Device Debugging Sub-project
Chicago Meeting
5/3-5/4
(document version: 1.0)

Attendees

Wind River
Doug Gaff – Boston, MA
Eugene Tarassov – Alameda, CA
Randy Rohrbach – Boston, MA
Ted Williams – Mendocino, CA
(Tomas Evensen, Rudi Frauenschuh, Michael Scharf - not attending)

IBM - Rational
Peter Nicholls
Alan Boxall
Darin Wright
(Douglas Schaefer - not attending)
(David Daoust - not attending)

QNX
Sebastien Marineau (cancellation)
Derrick Keefe
Mikhail Khodjaiants
Alain Magloire

TI
Andy Waterson
Martin Imrisek
Brian Cruickshank
Paul Gingrich
Dobrin Alexiev
(Annie Chen - not attending)
Chris Recoskie (phone)

ATI (Mentor)
Aaron Spear
George Clark

TimeSys
Gene Sally (not attending)

Intel
Susan Macchia (phone)

Original Agenda

See comments from various participants below

- Introductions – name, company, location, Eclipse-specific roles
- Presentation of Device Software Development Platform (Wind River's top level project proposal) and the Debugger Subproject

- Platform Debug views needs
    - current issues
    - wish list of desired changes
    - actions and contributions
- Platform Debug Interfaces and CDI Interfaces discussion
    - current issues with each
    - wish list of desired changes
    - actions and contributions

Homework prior to meeting:
- Review DSDP / Debugger proposals

<u>Alain Magloire</u>

Is it possible to have a change in the agenda:
I believe with prior talks, we passed this hurdle (CDI limitation and all) CDI has its "raison d'etre". But I would like to move things forward The proposal on the last conf. call was some sort of new interface based on the Platform Debug interfaces and the particular requirements to handle sync/async.
This need to be dealt with at much lower level and probably better to start with a clean sheet.

CDI will still be around for a time (some companies use this as the point of entry in there products) and it may still be around as an independent layer to all the GDB incantations in the open-source.

<u>Mikhail Khodjaiants</u>

There are CDT-specific extensions of Platform Debug with it's API (org.eclipse.cdt.debug.core and org.eclipse.cdt.debug.core.model packages) and UI components (org.eclipse.cdt.debug.ui plugin). It's a higher (than
CDI) level of abstraction. We are planning to clean it up and make public and would be interested to add this topic to agenda.
So, I would suggest to replace the "Platform Debug Interfaces and CDI Interfaces discussion" topic by the following two topics:
"Platform Debug Interfaces and CDT extensions"
"CDI Interfaces discussion"

<u>Eugene Tarassov</u>

1. Asynchronous lazy retrieval of data: icons, labels, children, etc. No Jobs!
2. Cancellation of pending data retrieval requests when they become outdated because of scrolling, selection change, etc.
3. Views should not depend on a rigid debug model structure, like IDebugTarget -> IThread -> IStackFrame. Views should not need to know that given debug model element is stack frame, instead every element should implement interfaces for retrieval of presentation data - icons, labels and children, and interfaces for supported actions, like IStep.
4. Interfaces, which assume particular application execution model or just assume too much about target, like IThread, IStackFrame, IRegister, etc., should be separated from views, redesigned and become optional (views should not need or use them). Those interfaces should be accessed through IAdaptable mechanism and be used only for interoperability between debuggers and other tools, which need target access.

5. Tree based views should look more like a table, e.g. should have resizable columns and allow in-place editing.

Randy Rohrbach

1. Interfaces need to support grouped requests for data. Single element data access can be extremely inefficient with the target. Coalescing algorithms are problematic.
2. We need to emphasize the ability for each view to be written to be totally extensible, allowing for the developer to be able to control/extend the rendering needed to work with the capability of the debug engine.
3. The same is true for the interfaces themselves, extensible with no rigid assumptions about data model (e.g. no stacks required).
4. We want to make the GUI/interfaces as data driven as possible so the debug engine can drive the show. Do we need to propose a generic configuration extraction/specification interface capability?
5. Need to emphasize the language neutrality of the interfaces. But also we should try and provide a standard for the language specific information/attributes interfaces.

Ted Williams
(I agree with Eugene's list, but I have a couple of comments on the details.)

> 1. Asynchronous lazy retrieval of data: icons, labels, children, etc. No Jobs!

