A Packet Utilisation Standard (PUS) Model

This is a simulation model of the European Space Agency (ESA) Packet Utilisation Standard (PUS) in an operational environment linking a satellite with ground stations.

The Packet Utilisation Standard

The PUS is linked to the ESA Packet Telemetry and Telecommand Standards, which address the end-to-end transport of telemetry and telecommand data between user applications on the ground and Application Processes onboard the satellite, and the intermediate transfer of these data through the different elements of the ground and space segments. The Packet Utilisation Standard complements the Packet Telemetry and Telecommand Standards by defining the Application-Level interface between ground and space, in order to satisfy the requirements of electrical integration and testing and flight operations. PUS relates to the utilisation of telecommand and telemetry source packets for the purpose of monitoring and remote control of subsystems and payloads. PUS is also used to manage onboard monitoring and control functions for manned missions.

PUS packets are used in conjunction with specific Packet Utilisation Standard Services, which correspond to particular monitoring and control functions.

Service specifications define:

- what can be requested of the Service;
- what can be issued by the Service to notify the user of the success or failure of this request and to report events detected during the execution of the Service activities;
- what internal activities the Service should perform to process a request or to detect or process events related to the Service

PUS packets are used to control and monitor these services via a well-defined set of interactions between the provider of the Service (i.e. an onboard Application Process) and the user of the Service (i.e. a ground system).

The MLDesigner PUS Model

The MLDesigner PUS model is comprised of three major modules

- ground station (sending)
- satellite
- ground station (receiving)

The sending station sends configuration commands to the satellite. The satellite receives the commands, processes them, and, when necessary, sends messages to the receiving station.

The top-level model PUS model shows the three major modules.

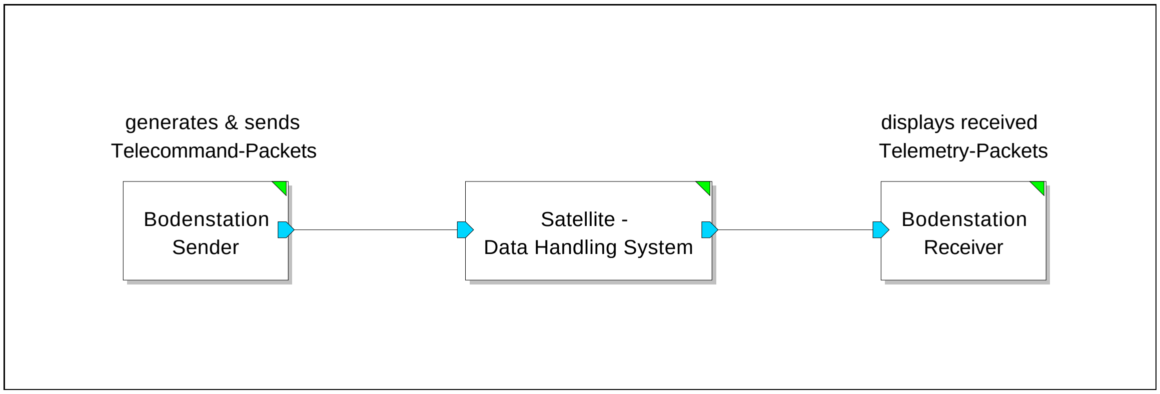
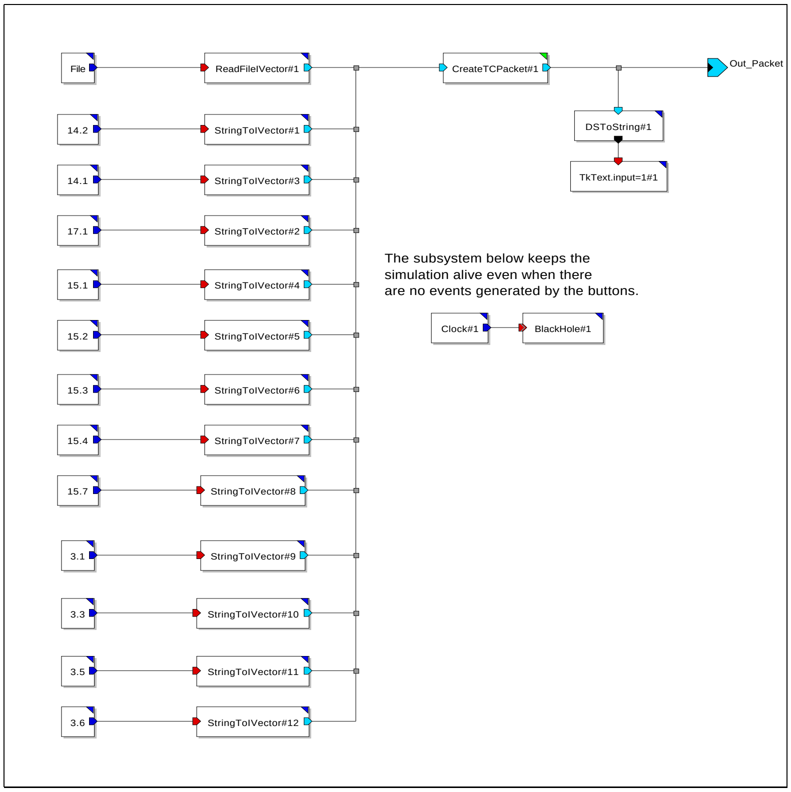


