# Function Block Approach for Abstraction of ISO 11783 Virtual Terminal User Interfaces in Agricultural Machines

Samuel Brodie, Seung-Yun Baek, Yong-Joo Kim and Timo Oksanen

*Abstract*— **The ISO 11783 (also known as ISOBUS) standard defines a protocol for communication between an agricultural implement and a Virtual Terminal (VT) — a commonly used, standardized, automatic/plug-and-play, inter-manufacturer device for displaying the user interface of the connected implement to the driver. However, unlike other aspects of the ISO 11783 standard, the VT is dependent on the specifics of the chosen user interface. Thus, software libraries cannot abstract the functionality of the VT by simply abstracting the message encoding and decoding, as is the case for sending and receiving periodic signals such as status parameters etc. In this paper a modern code architecture is presented which uses a separation of concerns methodology to devise VT-specific abstraction layers which ease development for designers who may not be well versed in the ISO 11783 standard.**

## I. INTRODUCTION

ISO 11783 (also known by the marketing name ISOBUS) is a widely-used standard series defining plug-and-play inter-manufacturer communication in agricultural machines, namely tractors and implements. ISO 11783 defines not only the sharing of information between nodes, but also defines a Virtual Terminal (VT) device which is a screen acting as a graphical user interface (GUI). Due to standardization, an implement can automatically upload its GUI information to the VT and the VT will be able to interpret and display it [1]. This avoids the situation of a farmer needing a separate control box and/or display for each implement [2] and is therefore a desirable feature from a usability standpoint.

Even as tractors progress to higher autonomy levels (up to SAE J3016 level 4 High Driving Automation [3]), the variation between distinct implement types will likely mean that a proprietary GUI must still be offered by the implement for the driver to handle specific tasks. A standardized interface design would not be sufficient to cover the multitude of differing implement types.

It can be difficult for entities that want to add ISO 11783 compatibility for the first time due to the breadth and depth of functionality covered by the 14 standard documents (consisting of 1144 pages [4]). For this reason, there are paid and free/open-source libraries which can abstract the low-level details away from developers and ease development. Such libraries may be useful for implement manufacturers who want their equipment to be ISO 11783-compatible (and get ISOBUS certification); in particular, small manufacturers who may not have the scale to develop their own library.

ISO 11783 networks use CAN 2.0B as the physical layer and encode a parameter group number (PGN) into the ID field of the CAN frames which signifies the frame's purpose. The data field of the CAN frame is encoded differently depending on the PGN. For example, ISO 11783 defines that the rear hitch pitch angle must be encoded into bytes 5 and 6 of a CAN frame with a PGN of 61697, and using a scale of 0.002 degrees/bit [5].

The abstraction of the ISO 11783 protocol can include many aspects:

- Encoding/decoding CAN ID and data fields to/from usable data such as engine speed.
- Abstracting the ISO 11783 address claiming procedure which is required before a participant can begin sending messages on the bus.
- Implementing Transport Protocol and Extended Transport Protocol which allow the transfer of data which is longer than 8 bytes (the payload of one CAN frame) [6].

Together, this can make certain aspects of ISO 11783 relatively easy to implement for manufacturers even without any knowledge of the underlying protocol. For simple, periodic data (such as hitch pitch angle or wheel speed) complete abstraction of the underlying protocol is possible because the information can be passed to a function which simply encodes a CAN frame (ID and data fields) as specified in the standard and writes it to the bus. However, the VT-related parts of the ISO 11783 protocol cannot be abstracted so simply and GUI designers must have detailed knowledge of the VT protocol to build functional code. This is because the VT protocol cannot be abstracted by abstracting only the underlying CAN IDs and encoding, but instead it requires abstraction of the whole VT protocol stack, the details of which are dependent on the proprietary GUI of each implement. For designers and control engineers who are not

*The project is supported by funds of the Federal Ministry of Food and Agriculture (BMEL) based on a decision of the Parliament of the Federal Republic of Germany. The Federal Office for Agriculture and Food (BLE) provides coordinating support for digitalisation in agriculture as funding organisation (28DE112B18).

