Simple High Performance Transport - SHPT (Draft)

Revision 0.4c

March 23, 1998

Shigeru Ueda

Takashi Isoda

Akihiro Shimura

Contact e-mail address: oid3-1394@pure.cpdc.canon.co.jp

CANON INC.

5. Simple High Performance Transport(SHPT) Model (informative)

Note 1: This document focuses how the *unordered model* defined by the SBP-2 provides efficient bi-directional communication. This document also describes a simple method to manage the linked list of ORB in the initiator that utilizes the *unordered model*.

Note 2: Multiple service channels are assumed to be provided via utilizing multiple WRITE/READ queue pairs or some other means. Extended reconnect time out is also assumed to be provided via the mechanism introduced by the imaging profile or some other means.

Note 3: The command and status related to device control are not included in this document to focus on the bi-directional communication.

Simple High Performance Transport (SHPT) defines a small command set and a behavior model on top of the Serial Bus Protocol 2 (SBP-2). The SBP-2 is a transport protocol defined for IEEE Std 1394-1995, Standard for a High Performance Serial Bus. The SHPT provides full duplex communication capability between an initiator device and a target device.

This clause describes components of the SHPT model. In addition to the information in this clause, users of this document should also be familiar with the SBP-2 and its normative references.

The SHPT uses the data exchange mechanism provided by the SBP-2. The command block ORB (operation request block) works as a data transfer request for both direction as specified by the SBP-2. The SHPT introduces a queuing model on the target to queue those requests. In order to achieve full duplex communication, the SHPT utilizes the *unordered model* defined by the SBP-2, and controls the flow of those requests by using each queue. The unsolicited status defined by the SBP-2 is used as a request indication for an asynchronous data transfer from the target to the initiator.

The SHPT provides full duplex communication capability over single login with small additional resources by adopting the flow control based on the requests rather than on the transporting data itself. The benefits of the SBP-2 like high performance and low overhead are achieved via the features of SBP-2 like shared memory model and listed execution. The SHPT also inherits these benefits by relaxing the synchronization between both ends with the queuing model.

5.1 Target Model

Figure 1 illustrates an example block diagram of a command block agent for the SHPT target. The command block agent contains one command fetch agent, two command pre-fetch queues, called **WRITE queue** and **READ queue**, and two execution agents, called **WRITE execution agent** and

READ execution agent connected to the **WRITE queue** and the **READ queue** respectively.

The command fetch agent fetches the normal command block ORB's in order. When the command fetch agent fetches the normal command block ORB, the command fetch agent examines the parameter specified in the *command_block* field of the command block ORB. The fetch agent dispatches the command block ORB to either of the **WRITE queue** or **READ queue** according to the parameter. All WRITE commands are dispatched to the **WRITE queue**, and all READ commands are dispatched to the **READ queue**. The **WRITE execution agent** and **READ execution agent** execute the commands queued in the **WRITE queue** and **READ queue** respectively.

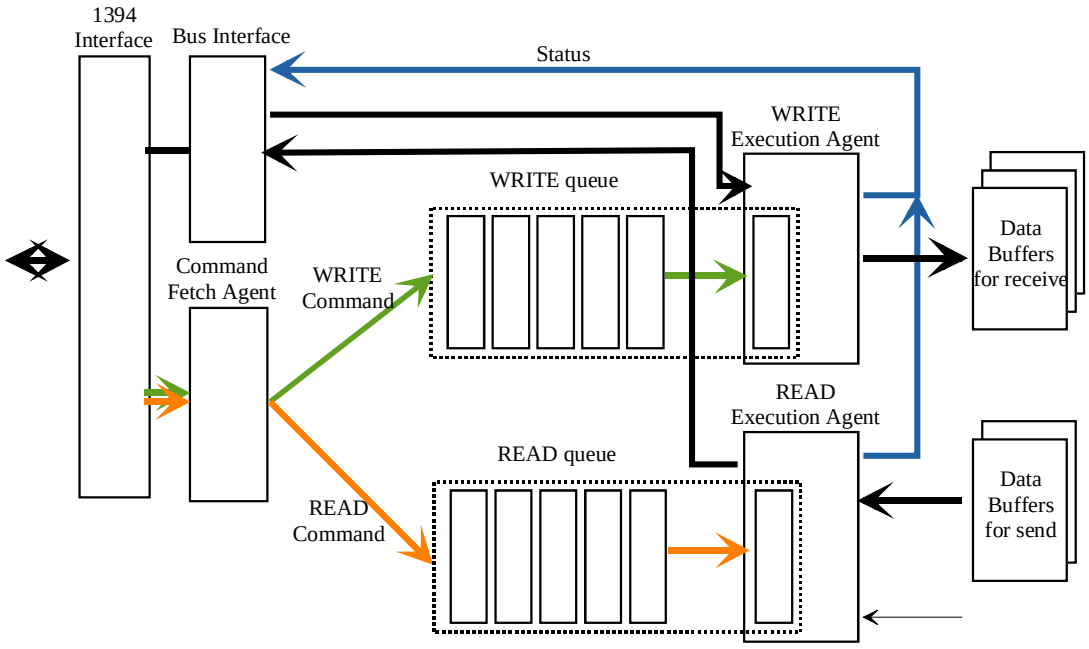

Figure 1 - Target Model

Each execution agent executes the dispatched command in the connected queue in order and independently of other agent. Each execution agent executes the data transfer associated with the command according to the parameters specified in the command.