I would like to see the async interface include a mechanism for associating keys with requests. Ideally, an Object.

```
myRegisterObject.getValue(AsyncRegisterRequestListener listener, Object key);
```

Matching the incoming data to requests would be simpler, especially since a Runnable (or sequent) could be passed as the key.

> 5. Tree based views should look more like a table, e.g. should have resizable columns and allow in-place editing.

...and should allow for display and edit of additional attributes (ability to add additional columns) based on the capabilities of the selected context.

Questions and additional discussion items from each group

QNX
- CDI vs. Platform
- Requirements for both interfaces

IBM
- Problems with current API's
- Where do fixes go?
- Concerned about pushing too much into platform

WR
- Present overview of WR Design to illustrate requirements
- Debug multi-languages (cross debugging): Ada, C, Java, Fortran
- "Pluggable" Debuggers and Toolsets

ATI
- Pluggable Toolsets
- Multicore Systems
- TM

TI
- Multicore target interactions (global breakpoints, sync run control, mixed asm)
- Evolution of CDT API's
- What do we have 2 sets of debug API's?

Presentation topics
- WR's debug implementation
- Darin's vision for the future

<u>Notes</u>

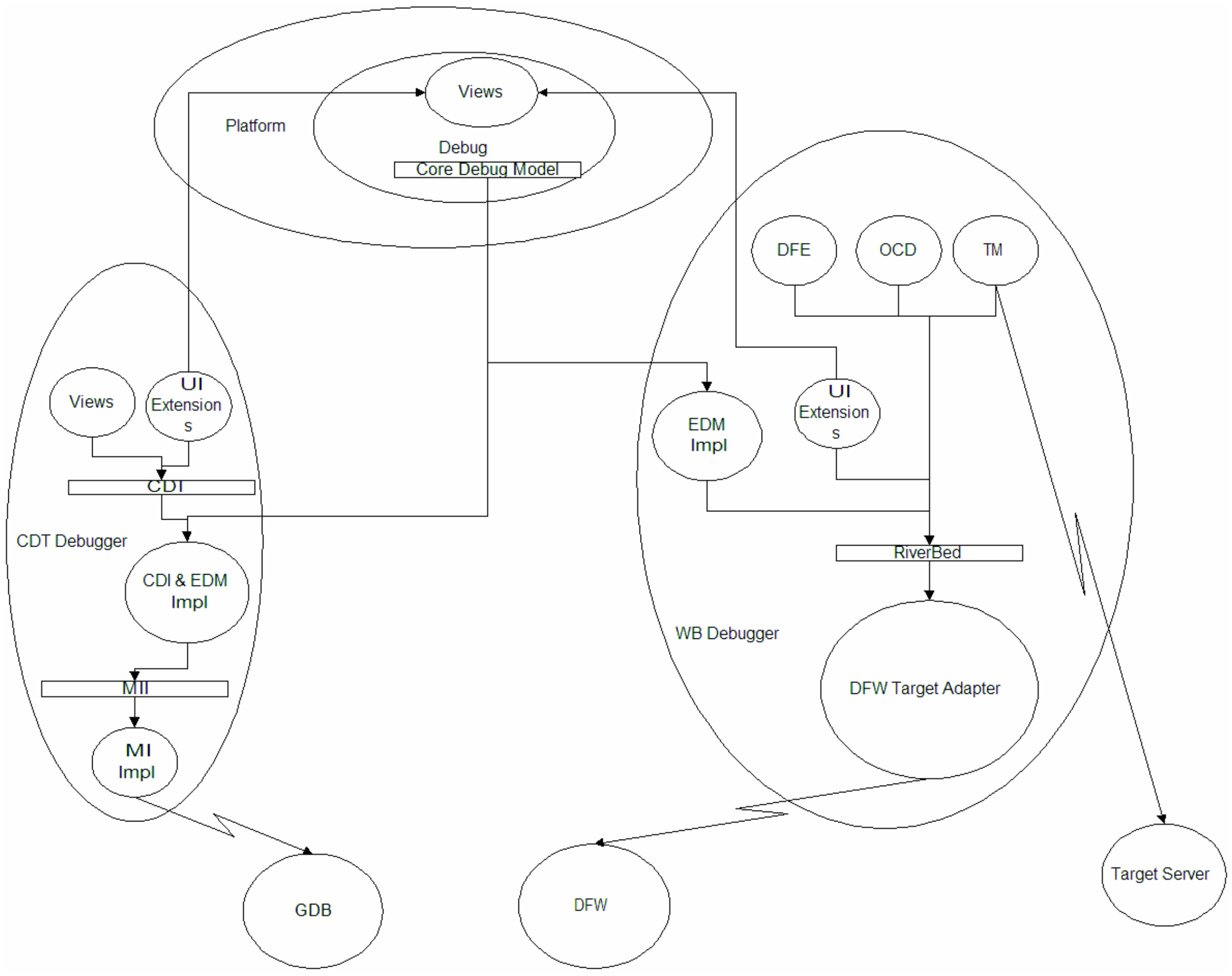
Why are there 2 debug interfaces?
- History was that platform interfaces weren't stable.
- Probably time to merge the two. CDI can go away and CDT is just an extension.

