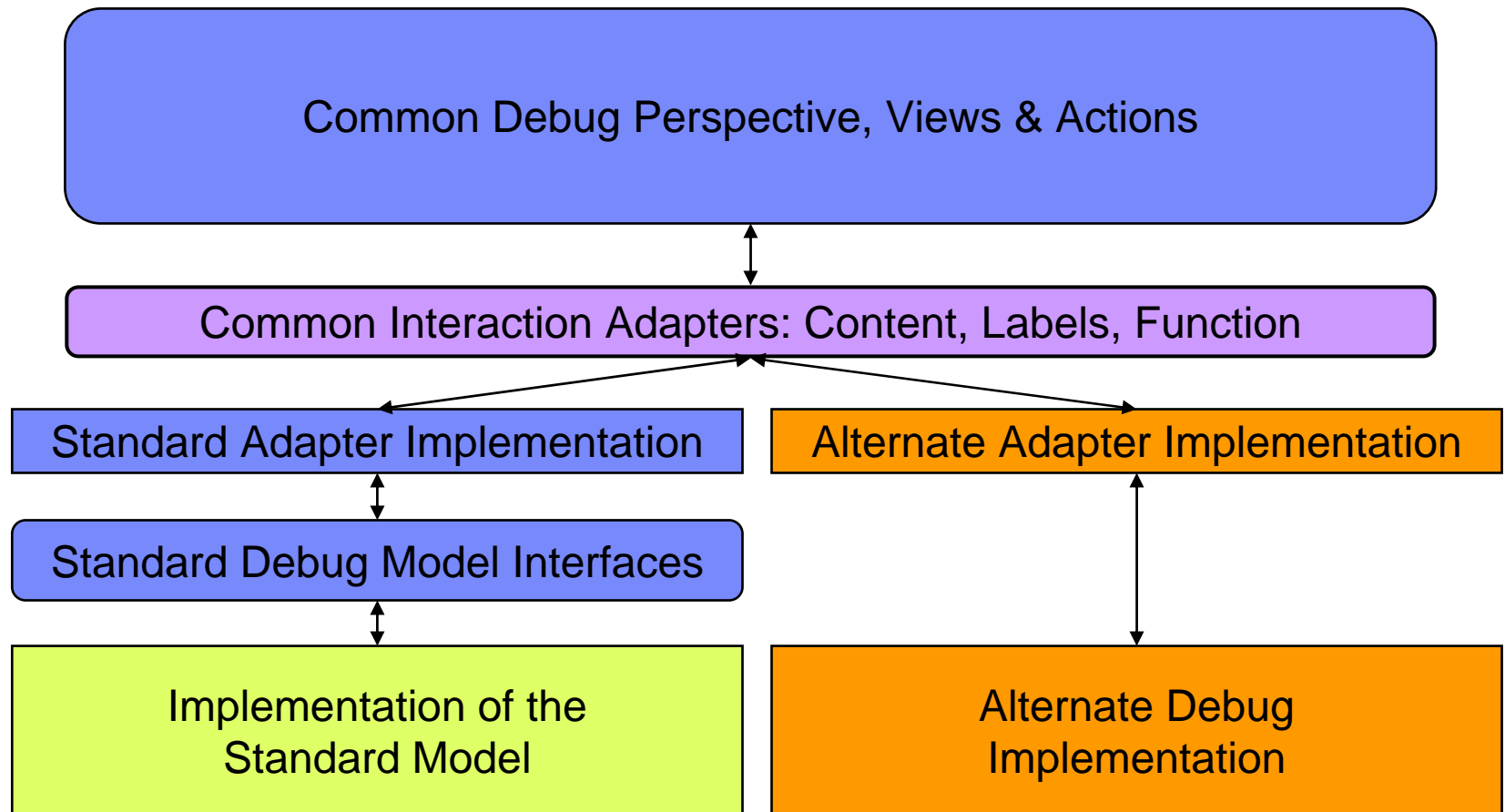
Key Requirements

- A flexible debug element hierarchy in all debug platform views
 - Additionally, support model specified columns
- A pluggable update mechanism for all views
 - Models control what gets updated and when
- Support asynchronous interactions between UI and model
 - Interactions must not block the UI thread and be cancelable
 - Allow implementations to coalesce requests
- Flexible view wiring and retargettable actions
 - For example, the input to the variables view may not be a frame
- Enhanced support for debugging multiple sessions
 - Different sets of views for different sessions

Realization: One Requirement

- The debug platform should support any debug implementation
 - The “standard debug model” is one possible implementation

Adaptable Debug Platform



Evolving the Platform: QA from the Community

- With input from the community, solutions were proposed to address the key issues and requirements
 - Solutions presented and discussed at face to face meetings
 - Implementation has undergone typical iterative refinement
- To ensure the platform provides viable, long term solutions, the new support will be provided as provisional APIs in 3.2
 - Community needs time to evaluate and experiment with the APIs
 - Provides adequate time for feedback and revision before APIs are committed as public