Figure 1: PUS top-level model

The sending ground station is essentially a large command system with a repertoire of control packets to send to the satellite.



The subsystem below keeps the simulation alive even when there are no events generated by the buttons.

Figure 2: Sending ground station

The module is linked to a Tcl control panel which is used to select the commands dynamically during simulation. When the appropriate button is pressed, the sending station module generates the appropriate control packet and sends it to the satellite model. A keep-alive circuit keeps the system operating where there are no commands.

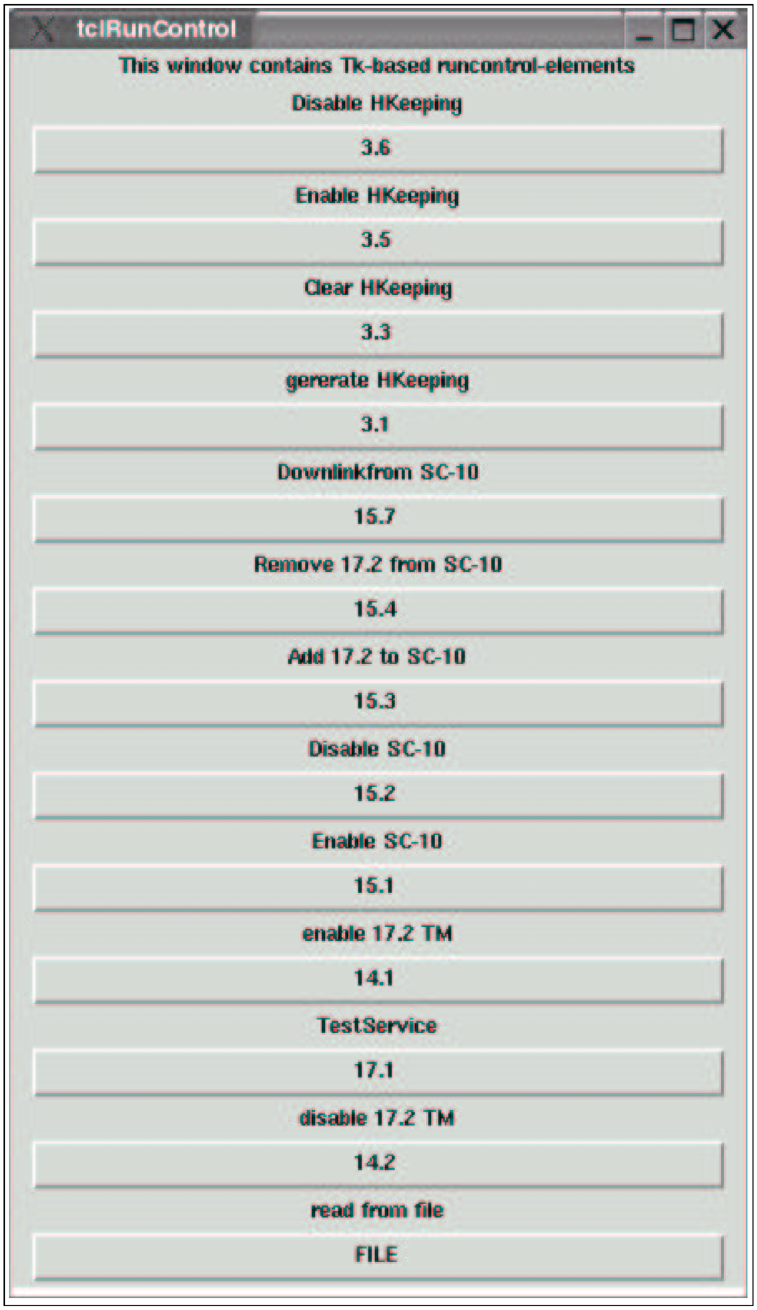

This window contains Tk-based runcontrol-elements

Figure 3: The PUS dynamic control panel

The satellite model consists of an uplink a router that sends received packets to applications services or downlink, a downlink, an applications module, a services module, and a housekeeping module.