Samuel Brodie and Timo Oksanen are with the Technical University of Munich (TUM), Chair of Agrimechatronics and Munich Institute of Robotics and Machine Intelligence (MIRMI), Freising 85354 Germany (corresponding author: samuel.brodie@tum.de).

Seung-Yun Baek and Yong-Joo Kim are with the Department of Smart Agriculture Systems, Chungnam National University (CNU), Daejeon 34134, Korea.

**ISBN-13:** 978-3-911430-00-5

familiar with ISO 11783 this can pose a challenge and a large time investment.

This paper presents a software architecture which goes beyond simply abstracting the CAN message encoding and rather abstracts the whole lower-level VT protocol details away. The purpose of the architecture is to allow designers to focus on design-level code without the need for direct VT control.

## II. Background

One of the latest big developments in ISO 11783 is the Tractor-Implement Management (TIM) functionality. Class 3 machines (machines that are able to control tractor functions from the implement) have been studied from the perspective of automation and functional safety [7]. This has followed on to become ISO 11783 Tractor Implement Management (TIM) which standardizes an authentication framework to allow implements to authenticate with the tractor before taking control of certain functions (speed, steering, hitch position etc.) [8, 9].

Another area of ISO 11783-related research is the development of a so-called High-Speed ISOBUS (HSI). HSI is being developed as the next generation of ISO 11783, replacing the CAN bus with an Ethernet network and an appropriate middleware [10, 11].

There is a range of scientific work researching the adaptability of ISO 11783 in the fields of autonomy and robotics. Owing to the inter-manufacturer, plug-and-play properties of ISO 11783, a new and promising area of research is investigating how ISO 11783 networks can be utilized in agricultural robotics. For automated robots which are able to attach and use small implements (rather than single-task robots), these properties are valuable [12, 13]. In [12] and [13] the authors create a VT GUI for a CEOL robot (AgreenCulture, Toulouse, France).

### A. ISO 11783 Virtual Terminal

ISO 11783 defines the use of a VT to act as a GUI [1]; commonly these are touch screens, but this is not necessarily the case. In Figure 1, an example VT display is shown. On the right-hand side is the *soft key mask* area which contains the *soft key designators*. Activation of the *soft key* can be done with *physical soft keys* assigned to each *soft key designator*; these can be either hardware buttons or linked to an area of the touch screen. The larger section of the screen is the *data mask area* where UI elements such as shapes, text, and inputs can be shown based on the proprietary design of the connected implement [1].

At initialization, both the VT and implement must complete the address claim procedure (as any node must) to begin communicating on the bus. Then the implement uploads its Implement Object Pool (IOP) to the VT. The IOP is a set of objects such as numeric input field, shapes, or string output field (where the object types are defined in ISO 11783-6 [1]), each with a unique *Object ID* which are encoded into a byte string that is uploaded by an implement and decoded by a VT. The IOP defines the GUI with several object types. The objects are defined in the IOP in a hierarchy (objects have parent objects) and objects have specific attributes depending on their type (commonly *X* and *Y position* relative to the parent, *color*, and *value*). Some attributes which are more specific include font type, and the *scaling* and *offset* which ISO 11783 uses to

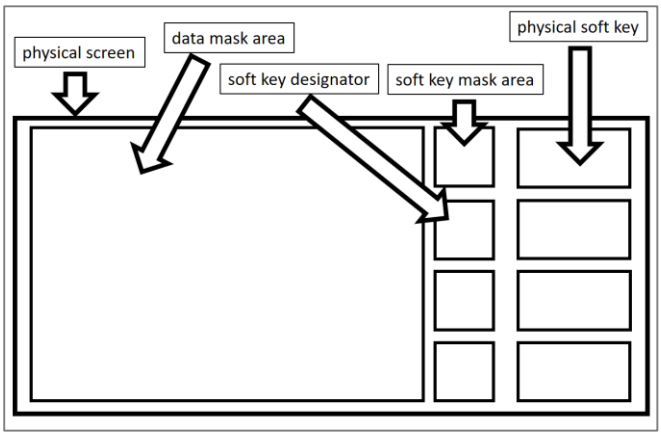


*Figure 1 – An example Virtual Terminal screen layout with annotations (adapted from [1])*