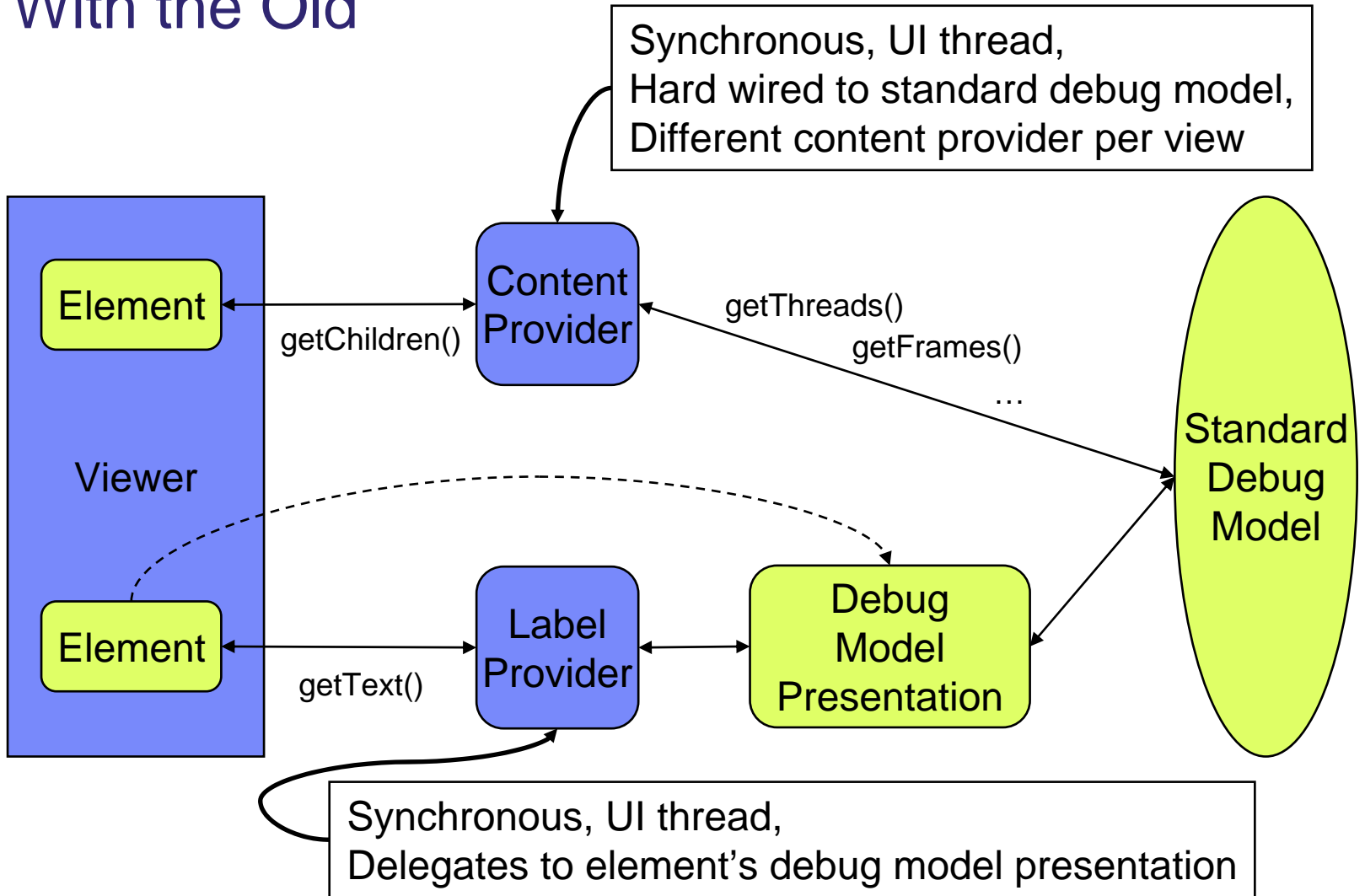
Being a Flexible Platform

Technical challenges and solutions

Challenge: Model Driven Content and Labels

- Requirements
 - Display arbitrary element hierarchies in all debug views
 - Interactions to retrieve content and generate labels must be asynchronous and cancelable, non UI thread
 - Client must have complete control over the implementation of content retrieval – i.e. whether requests are processed in parallel vs. sequential, whether requests are coalesced, etc.

Out With the Old



Re-inventing the Wheel?

- What about **IWorkbenchAdapter** ?

- `getChildren(Object)`

- `getImageDescriptor(Object)`

- `getLabel(Object)`

- `getParent(Object)`

- `getBackground(Object)`

- `getForeground(Object)`

- `getFont(Object)`

We Need a Different Wheel

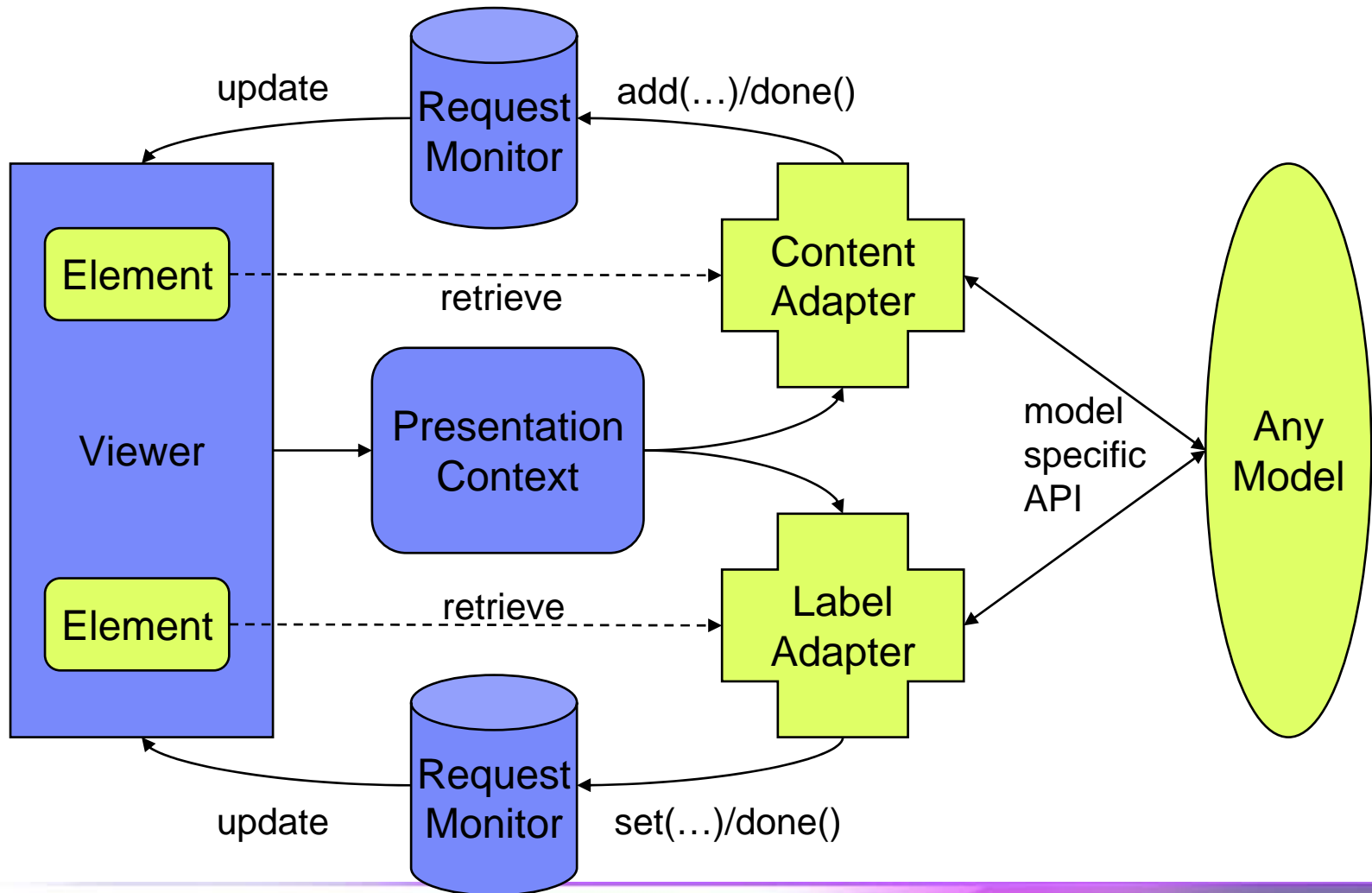
- Limitations
 - Does not allow for different content/labels in different views for the same element, as no “context” is provided
 - For example, a stack frame’s children are variables in the variables view and registers in the registers view
 - Enforces an element to know its single parent
 - Not all objects know their parent – for example a value (piece of memory) does not know what variable is referencing it
 - The same object may have many parents – more than one variable can reference the same memory

Workbench Asynchronous Content

- What about **IDeferredWorkbenchAdapter** ?
 - The workbench supports asynchronous content via this interface

```
fetchDeferredChildren(Object, IElementCollector,  
    IProgressMonitor)  
isContainer()  
getRule(Object object)
```
 - Still no context for view dependant content
 - Labels are computed synchronously in UI thread via label provider
 - Enforces implementation with a content manager that uses jobs
 - Community expressed requirement in having complete control on how and if jobs are used to schedule and retrieve content

In With the New: Content & Labels



Properties of the Interaction

- The interaction is **extensible**
 - Extensibility provided by adapters, allows for arbitrary model API
- The interaction is **context sensitive**
 - The *presentation context* identifies the part requesting content, allowing for different content in different views
- The interaction is **asynchronous**
 - The adapter retrieves content in a non-blocking fashion, reporting results to a *request monitor* and notifying it when done
- The interaction is **cancelable**
 - The request monitor is a special progress monitor allowing any request to be canceled by the platform or model
- The interaction **reports errors**
 - The request monitor carries a status to inform the user of failures that arise

