Eclipse DSDP-TM
Target Connection Adapter (TCA)
Design Proposal

Peter Lachner, Intel
Rev 1.0.1, August, 2005

Abstract:

The following paper outlines a design for an open and flexible communication interface in Eclipse. We introduce target connection adapters (TCAs) or so called 'connectors' to establish a link between a host application and a target system. The design allows maximum flexibility regarding the type of the connection by abstracting the actual implementation of a connection from the logical link to the remote system. This design allows applications using own or foreign 3rd party connectors to communicate with any target devices.

1. Introduction

The target connection adapters supply an infrastructure which allows the management of communication channels between applications running inside the Eclipse framework and an arbitrary amount of different target systems. The communication framework shall not make any assumptions about nor introduce restrictions on the way how this communication is established. For this purpose the framework uses so called connectors.

A connector is an entity which acts as the communication agent between an application and its target system. In this context, an application can be a wide range of different programs as

- an interactive console terminal emulation
- a file transfer and download utility
- a SW debugger
- a browser for OS information

A target system can be anything like

- an application on the same host
- simulation tool
- a remote computer
- an embedded device

To support different use cases a target system might have several connectors assigned to it. Several connectors for the same target may be active at the same time.

The remote connection view is the central place for managing connectors. It shows all defined connections between applications and remote systems. It also allows creating new links by either using an already registered connector or adding new connectors to the system. Creating a link means to assign a connector to a particular target and then attach this connector to an application.

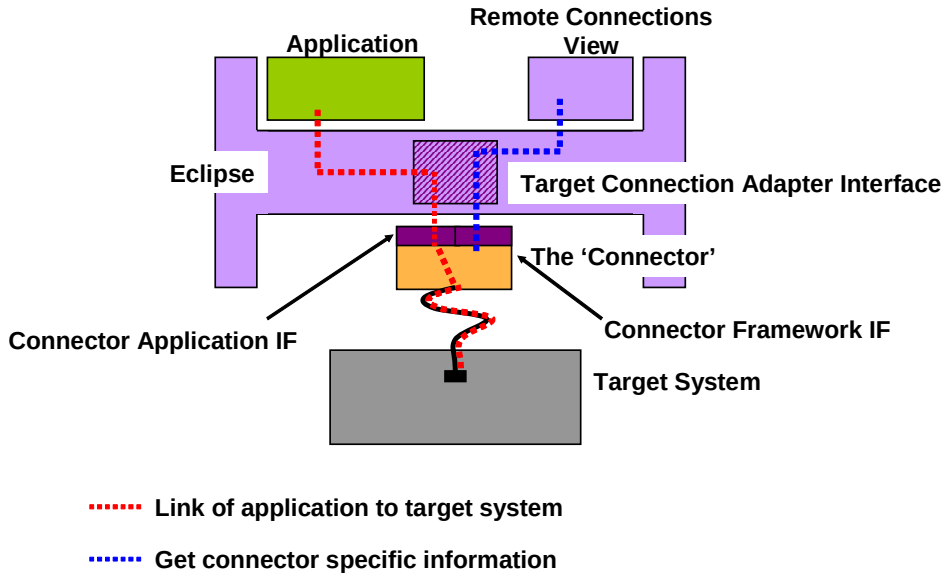
2. Connectors

There are no restrictions to the functionality supplied by a connector implied by the Eclipse framework. Suppliers of components and applications in the Eclipse framework must be able to exploit all features and functions of their own application and/or remote system software. On the other hand the development and supply of a connector must be possible to be independent from a host application, intended to be using this connector.

A connector is a runtime installable component (plug-in) into Eclipse. It contains

- the implementation of the required interfaces to the framework to control and manage the connector
- the implementation of the required interfaces to the application to communicate with the target system
- the implementation of the communication to the target system
- the implementation to store/retrieve dynamic information
- the implementation of the detailed views for the general remote connection Eclipse view (blue dotted line picture 2.1)

The picture below shows the individual components.

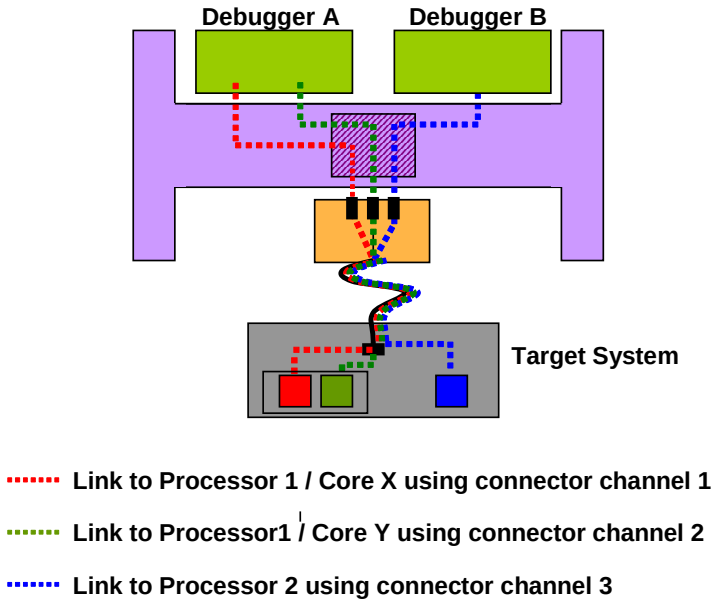


Picture 2.1

A connector contains static and dynamic information. Static information describes the capabilities of the connector. Dynamic information contains all data when attaching a connector to a particular target.