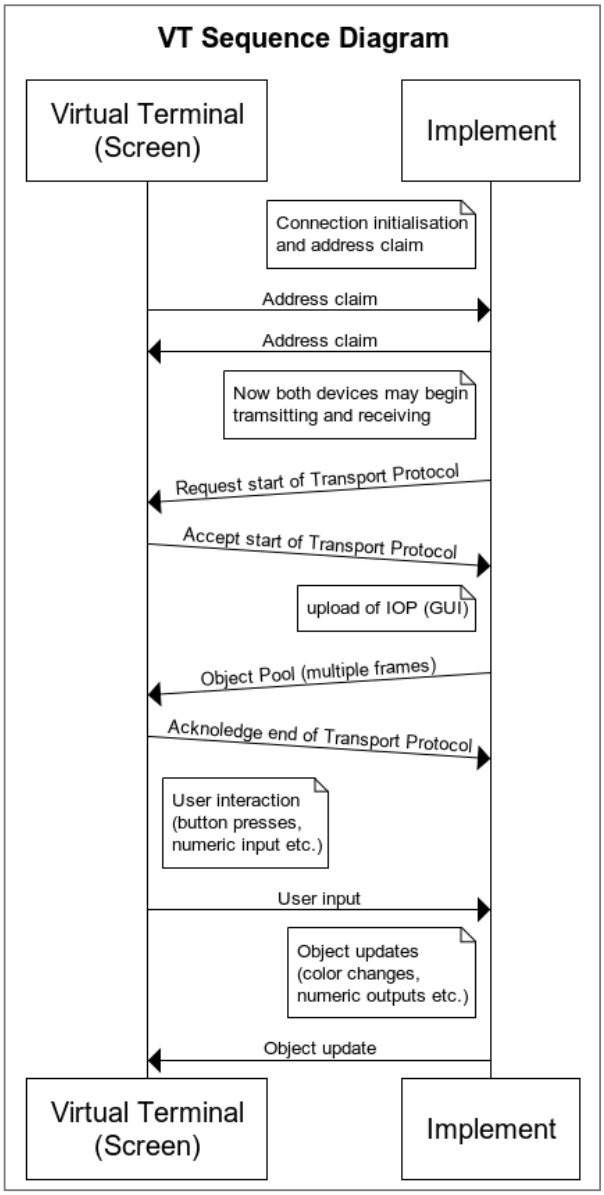


*Figure 2 A sequence diagram showing the initialization of a Virtual Terminal*

represent decimal numbers (as all values are sent on the bus as unsigned integers. The IOP can be generated using software such as OSB VT Designer (OSB connagtive GmbH, Munich, Germany) or Jetter ISO-Designer (Bucher Automation AG, Ludwigsburg, Germany). Once the IOP has been uploaded, the initialization is complete and the VT can display the implement's GUI to the user. The entire process is shown in Figure 2. The library handles the scaling fo the IOP to the screen size of the connected VT.

The different "screens" of the GUI are called *masks*. The implement should be aware of which mask is active at any time, however, the VT status message also contains information about currently active mask. The implement can manage transitions between the masks in two ways [1]:

1. Through sending a *Change Active Mask* message to command the VT to display a particular mask.
2. Through the use of *Macros* in the IOP which can be described as a list of commands which the VT will automatically execute after specific trigger conditions have been met (e.g. when a soft key is activated, the VT should automatically execute a *Change Active Mask* command, without the implement needing to send any CAN messages). The standard notes that "a Macro that executes a Change Active Mask command will trigger a VT Status message, but will not trigger a Change Active Mask response."[1]

The implement must keep track of the objects that are visible on the VT at any time. Primarily this consists of knowing which objects are part of which mask (objects can also be hidden/shown within a mask; updating hidden objects these will induce unnecessary busload but will not cause display errors).