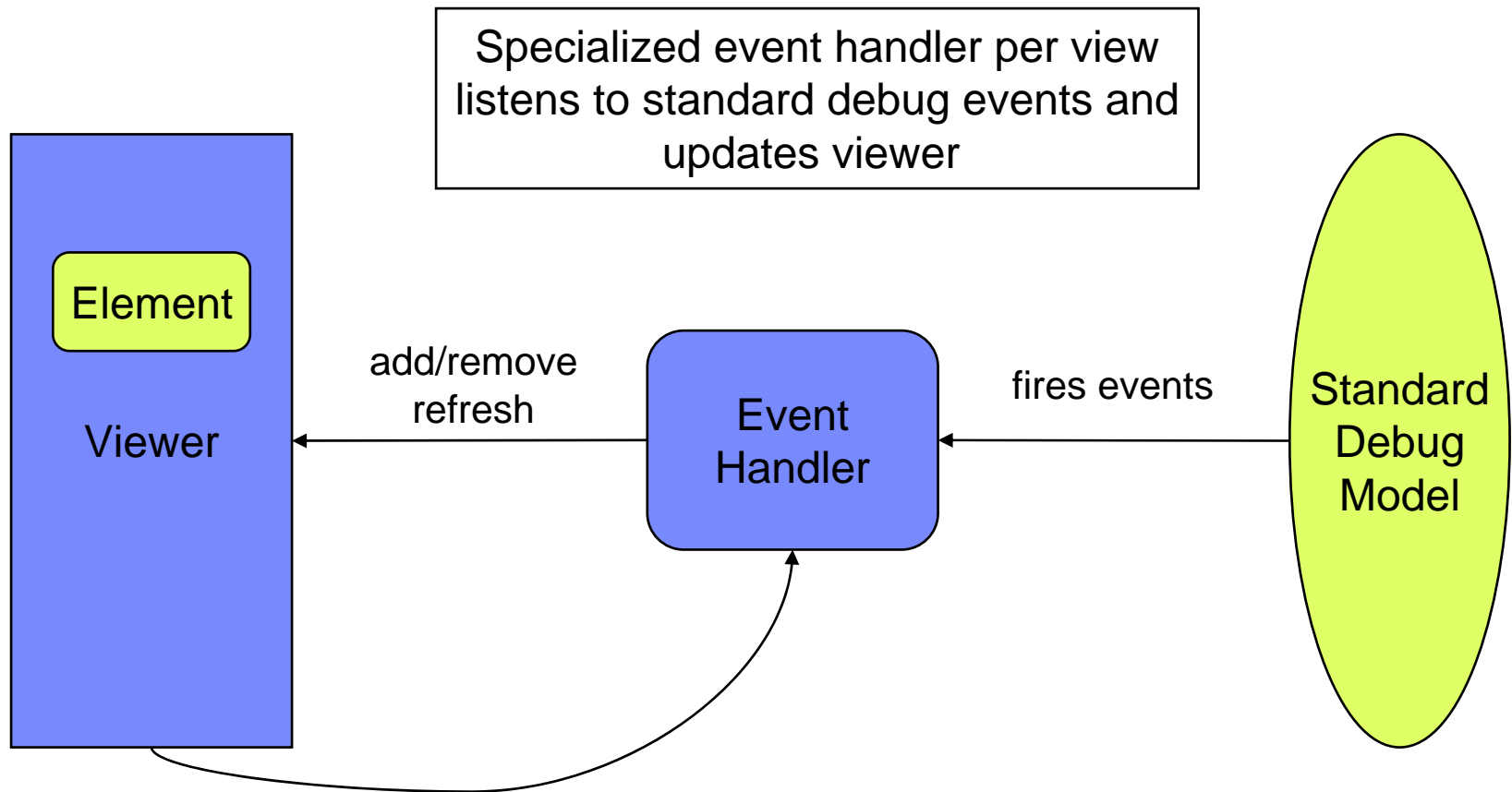
Under the Hood

- The debug platform implements custom viewers
 - Leverage SWT.VITRUAL technology
- The viewers deal with asynchronous content at all levels
 - Viewer API to refresh, add, etc., just initiates an operation creating a ripple of update requests as content is retrieved triggering subsequent updates
 - You can set a selection or expansion before the content exists
 - Methods can be called in non-UI threads
- The viewers are not currently provisional API
 - The viewer implementation is the most volatile piece of the implementation
 - We have been able to evolve the viewer implementation independently of the adapters that interface with it

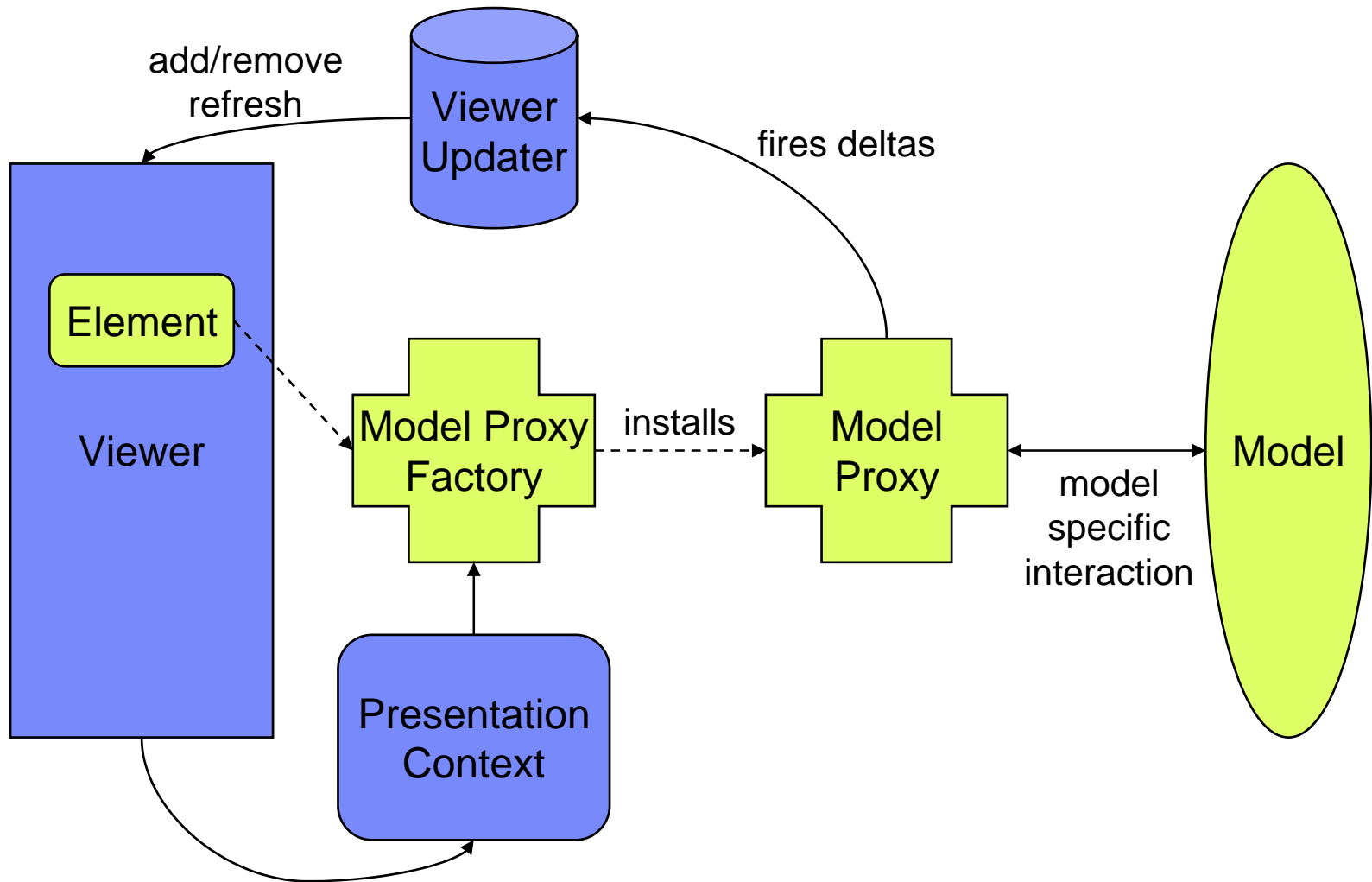
Challenge: Model Driven Updates

- Requirements
 - Extensible mechanism for updating arbitrary element hierarchies in all debug views
 - Clients must have fine grained control over what and when elements get refreshed