CDI:
```
Session
        Target
                Thread
                        Stack Frame
                                Local Variables
                Thread Local Storage
        CV
                Classes, registers, Memory, Signals, etc
```

CDI - replace it with PDI + extensions for a language bias
Darin wants to keep most language specific stuff out of PDI
There are scalability issues when multiple debuggers are in the picture

Darin - adapters for customization of content of debug view in 3.1
Other debug views will support this, too.

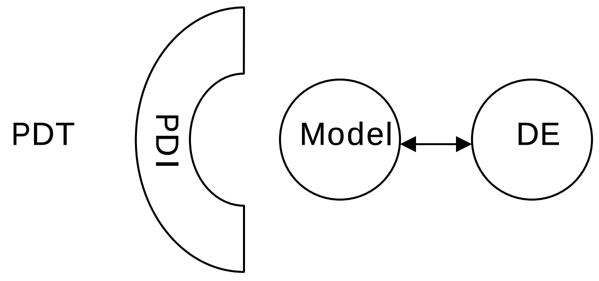
<u>WR - Eugene presentation of DFE views / expressions model</u>



Benefits

- Generic nodes vs. detailed interfaces with specific hierarchies
- No multithreading, queue
- Flow control (multiple requests made of the debugger), reordering, coalescing
- Cancellability

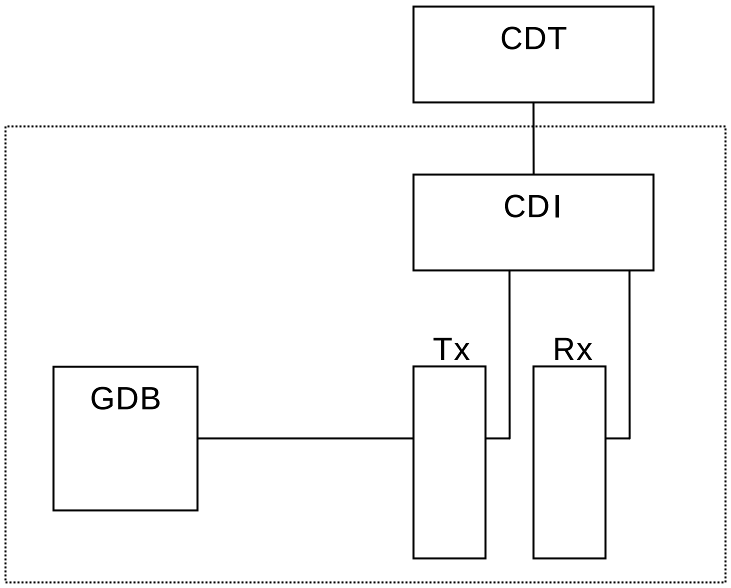
<u>IBM - Alan B</u>



PDI implementation to DE (Debug Engine)
Their internal debug model works on dealing with deltas between previous state and current state, so you don't request everything, you just find out what has changed.
The model determines its capabilities from the engine.

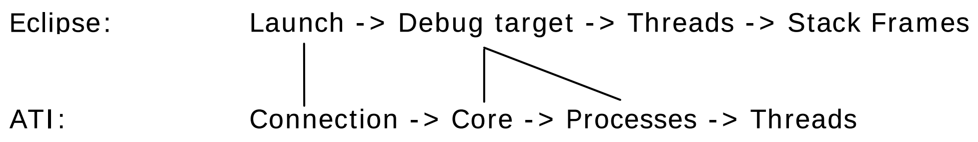
QNX - Alain M

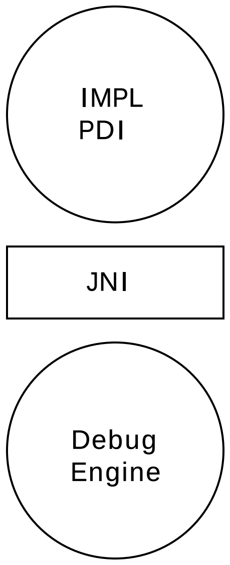
Goal was for something that the open source community could use.