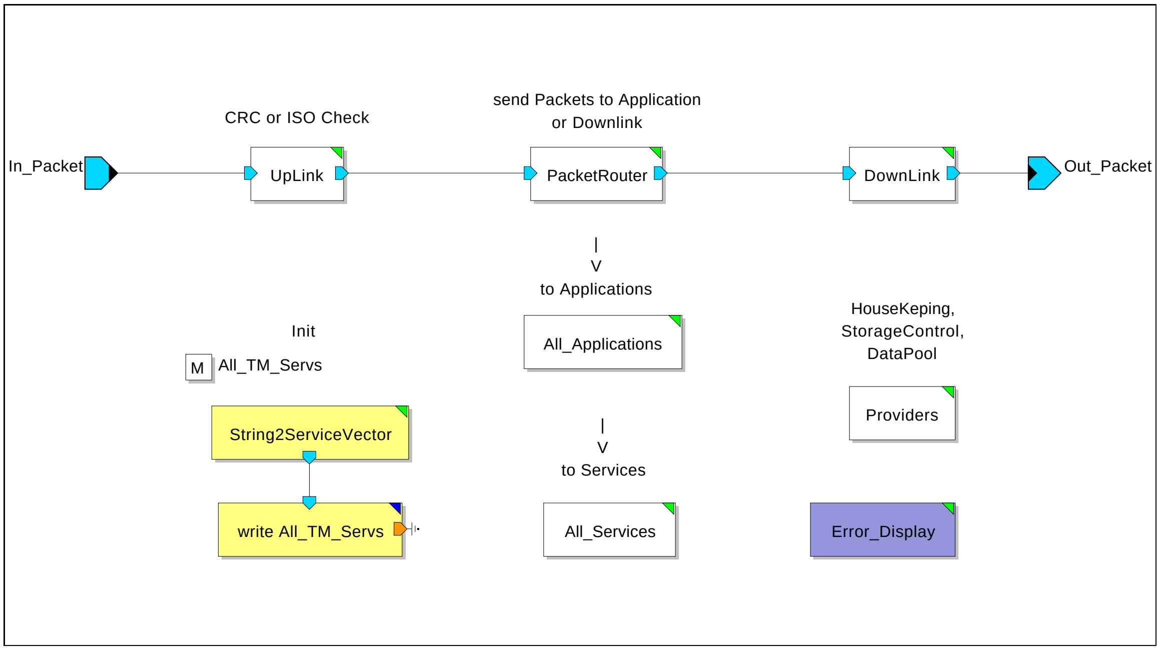


Figure 4: The satellite module

The packet router is shown below.

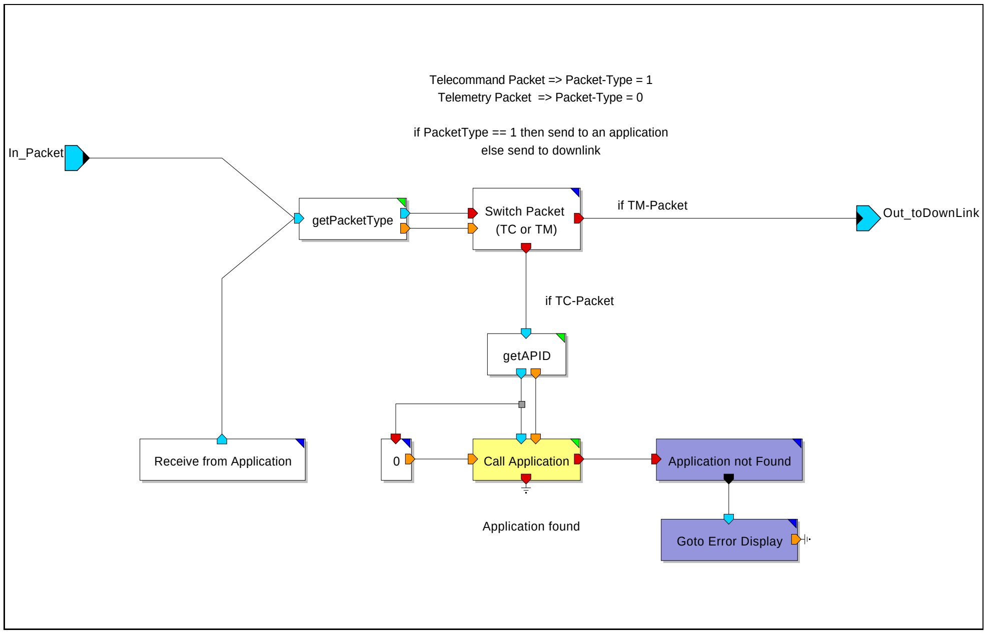


Figure 5: Packet router

The applications module shows two applications.