Out With the Old

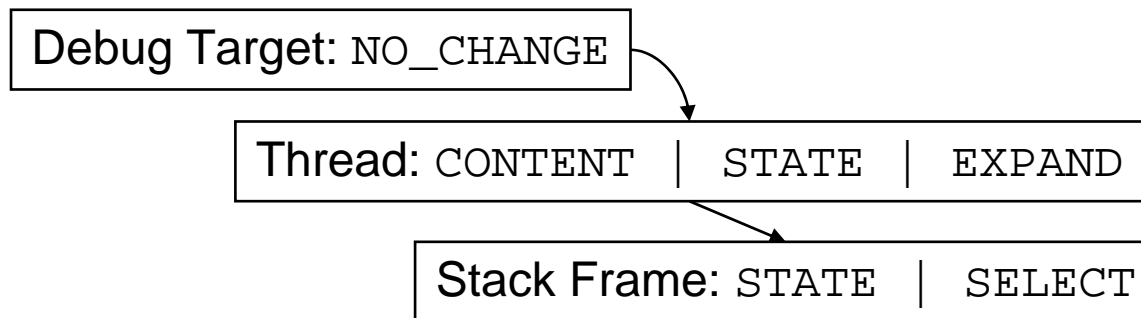


In With the New: Updating



Model Deltas

- Model deltas are similar to resource deltas
 - Describe incremental changes in a hierarchical model
 - Model deltas describe what changed, how it changed, and what action to perform on an element



Properties of the Interaction

- Viewer update is **extensible**
 - Extensibility provided by adapters
 - Interaction with the model is no longer defined by the platform, and no longer requires models to fire events
- Viewer update is **context sensitive**
 - The *presentation context* identifies the part requesting a model proxy, allowing for different updates in different views
- The model proxy is **insulated the from viewer implementation**
 - The *update policy* drives the in response to *deltas* which insulates the model from changes in the viewer implementation

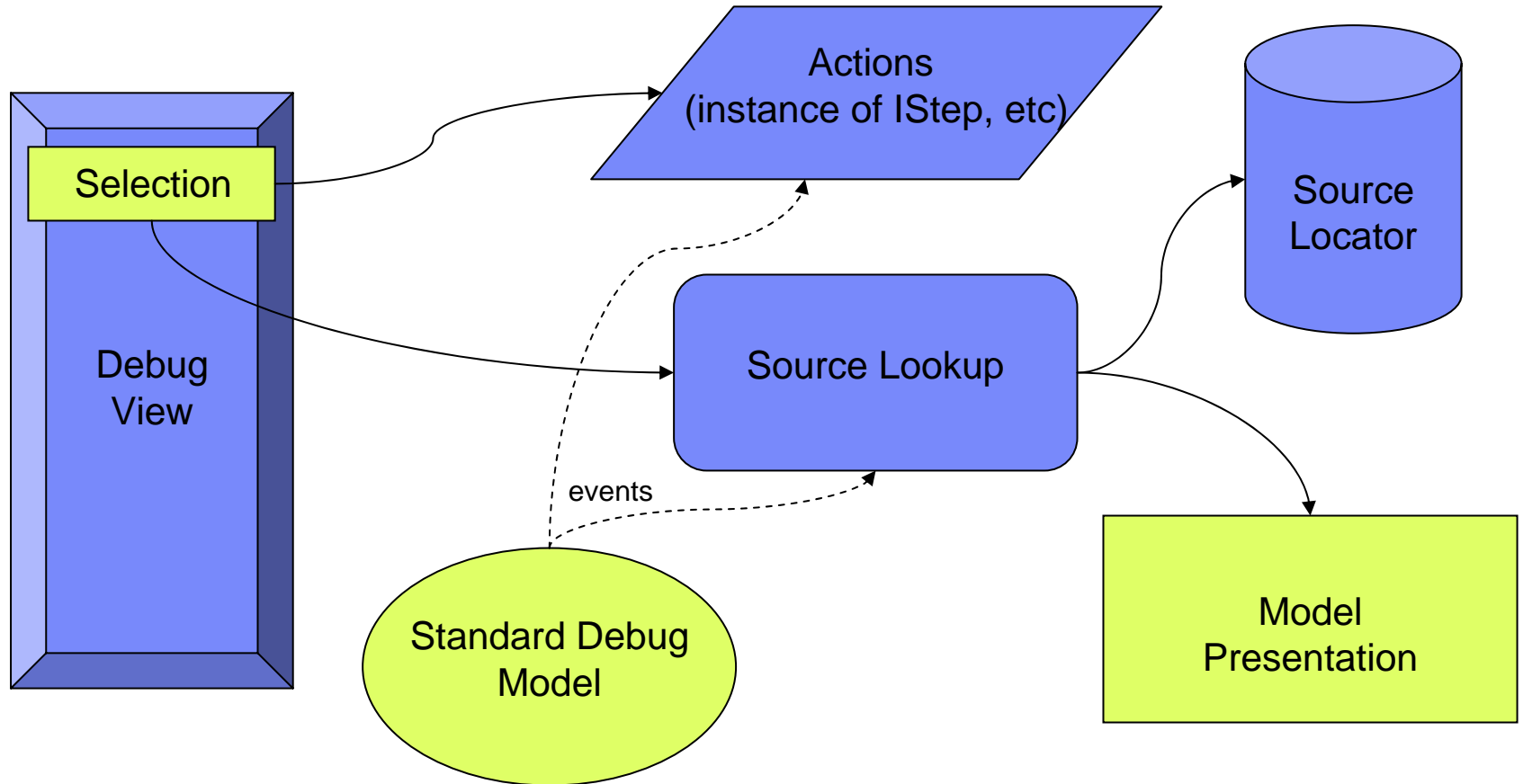
Result: Generic Model Driven Views

- The content, label, and model proxy adapters support generic views displaying generic models
 - The interfaces and interactions are not debug specific
 - The technology could be applied to other domains requiring asynchronous interactions
 - Technology could eventually be pushed down the food chain