### *B. Code*

Popular open-source ISO 11783 libraries such as IsoAgLib and AgIsoStack++ abstract the CAN message encoding but still require the developer to understand the underlying ISO 11783 VT [14, 15]. For example, in the AgIsoStack++ tutorial, the developer must do the scaling and offset of numeric values themselves [16] because the VT messages do not allow encoding of floating point or signed numbers.

Another example is the difference between VT buttons (clickable objects drawn on the central data mask area of the screen) and VT soft keys (clickable objects in the soft key area at the sides of the screen). VT buttons have a latchable property defined in ISO 11783 (i.e. the object switches between two states each time it is clicked, as opposed to when held/released), but VT soft keys do not. Therefore, if a designer desires a latchable soft key then they must implement this code themselves because a latchable soft key is not included in the standard. This puts extra work on the designer who must understand the ISO 11783 scaling and offset, the soft key states, and build latching behavior independently.

## III. ARCHITECTURE

We have seen that the ISO 11783 standard makes it a requirement for implements to track which objects are drawn

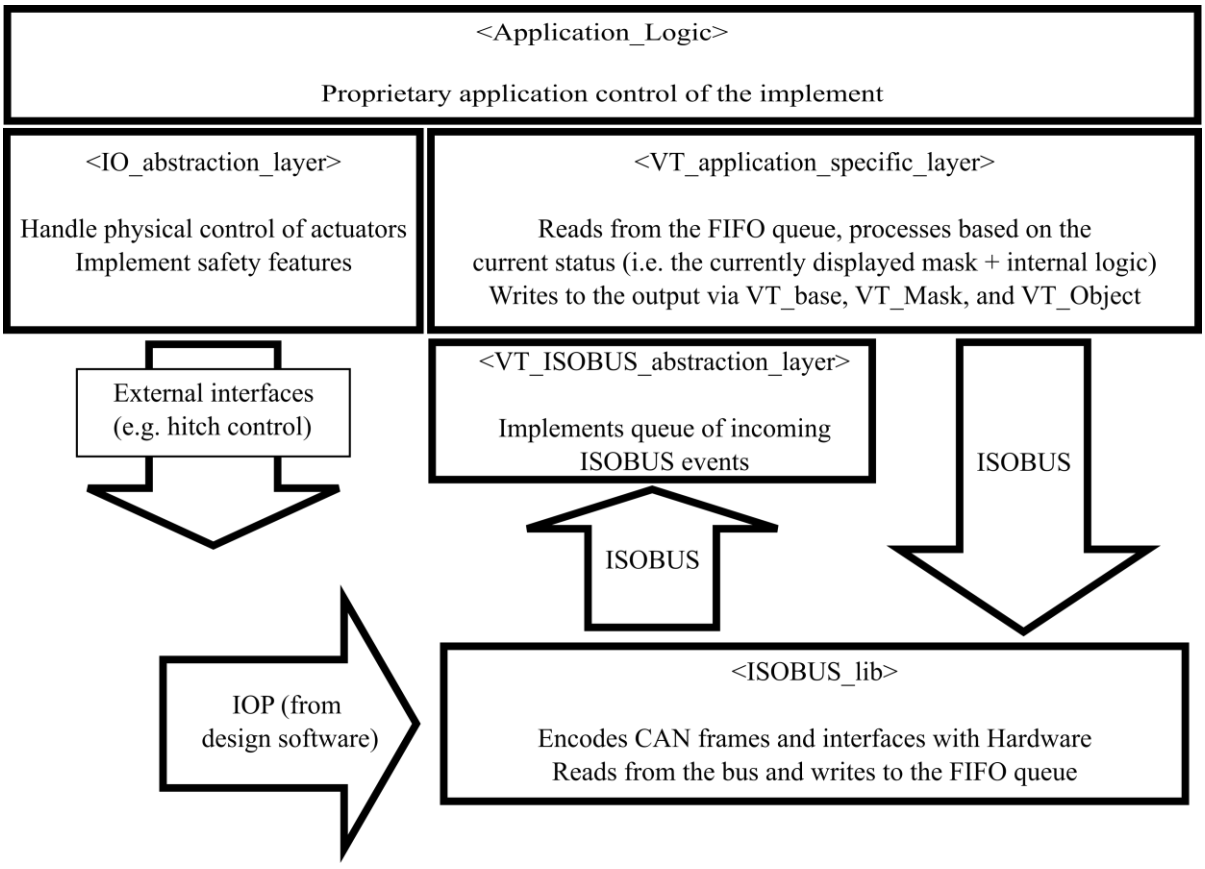


*Figure 3 – The presented software architecture. The application logic is separated from the Virtual Terminal-specific layer*