Connectors might supply multiple channels over one connection at the same time. This shall allow sharing one physical connection between several parties. An example is a JTAG connection to a target, which contains several CPU cores linked into one JTAG scan chain. This offers the ability to have even different applications using different channels of same connector to access their particular processor core.

The picture below shows, how in a heterogeneous multicore system 2 debugger applications can connect to the target system sharing one connector. Other configurations using 2 connectors one for each debugger or using 3 connectors, one for each processor shall be possible as well



Picture 2.2

In order to serve this wide range of different models, connectors have the following classes of information

- connector properties
- connection information

2.1 Connector properties

A connector property describes the class of service or operation which can be supplied to the application. The framework defines a list of specific property classes to allow applications to connect to foreign connectors.

There are a set of defined property class and protocol values (names), to allow applications to connect to foreign connectors. Applications can use this information to check if a connector is suitable, or even to search for a proper connector in a repository of connectors using connection browsers and wizards.

It is permissible to define own, proprietary values for channel class and protocol, but this would of course limit the ability to use and leverage connectors supplied by other parties.

Connectors can be chained to build hierarchies. This would allow for example to stack on top of a dumb JTAG connector a GDB protocol speaking debug connector.

The table below shows connector properties with proposed pre-defined values.

Connector Properties			
Property Name		**Description**	
Connector ID		Name of connector	
Description		Connector description	
Version		Version information with major.minor.patch notation.	
Channels		Number of different communication channels supplied by this connector	
For each channel	Channel Class	The class of service this connector channel can supply.	
		Value	**Description**
		"debug"	Support services to allow debugging of programs
		"data-transfer"	Up-/download, file transfer
		"file-system"	Supply directory services
		"console"	A remote console terminal
		"OS-info-*xxx*"	Access to *xxx* OS elements 1)
		"FLASH-programming"	Allows programming of FLASH devices
	Channel Protocol	What SW protocol is used by this channel	
		Value	**Description**
		"GDB"	Uses GDB protocol
		"ftp"	FTP file transfer protocol
	Target Architecture	Processor architecture this connector supports	
		Value	**Description**
		<processor-type>	Processor type 2)
		"any"	Architecture specification not needed or not applicable
	Share Channel	Can this channel can be shared between several applications and or stacked connectors	
Link Type		Type of connection to target	
		Value	**Description**
		"JTAG"	JTAG based connection
		"USB"	USB connection
Multiple instances		Allow multiple instances of this connector active at the same time with the same target. E.g. multiple terminal server consoles using telnet.	
Non-terminating		This connector only connects to another connector 3)	

Table 2.1

1) Due to the fact that the different OS's have a very broad variety of different elements as task, threads, events etc, it is expected that we probably wouldn't see an 'OS-info-any' connector. But for an OS vendor, it might be attractive to supply a connector perfectly suited for his OS to be used then by 3rd party SW vendors for their applications.
2) This field can contain a list of values with wildcards e.g. "ARM, ARM10" or "PXA27*" etc.
3) Only terminating type connectors can be talk to a target system.

Of course not all combinations make sense like class "debug" with protocol "ftp".

It should be mentioned, that the class characterizes a specific kind of service. The exact description on what particular functionality can be obtained from a channel is implicitly defined by the specified protocol. An application can choose to use several channels or connectors at the same time to join their features. On the other hand, a debug protocol might be able to support console-I/O as well. Nevertheless the primary use-case for such a connector would be of course debugging. It also allows to group connectors when shown in browsers. It makes no sense to list file-system class connectors in a connection setup view of a debugger.

An application calls the remote connection view to select a target connection. The framework supplies a connector instance ID, which can be stored by the applications persistent storage for future sessions.

2.2 Connection information

The connection information contains all information in order to establish a real connection. The content is not defined by the framework, since it is not predictable what kind of data must be held. Typical data are IP-address, port numbers, connection speed, etc.

3. Interfaces

There are 2 classes of interfaces involved. One is to manage connectors within the Eclipse framework. The other is used by applications to communicate with the remote system.

3.1 Connector Framework Interface

This interface is used to manage a connector inside Eclipse. The interface consists of functions to

- instantiate a connector template to attach to a particular target system.
- supply static and dynamic information for display in remote connector views.
- supply a GUI to modify dynamic data configuring the connector. Those methods are called from the remote connection view when selecting an 'edit' action.

3.2 Connector Application Interface

Each connector supplies the same functional interface to an application. This interface consist of functionality which allows to

- open the connector to establish a link to the target system
- close the connector to shut down the link
- write data to the target
- read data from the target
- connector control functions to configure the link

Operations shall be allowed to be synchronous or asynchronous. Asynchronous communication can be achieved by active pooling for data, or by establishing call-back functions via the control function.

The use of the Eclipse Communication Framework as the interface used by applications and connectors is considered. More investigation needs to be done, but at this stage, this does not affect the fundamental design idea of 'connectors'.

Things we need to consider here is, that we need to allow foreign applications (in this context for example a debugger written in C or C++) to use the Eclipse framework to utilize connectors, which are not written in JAVA as well.