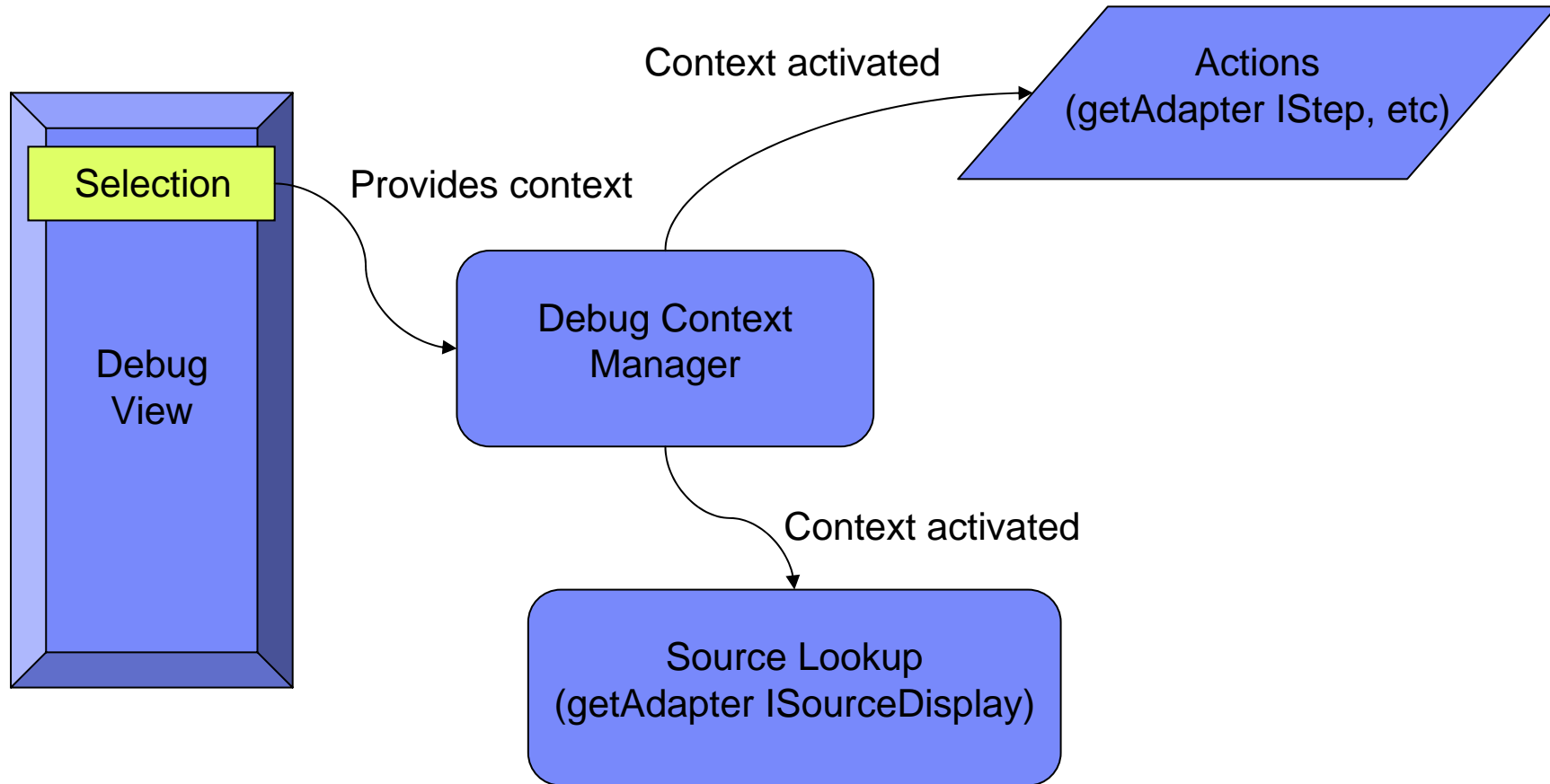
Challenge: Model Driven Interactions

- Requirements
 - A debugger implementation that does not conform to the standard model must still integrate with
 - the standard actions such as step, terminate, resume, etc.
 - source lookup
- In the old implementation, everything was tied to the selection in the debug view
 - Actions, input to variables view, source lookup, etc.

Out With the Old



In With the New



The Active Debug Context Drives the Debugger

- The Debug view is just an `IDebugContextProvider`
 - By default, the Variables view listens to the active context within its window, and displays content provided by the active context's content adapter for the variables view
 - Any context can be used to populate the variables view
 - A Variables view could be linked to a specific Debug view
- The Debug actions are `IDebugContextListener`'s
 - When the active context changes, the action asks the context for its associated capability adapter (`IStep`, `ITerminate`, etc.), and updates based on the current state of the context.
 - When an action is invoked, it disables until the next context change triggers its update

Properties of the Interaction

- The standard actions become **retargettable**
 - The actions work on the active context via operation adapters
- Source lookup becomes **pluggable**
 - Source lookup works on the active context's source display adapter rather than stack frames
- Provides **flexible view wiring**
 - The active context provides the input to the variables view so models can provide adapters to show content for any element
 - The active context drives expression evaluation in the expression view
 - The active context drives debug view management (automatic view opening/closing)

Some Community Testimony

“The content and label providers, and model proxy APIs are very well aligned with this design, and I haven't had any issues implementing them.”

“The integration between the old and new components is very smooth, and I can even have implementations of both working side by side talking to the same communication layer.”

- pawel.piech@windriver.com

A Tour of Other Enhancements

The memory view and launching

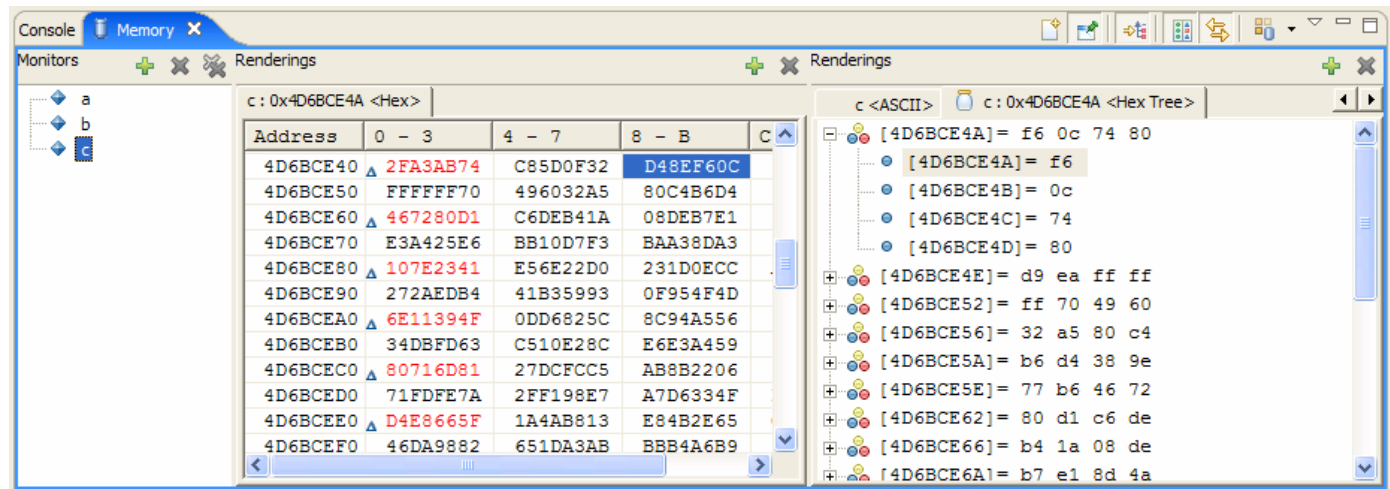
The Memory View

Introduction

- What does the Memory View do?
- The History
- The Design
- How can others reuse this technology?

What does a the Memory View Do?

- Show content of a memory monitor with memory renderings.
- Memory Renderings are UI entities for rendering the content of a memory monitor using any swt widget.