The target stores a status block in the initiator's memory according to the value of the *notify* bit of the command block ORB after executing the command as specified by the SBP-2. Each execution agent shall store status_block in order within each execution agent.

The fetch agent (and execution agents) shall not refer to the *next_ORB* field of the ORB's in the initiator's memory that is already fetched and already in either pre-fetch queues except for the ORB contains null *next_ORB* pointer.

Note: The *next_ORB* field of the backward ORB's in the initiator's memory may not be valid pointer any longer since the pointed ORB may already be completed by the unordered execution.

When the target has data to be sent to the initiator and no READ command is available from the initiator, the target may store a **data available status** in the initiator memory. This status block may be either a normal status_block or an unsolicited status defined by the SBP-2. This status may be used as an data indication from the target by the initiator.

5.2 Initiator Model

The initiator of the SHPT has two **i/o request queues** as illustrated below.

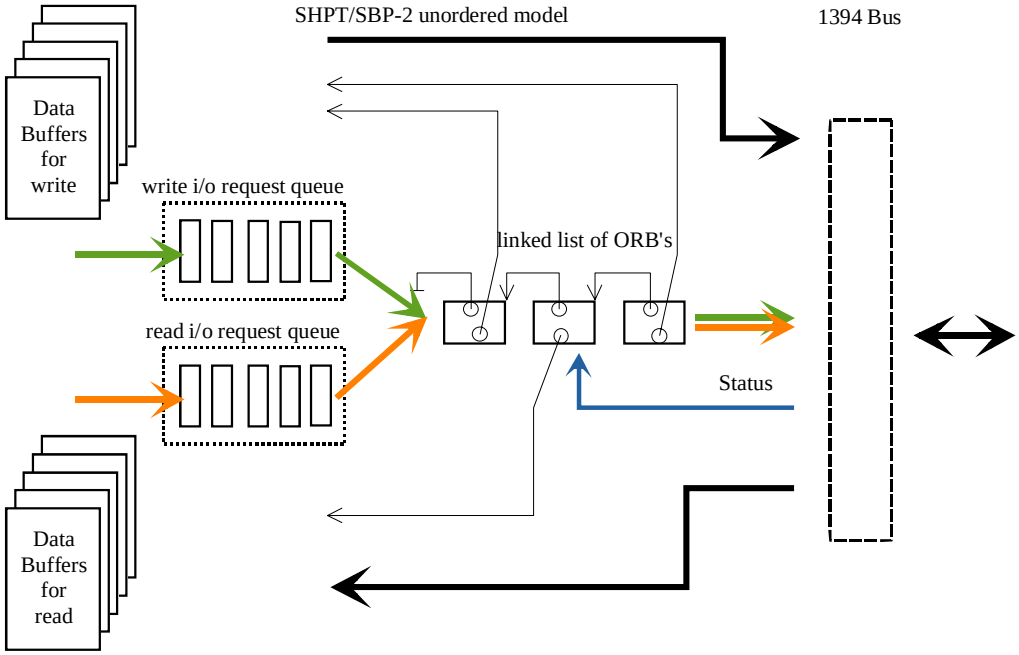

Figure 2 - Initiator Model

The initiator manages a constraint on appending a new **task** to a current **task set**.

The **WRITE queue** and **READ queue** in the target queue the command ORB's destined to the **WRITE execution agent** and **READ execution agent** respectively. The initiator restricts to append a new **task** destined to the each execution agent in the manner that the number of the commands destined to each execution agent in the **task set** does not exceed the available depth of each queue in the target.

In order to manage this constraint, the initiator retrieves the depth of the each queue from the target before starting a communication.

The initiator creates a command that specifies the **WRITE execution agent** as a destination in case of the data transfer from the initiator to the target. The initiator creates a command that specifies the **READ**

execution agent as a destination in case of the data transfer from the target to the initiator.

The initiator becomes aware that the target has consumed the content of each queue by receiving the status block specified by the SBP-2 corresponding to the command destined to each queue.

The initiator may free the completed ORB indicated by the status block, and complete i/o request in the head of the **write i/o request queue** or the **read i/o request queue** depending on the information in the status and remove the i/o request from the queue.

Note: The initiator does not need to update the *next_ORB* field of the ORB pointing the completed ORB in the current **task set**, since the target never refers the field retroactively.

5.3 Error Recovery

The initiator may detect that the target has aborted the execution of a certain task and stopped processing of succeeding tasks in the list via status block or agent state register. When the initiator detects this case, the initiator shall discard all ORB's in the current task set and re-initiate fetch agent with recreated linked list of ORB from the contents of **write i/o request queue** and **read i/o request queue**. The initiator shall maintain relationship between i/o request in each queue and corresponding sequence identifier. The initiator shall also maintain the contents of the buffer associated with each request.

The target shall be responsible to prohibit to duplicate processing of the content of each i/o request. In order to do this, the target maintains the sequence identifier and buffer offset currently processing. After the target aborted the execution of a certain task and re-initiated by the initiator, the target shall examine the sequence identifier in new ORB. If the target finds from the sequence identifier that the request is already executed, the target may complete the request without execution. If the target finds from the sequence identifier that the request was processed intermediately, the target may continue processing from the point indicated by the buffer offset.