on the active data mask at any given time. This cannot be handled through abstracting the CAN message encoding, but rather requires additional code structures to maintain the state of the GUI. An architecture is therefore presented to address this.

The proposed architecture gives several different types, and layers, of abstraction. It is applicable to applications where an implement which will upload a GUI to the VT that has interactive elements which connect with physical effects. This design will allow user-input such as button presses and entry of text and/or numeric values, and it will also provide information to the user in the form of updated text, numeric values, colors etc.

### A. Layers

The architecture splits the overall application into layers as shown in Figure 3. Using a separation of concerns software design principal, each layer has a distinct purpose. This also abstracts lower layers from one another and separates the UI design from application logic and hardware Input/Output (I/O).

#### 1) Application_Logic

The *Application_Logic* contains the application in the highest level of abstraction. This layer defines the overall application of the implement, for example the logic of how the machine should function, such as when a user should be allowed to perform particular tasks, or enabling of a field mode and transport mode. This layer does not handle the lower level implementation details of ISO 11783.

#### 2) IO_abstraction_layer

The *IO_abstraction_layer* provides an interface between the application logic and the machine's I/O such as the connections between physical actuators, sensors etc. and their corresponding connectors. Here, the safety logic and control logic of the physical IO can be implemented and abstracted away from the application.

#### 3) VT_application_specific_layer

The *VT_application_specific_layer* abstracts the VT protocol away from the GUI designer and allows them to do GUI application functions without needing to understand ISO 11783 (e.g. the scaling and offset system, the masks system, and so on). Here, the GUI designer will program the GUI logic (which conditions trigger a change of mask, or update an object) without knowledge of ISO 11783.

This layer uses the VT classes (VT_Base, VT_Mask, VT_Object described in detail below) to abstract ISOBUS functions.

#### 4) VT_ISOBUS_abstraction_layer

The *VT_ISOBUS_abstraction_layer* does not need to be changed by the designer as it is not design specific. It implements a FIFO queue of ISOBUS VT events coming from the VT server (user) side (button presses, number inputs etc.). The purpose of this queue is to allow the program to check the ISOBUS message queue with a relatively high frequency, handle ISOBUS time-sensitive events such as address claim (250ms timeout), and queue VT events for later handling (at a lower frequency) if desired. That is to say, if a single-threaded program is performing a computationally intensive task, it must relinquish control to clear and respond to the ISOBUS CAN message queue in compliance with the 250 ms timeout of ISO 11783 VT messages (by calling *VT_ISOBUS_abstraction_layer::iterate()*). However, the *VT_ISOBUS_abstraction_layer* allows the VT messages to be acknowledged (with an appropriate CAN frame) while the event itself is queued for the application to handle later.

#### 5) ISOBUS_lib

The *ISOBUS_lib* layer contains a general ISO 11783 library which handles the encoding and decoding of CAN frames, and interfaces with the CAN hardware. This layer initializes the CAN bus hardware, handles address claiming, and uploads the IOP to the VT.

### B. VT Classes

The VT classes are to be used as the application-code representations of the IOP objects. The classes communicate with the lower layers and abstract the ISO 11783 implementation of common design functionalities such as latching soft keys. The classes are initialized with instances which correspond to a counterpart in the IOP of the same type (e.g. the developer should initialize one instance of *VT_Mask* for each mask in the IOP, one instance of VT_Obj_softkey for each soft key etc.).

Three classes are defined: *VT_base* which represents the root of the IOP, *VT_Mask* which represents a mask, and VT_Object which represents an object in general and from which the classes for specific object types inherit.