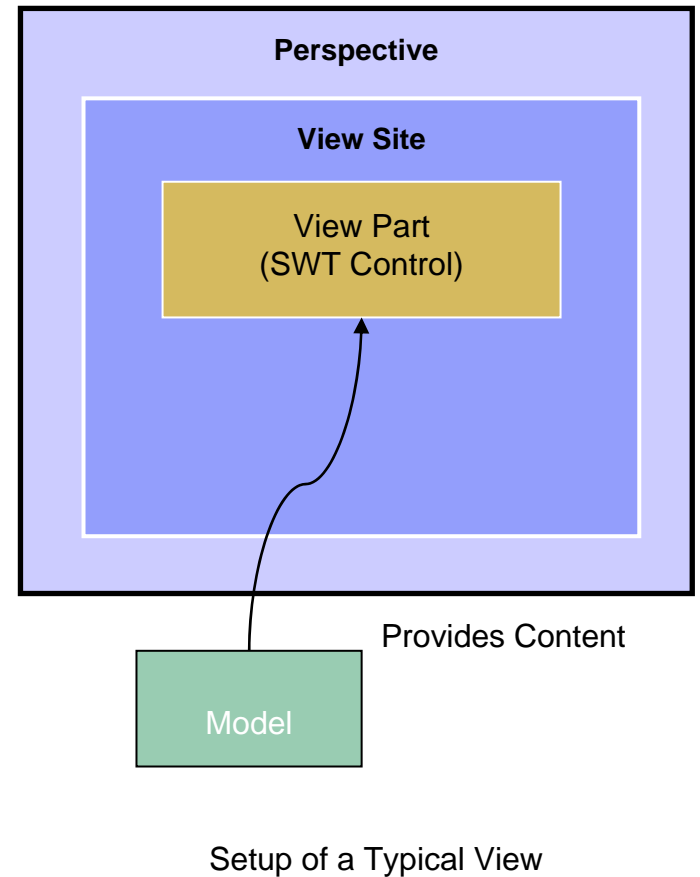
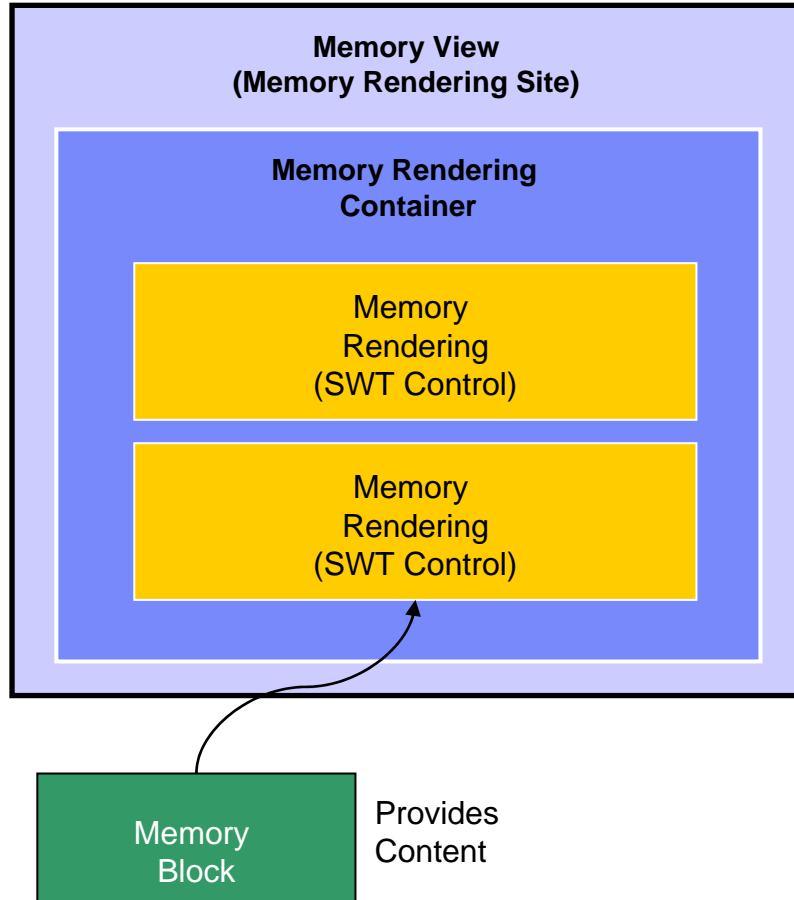
Back to not too long ago...

- Problem:
 - Platform provided APIs for creating memory blocks but had no view to show these debug elements.
 - Companies developed their own specialized view showing memory monitors.
 - IBM: Storage View
 - CDT: Memory View
 - Wind River: Memory View
 - and more...
- Platform needed to provide a standard view for showing memory.

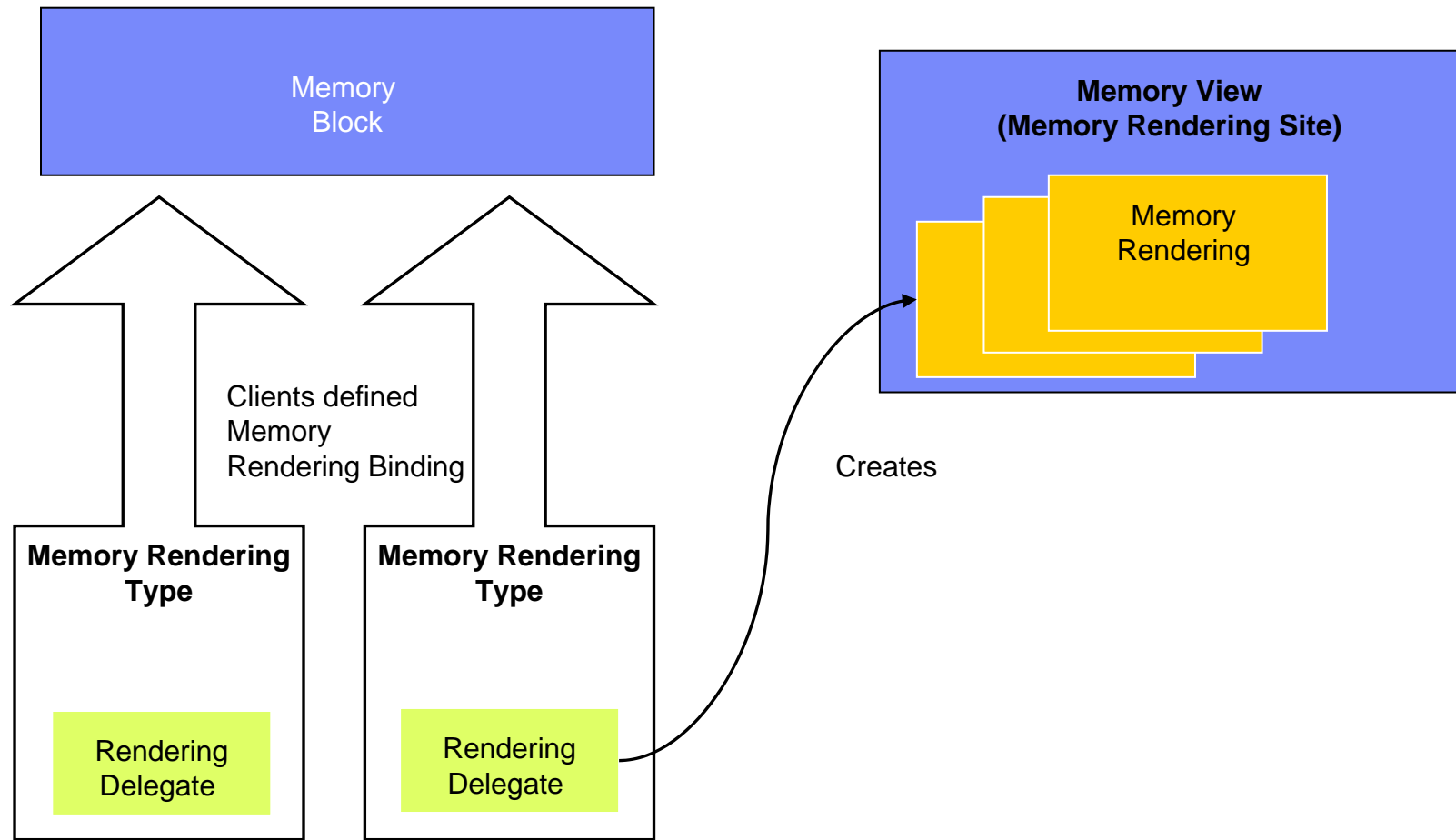
Requirements

- Extensible
 - Do not tie clients to any type of viewer. Clients decide how memory monitors should be displayed.
- Portable
 - The view showing memory monitors need to be portable in a way that it can be shown in views other than the Memory View
 - E.g. Showing memory in the Variables View's details pane.
- Easy to Adopt

View within a View



Memory Renderings and Memory Rendering Types



What is a memory rendering?