GDB has limitations. Stack frame for example. They can't just get deltas, they much request all the information.

ATI - Aaron S
Xray and CodeLab debuggers are the historical debuggers.


Eclipse:
Launch -> Debug target -> Threads -> Stack Frames
ATI:
Connection -> Core -> Processes -> Threads



They worked hard to not touch the eclipse PDI code. They've created a bunch of their own views, since the platform views don't have enough functionality. These views access internal interfaces and not the PDI.

Architecturally, they are similar to Wind River in the data retrieval method. They are asynchronous. They have seen good scalability, too. The debug view seems to make a lot of unnecessary requests that impact performance (like expansion of the entire call stack.)

Serialization is done in the debug engine, but this is mostly a limitation of the debug connection (like a 3rd party JTAG tool).

<u>TI - Dobrin A (Eclipse product)</u>

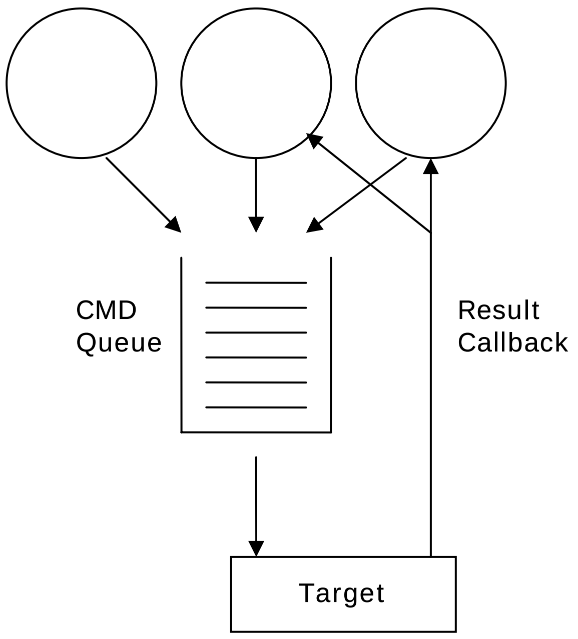
Code Composer Studio - existing product.
First experience with Eclipse - small group to do GUI and GDB interface - been working on it 9 months. Done for a specific DSP.
Code Composer Essentials is the Eclipse product.

"Active Project" - active debug, configuration - willing to contribute back to CDT
(See slides for CDT user issues). They configure target connection via the project settings.

They create launch sequences behind the scenes to hide the gdb facets of debugging.

They modified core Eclipse

<u>TI - Paul G (Code Composer Studio)</u>



This architectures is 10 years old. Was invented prior to multithreading (Win 3.1 days).
Hierarchy:

Boards -> CPU (core) -> Tasks

Separate queue for each CPU and each Task. A polling loop processed each queue in sequence.

They are considering rearchitecting this inside of Eclipse.

<u>Platform Debug Views - Darin</u>

Differences between 3.0 and 3.1. Async view update in background.
(drawing I didn't capture showing the background jobs for view update)

One note about this drawing is that the debug implementation can still serialize the multiple requests. The point is that the GUI doesn't block.

These changes still honor the API's, although they are exposing synchronization issues in the platform, and will likely expose synchronization issues in implementations.

<u>API Discussion</u>
What should the API's do that they don't do now?
- Cancellability - requires an asynchronous API

<u>Day 2</u>

Agenda
1. Platform Debug API

2. Debug View Requirements
3. Sharing some common implementation code (expressions, symbol table, program loader, …)
4. List from yesterday
5. Next steps

Platform Debug API needs

- Non-blocking UI with ability to do next step without waiting for views to update. (Ability to step as fast as the target can)
- More generic interfaces, instead of a fixed hierarchy
- Flow control

(Additional API discussion that didn't reach conclusion.)