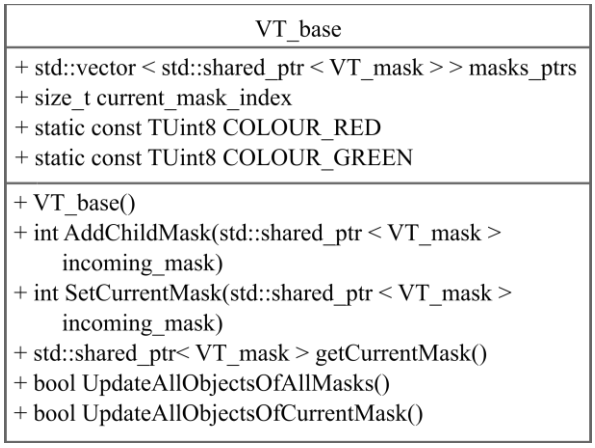


*Figure 4 – UML diagram showing the key elements of the VT_base class*

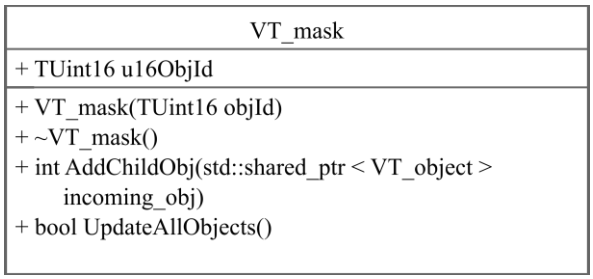


*Figure 5 – UML diagram showing the key elements of the VT_mask class*

### 1) *VT_Base*

Each of the GUIs (normally only one per implement but not necessarily) defined in the IOP will have a corresponding instance of the *VT_Base* class as shown in Figure 4. Instances of *VT_Base* have pointers to their child *VT_Masks*. *VT_Base* is responsible for tracking the mask which is currently displayed on the VT.

### 2) *VT_Mask*

Each of the masks defined in the IOP will have a corresponding instance of the *VT_Mask* class as shown in Figure 5. Instances of VT_ Masks have pointers to their child *VT_Objects* as a means of tracking which objects are shown on the screen whenever that mask is the active mask.

### 3) *VT_Object*

As shown in Figure 6, the VT_ Object class is a base class. Each ISO 11783 object type has a class derived from *VT_Object* such as numeric inputs, softkeys etc. Each object defined in the IOP will have a corresponding instance of a class derived from the *VT_Object* base class.

The *VT_Object* class handles the scaling and offset because ISO 11783 VT communication doesn't support floating point values, instead unsigned integer values are sent, and scaled for display by the receiver which uses the offset and scaling factor defined for that object in the IOP. This means that updating values is not a simple as purely encoding the CAN message correctly, but it also needs to factor in the unique properties of each object.

The ISO 11783 *soft key* object type is used to draw a soft key on the VT (different to a button, these are the special keys shown at the edges of the screen). This soft key type does not have any concept of latching and so latching must be implemented by the implement by maintaining the latched state of the soft key and updating the visuals of the object to inform the user of the latch status.

## IV. USE CASE: HITCH CONTROL

As a use case for the proposed architecture, a hitch-control application was made and the I/O layer had simulated inputs and outputs. The function of the application is as follows:

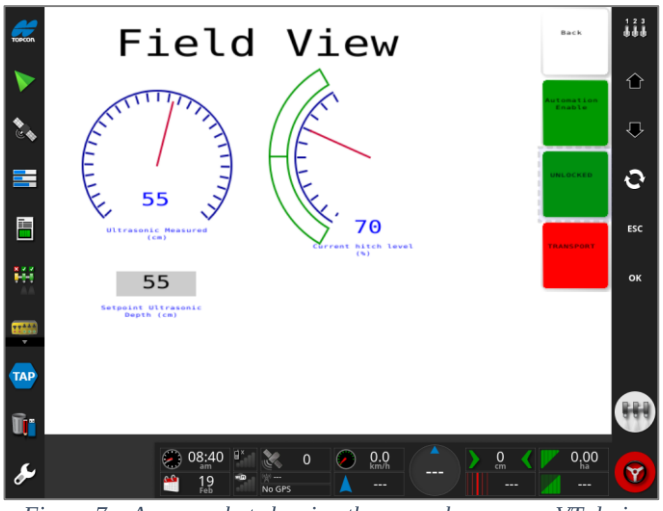


*Figure 7 – A screenshot showing the example use case VT design, taken on a Topcon VT*