- Client may use any swt widget to show memory monitors.
- IMemoryRendering APIs are similar to a IViewPart API.
- Life Cycle:
 - `public void init(IMemoryRenderingContainer container, IMemoryBlock block);`
 - `public Control createControl(Composite parent);`
 - `public void dispose();`
- Easy to learn and implement. Just like writing a view!

But it's still too much code to write!

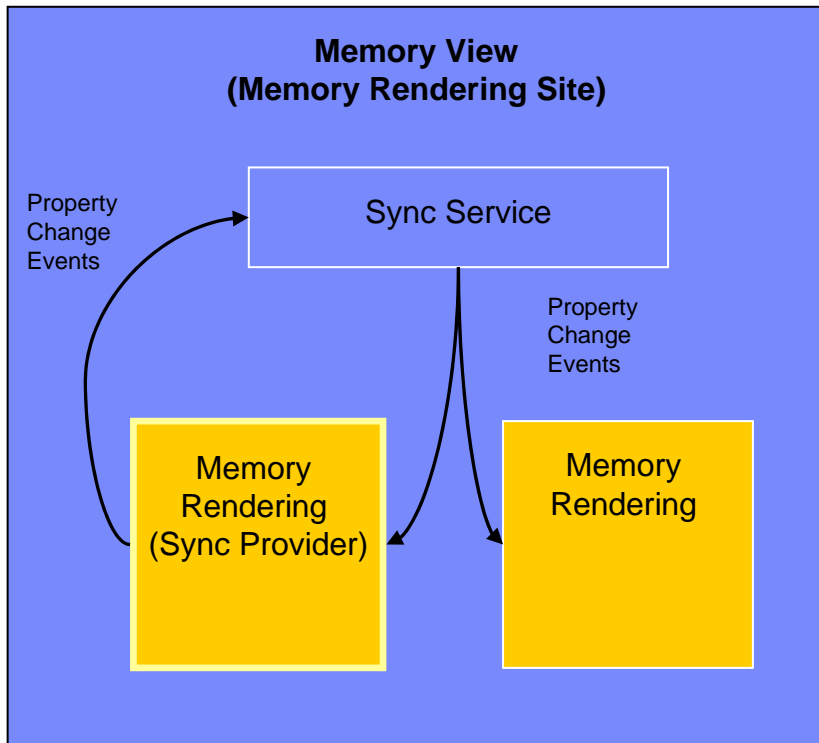
- Writing a view is difficult.
- Most clients want to see memory in a table.
 - Duplicated effort if we all have to write our own table renderings.
- Platform provides a set of most common renderings:
 - HEX, ASCII, Signed Integer and Unsigned Integer
- Default renderings not tied to any debugger. Client simply creates a memory rendering binding to reuse these renderings.

- To get the Memory View up and running:
 1. Add Memory Monitor Support.
 2. Bind memory block to existing renderings.... and you are done!

Synchronization between renderings

- Synchronization is needed to provide a better user experience.
 - User does not have to manually keep the renderings in sync.
- Challenges:
 - Renderings come and go – hard to keep track of what renderings are currently displayed and need to be synchronized.
- Solution:
 - Synchronization Service
 - Communication Hub between the renderings
 - A rendering only notifies the sync service about its changes. It does not need to be aware of other renderings.
 - A rendering only receives change event from a sync service. It does not need to know who to listen for change events from.

Synchronization



- Renderings define a set of properties that can be synchronized. (e.g. selected address, top visible address)
- When a property is changed the rendering fires a property change event.

Did we do what we set out to do?

- Extensible?
 - Memory Renderings allow clients have the freedom to show anything in the view. It's one step further than flexible hierarchy.
- Portable?
 - Memory Renderings allow clients to put any swt widget in a rendering. Any view that implements IMemoryRenderingSite can host a memory rendering.
- Easy to Adopt?
 - Memory Renderings APIs are similar to IViewPart APIs
 - Platform provided a set of rich-featured renderings for clients to reuse.
 - Separation of memory renderings to any debugger allow renderings to be contributed globally and be reused easily.
 - Getting the Memory View up and running is a 2 steps process.

How can others reuse this technology?

- Most views in Eclipse are tied to a specific viewer.
 - Variables View is tied to a tree viewer showing variables.
 - Navigator shows local file system using a tree viewer.
- Concept of renderings can make views more flexible.
 - Variables can be shown in a table tree.
 - Navigator can show local file system with icons.
- Clients can better define how their model should be represented.

How can others reuse this technology?

- Synchronization Service can be generalized for synchronizing views.
- Similar to selection service, but provides more flexibility:
 - Selection Service only allows clients to synchronize selection in a view. Does not allow views to synchronize other things or more than one thing. (e.g. scrolling, font settings, etc.)
 - Synchronization Service will give better performance.
 - Allow clients to define event filters.
 - Only relevant parties are notified. (i.e. renderings showing the same memory block.)

Launching Enhancements

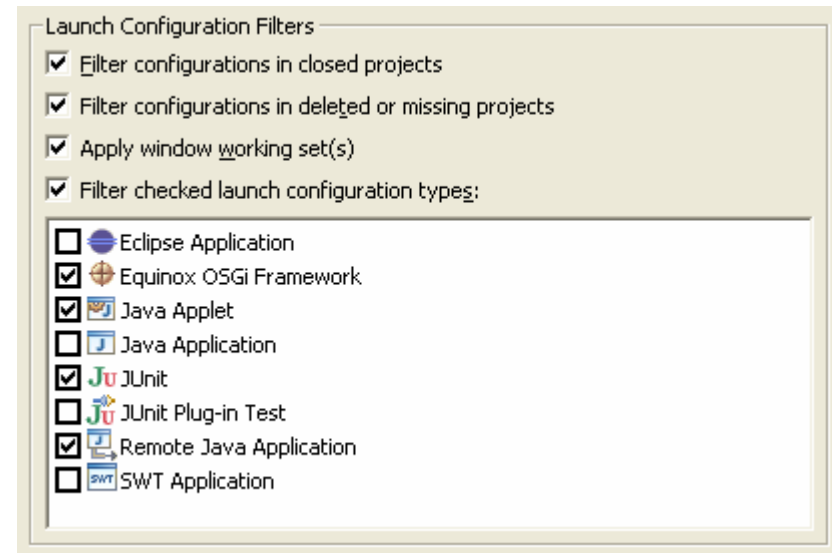
Launch Configuration Resource Mappings

- Support has been added to associate a set of resources with a launch configuration.
 - Allows the debug UI to perform resource based filtering and cleanup when a project is deleted
 - Clients that contribute a launch configuration type are responsible for maintaining the mapping
 - API has been added to set and get resources:

```
public IResource[] getMappedResources()  
public void setMappedResources(IResource[]  
resources);
```