Update Policies

- Update only what is visible
- Update nothing
- Update all
- Periodic update
- Update no more frequently than x seconds
- Target state change update
- Delay update x seconds after suspend

<u>View requirements</u>

Table for who uses what from the Platform and CDT:

Debug View	**Wind River**	**ATI**	**IBM**	**TI**
Debug	Use	Use	Use	Use
Variables	Custom – Locals	Use	Use	Use
Registers	Custom	Custom	Use	Use
Memory	Custom	Custom	Use	Use
Breakpoints	Use	Use	Use	Use
Expressions	Custom – Watch	Custom	Custom	Use
Debug Console	Custom – Output, OCD Shell, Host Shell		Use (contributing)	
Editor	Custom			Use
Modules (CDT)			Custom	
Signals (CDT)				
Disasm (CDT)	Custom		Custom	Use
	Threads			
	Stack			
	JTAG views	Custom		
	Kernel objects	Custom		
	Symbol Browsing	Custom		
	Terminal	Custom		

Note that TI used CDT now, but they are looking to add more functionality.

General view requirements

- Table trees for all data views
- Drag and drop between views
- For multiple copies of a view, user should be able to customize a view name

- Tooltips or hover support

Editor
- Extensions desired for disassembly content

Debug View
- Modify hierarchy (3.1) using adapters allows hiding of actions for Java debug
- Selection model customization
- Ability to do run control without debug view open
- Option to not close a fast view on a run control event
- Code that opens an editor should move outside of debug view
- Table tree instead of a tree control
- Stack is refreshed for all threads when a process stops
- Scalability when there are a very large number of threads
    - Performance of SWT lazy update
    - Presentation of lots of info (may need filtering)

Locals View
- Scroll position is not preserved (bug)
- FYI: "Logical Structures" to apply a different view of a data structure (3.0). These are tied to a variable and are application to multiple debug views.
- Pluggable hierarchy – is this needed?
- Pluggable details area?
- Table tree control
- Custom editors for variables

Register View
- Table tree control
- Drag & Drop registers between groups (this may belong only in the data model)
- Similar requirements to locals

Memory View
- Address space – should be a block extension
- Need background jobs for data retrieval

Breakpoints
- Control is out of sync with target for some amount of time until the breakpoint is planted/removed/disabled
- Breakpoint scope (belongs in model). Note: 3.1 has breakpoint categorization with filtering.
- Breakpoints associated with launches
- Default properties content? Or at least a generic way of dealing with breakpoint properties.
- Table tree control.

Expressions
- Table tree control.
- Assign expressions
- Same requirements as locals (rendering, details pane, etc)

Debug Console
- Current functionality to be contributed
    - Used to send commands to debug engine

    - History by target
    - Commands are line-oriented
  - Needs to be refactored to use the Generic console in 3.1
  - Playback and record functionality
  - Parsers
  - Character I/O
  - Escape sequences

<u>Contribution Ideas</u>

- Darin feels that these changes are too bug for 3.2, and that they are of more interest to device software. Probably 4.0.
- We need to agree on the API's and then folks need to contribute and plan to maintain.

Does is make sense to branch the platform debug code to start a parallel track and then merge in 4.0? Would create internal API's in 3.2 that would allow debug views to work with both the old and new API's. Eugene would like to use the same view with a replacement renderer to test the new views.

AI: Doug to talk to Rudi about Target Management - ATI, IBM are interested. They are wondering if it goes beyond the scope of device debugging. Also, how does it relate to MontaVista's proposals and the parallel tools project proposals.

<u>Next Steps</u>

- Darin to digest
- API discussion in Toronto in July (first or 3rd week)
- ATI and TI - can support with 1 or 2 engineers
- IBM - has one resource on memory

Command Language Debug (LDT) - BEA
Parallel Tools guys interested

AI: Michael Scharf participation in LDT
AI: Is LDT interested

<u>Multiple context debug</u>
Wind River uses color views now, where a color is associated with a context, but they are hard to get to. You need to setup everything on a perspective and leave it that way.

Now that 3.0 supports multiple views, how can we use this?

Ideas:

- Clone a view and lock to one or more contexts
- Clone a perspective and lock to one or more contexts
- Add decorators / colors to debug view to associate with views

- Pin a view to the current debug selection
- Make a new selection and clone/pin to that

Our initial goal here is to focus on a handful of contexts at a time. Massive parallel computing and the views & sync run control required for this are somewhat outside of the scope.

The way to create new views and assign them to contexts belongs in the platform.