There is a manual mode/auto mode switch. In manual mode, the up/down soft keys directly control the hitch position. In auto mode there is a setpoint (numeric input) field and the hitch automatically moves to the correct position based on the readings of a range sensor. There is a settings page — accessible via soft key — with numeric inputs to tune the controller (gain etc.). The application utilizes the previously defined layers. In Table 1 an example is presented of the

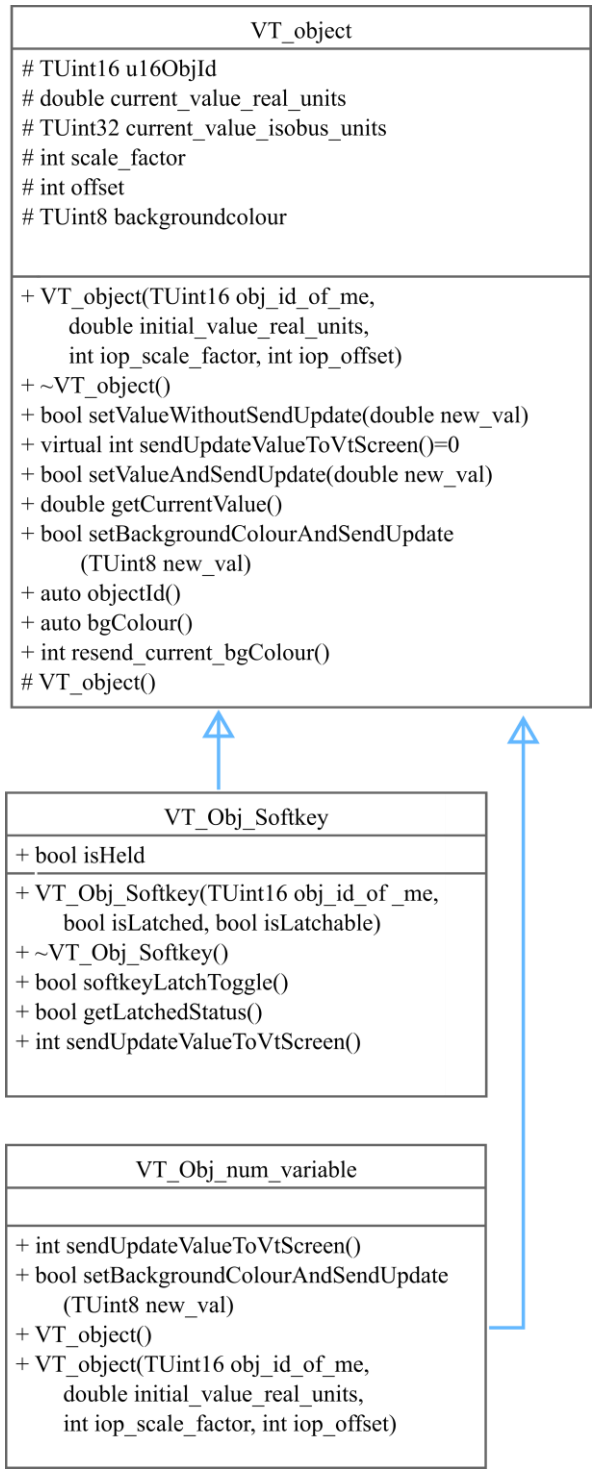


*Figure 6 – UML diagram showing the key elements of the VT_object, VT_Obj_Softkey and VT_Obj_num_variable* classes. *VT_Obj_Softkey and VT_Obj_num_variable inherit from VT_object*

logical flow of the program and in Figure 7 the main mask of the application is shown on a Topcon VT screenshot.