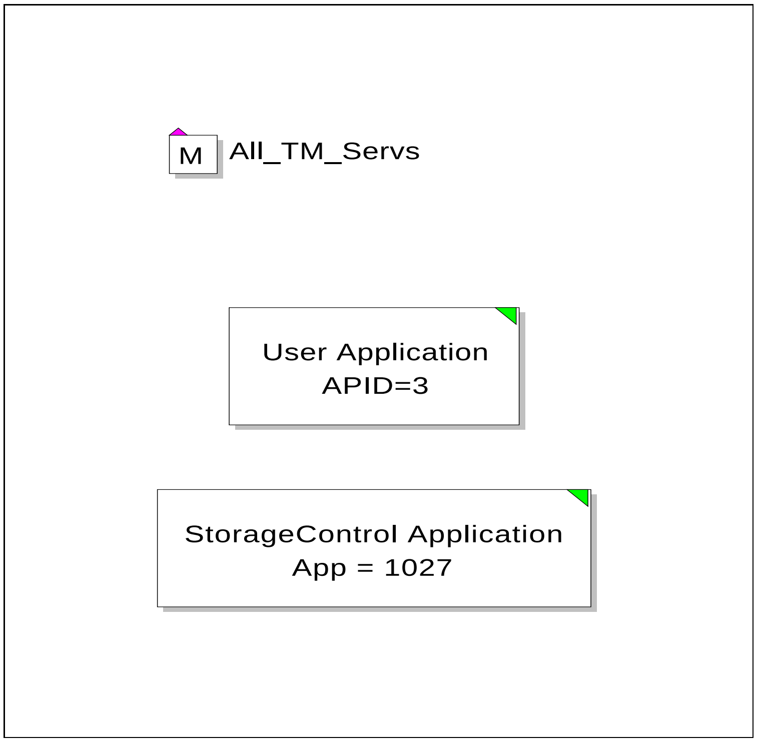


Figure 6: Applications module

The services module lists all the services provided by this PUS model.