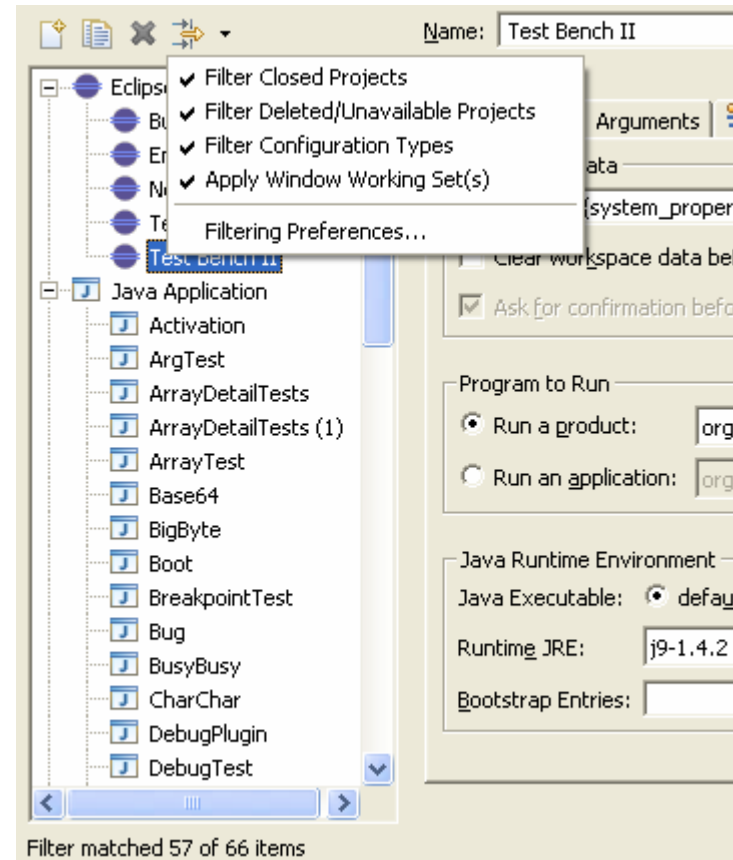
Launch Configuration Filters

- Launch configurations are filtered in the launch history and launch dialog based on user preferences
 - Closed projects
 - Unavailable projects
 - Window working sets
 - Specific configuration types



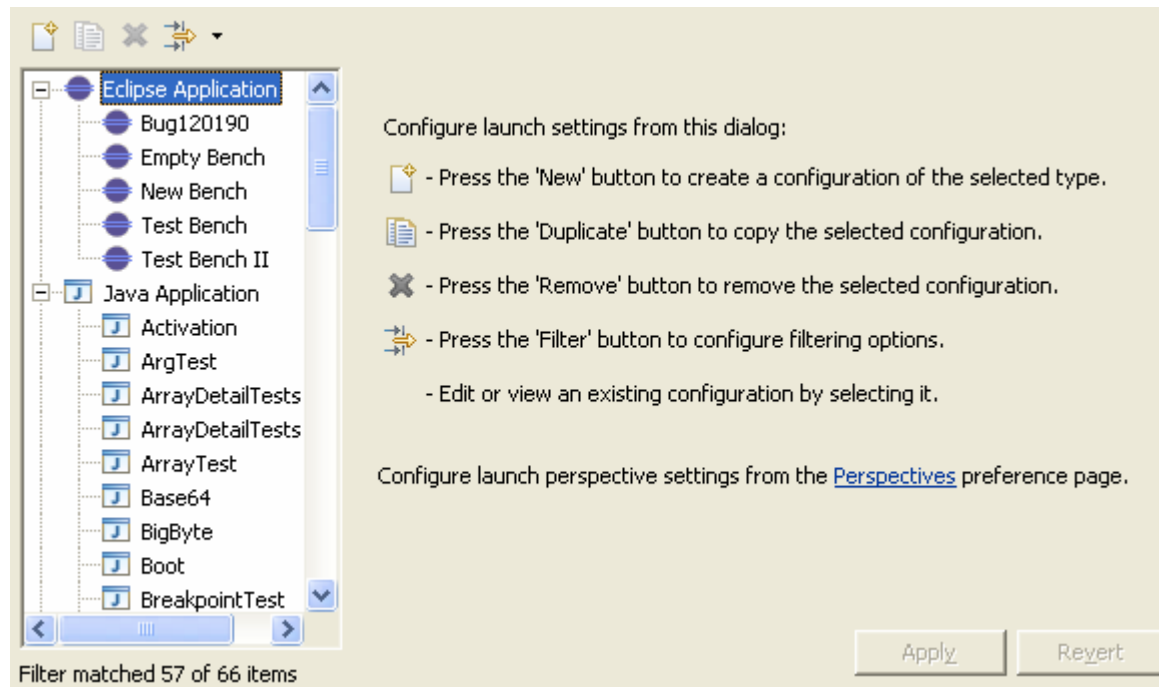
Launch Configuration Dialog

- A toolbar has been added to the launch dialog to create, delete, and copy configurations, as well as set filtering options.



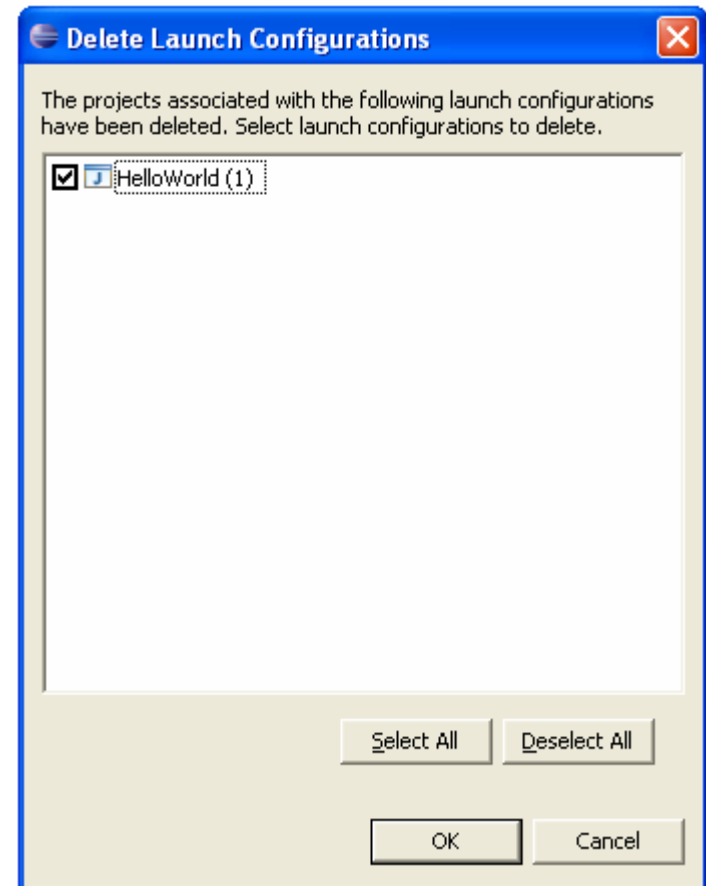
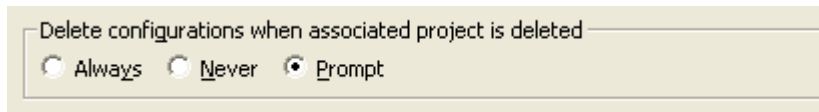
No More “Perspectives Tab”

- The perspective settings have been moved to the user preferences, and the dialog has instructions in its place.



Automatic Cleaning of Launch Configurations

- A user preference controls whether configurations are deleted when their associated project is deleted.



Launch Configuration Migration

- Support has been added to migrate launch configurations to support new tooling options
 - For example, the new resource mapping feature is a client responsibility and existing configurations must be migrated to leverage the new feature.
 - A migration delegate can be specified by launch configuration types
 - Identifies migration candidates
 - Migrates configurations
 - Users can selectively migrate configurations on the Launch Configurations preference page

Legal Notices

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.