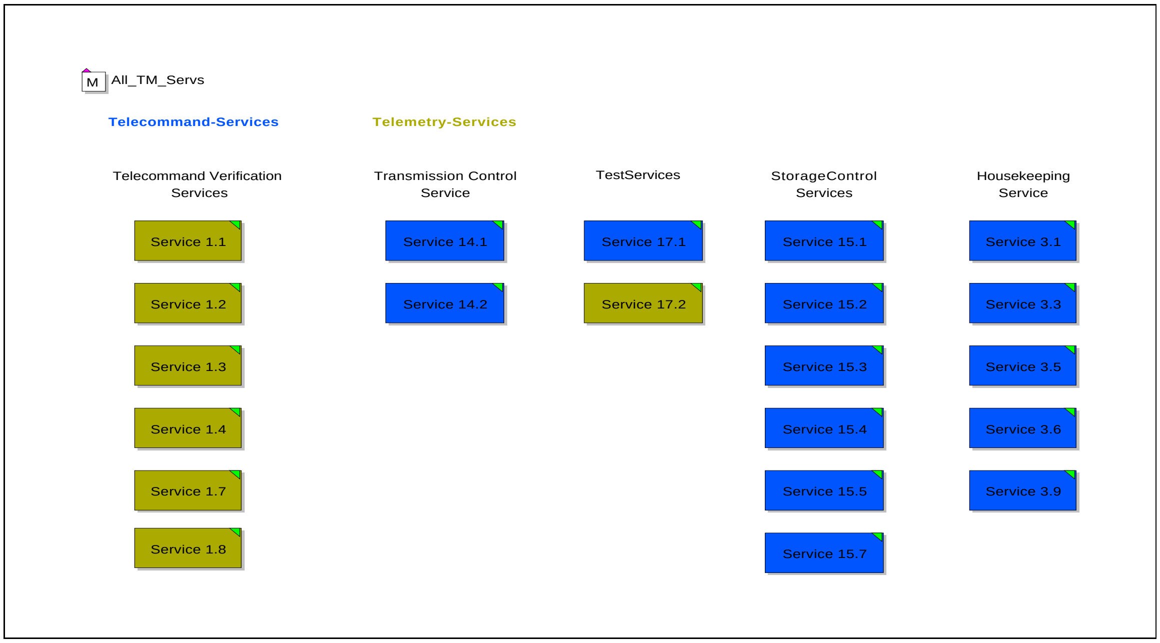


Figure 7: Services module

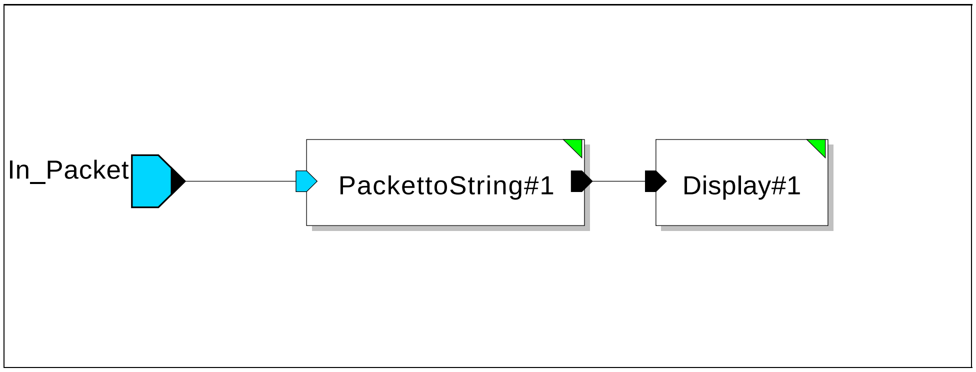
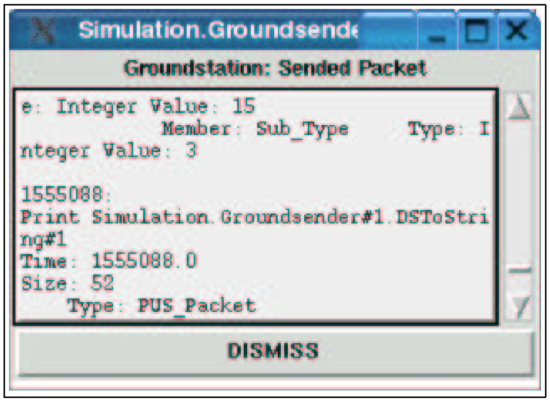


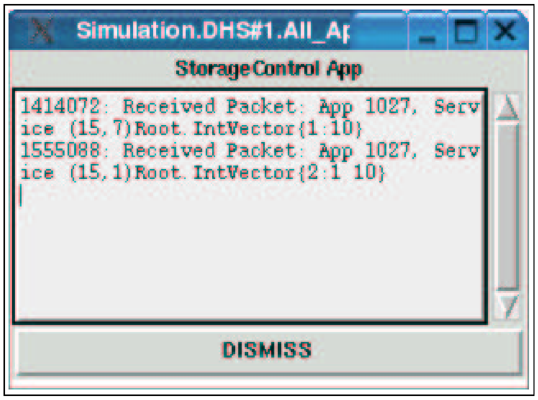
Figure 8: Receiving ground station

Dynamic windows show the receipt of various types of packets. Three of these windows are shown below.



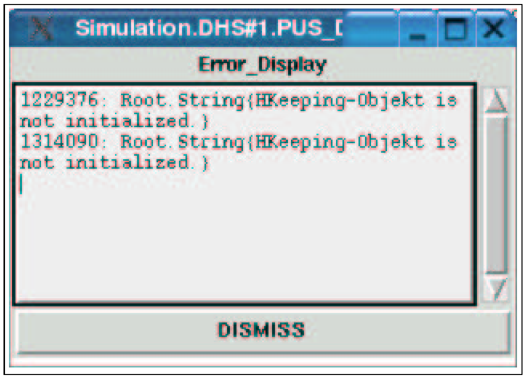
```
e: Integer Value: 15
           Member: Sub_Type        Type: I
nteger Value: 3

1555088:
Print Simulation.Groundsender#1.DSToStri
ng#1
Time: 1555088.0
Size: 52
    Type: PUS_Packet
```

Figure 9: Sent packet



```
1414072: Received Packet: App 1027, Serv
ice (15,7)Root.IntVector{1:10}
1555088: Received Packet: App 1027, Serv
ice (15,1)Root.IntVector{2:1 10}
```

Figure 10: Storage application



```
1229376: Root.String{HKeeping-Objekt is
not initialized.}
1314090: Root.String{HKeeping-Objekt is
not initialized.}
```

Figure 11: Error window

Note: Model developed by Peter Unger.