*Table 1 – Step by step example proof of concept application*

| # | Layer | Action |
|---|---|---|
| 1 | N/A | The system is powered on. The nodes initialize and the IOP is uploaded to the VT.<br>The user presses the manual-mode soft key on the terminal, resulting in a CAN message. |
| 2 | ISOBUS Library | The incoming CAN message is interpreted and added to the VT abstraction layer queue (button pressed/released event). |
| 3 | VT ISOBUS Abstraction | The event is held in the queue until the application is ready to act upon it. |
| 4 | VT Application Specific | The VT events queue is read and objects are updated appropriately:<br>The manual mode VT_Obj_Softkey class object latches and changes color. |
| | ISOBUS Library | The appropriate CAN message is sent to the VT to update the soft key color. |
| 5 | N/A | The user presses and holds the hitch-down soft key on the terminal, resulting in CAN messages from the VT. |
| 6 | ISOBUS Library | The incoming CAN message is interpreted and added to the VT abstraction layer queue (button held event) |
| 7 | VT ISOBUS Abstraction | The event is held in the queue until the application is ready to act upon it. |
| 8 | VT Application Specific | The VT events queue is read and objects are updated appropriately:<br>The desired position VT_numeric class object updates its value. |
| 9 | VT Application Specific | Application-defined minimum hitch setpoint options are enforced. |
| 10 | I/O Abstraction | The desired position of the hitch is sent to the I/O with the appropriate physical connections, feedback loop etc. |
| 11 | VT Application Specific | The current hitch position is evaluated and the actual position VT_numeric class object updates its value and sends appropriate CAN messages to the VT via the ISOBUS Library. |

## V. Conclusion

Many parts of the ISO 11783 protocols can be abstracted by simply abstracting the CAN message encoding, or by implementing logic to handle events such as address claims. However, the VT is too use case-specific for the full VT runtime functionality to be abstracted in this way. This paper shows a function block approach to address this. A set of class instances which mirror each VT object type can be used to allow further distance between the designer and the ISO 11783 standard. By utilizing instances of the appropriate class to mirror the objects in the IOP, a software architecture is designed so as to enable separation of concerns and allow VT GUIs and VT-enabled applications to be designed without detailed knowledge of the underlying protocol.

An example VT design was created to prove the concept and was tested by using the described approach to successfully create a working interface with automatic control in a plug-and-play multi-brand system.

In the future a full software package could be created which implements classes for each of the object types that can be put into an IOP.

## VI. References


[1] *Iso 11783-6:2018 Tractors and Machinery for Agriculture and Forestry — Serial Control and Communications Data Network — Part 6: Virtual Terminal*, Iso, 2018.

[2] S. Brodie, T. Oksanen, and H. Auernhammer, "Buzzword Isobus," *Informatik Spektrum*, vol. 46, no. 1, pp. 46-50, 2023.

[3] *Sae J3016 — Taxonomy and Definitions for Terms Related to Driving Automation Systems for on-Road Motor Vehicles*, Society of Automotive Engineers, 2021.

[4] T. Oksanen and H. Auernhammer, "Isobus— the Open Hard-Wired Network Standard for Tractor-Implement Communication, 1987-2020," presented at the Agricultural Equipment Technology Conference, Louisville, Kentucky, USA, Feb, 2021, 913C0121. [Online]. Available: http://elibrary.asabe.org/abstract.asp?aid=52060&t=6.

[5] Vdma. "Isobus Data Dictionary - Pgn> 61697 Primary or Rear Hitch Roll and Pitch." https://www.isobus.net/isobus/pGNAndSPN/2652?type=PGN (accessed 24th of January, 2024).

[6] *Iso 11783-3:1998 Tractors and Machinery for Agriculture and Forestry — Serial Control and Communications Data Network — Part 3: Data Link Layer*, Iso, 1998.

[7] A. Ronkainen, "Design Considerations for Isobus Class 3 Machinery System's Human-Machine Interaction," *IFAC Proceedings Volumes*, vol. 46, no. 18, pp. 259-263, 2013.

[8] *Aef Guideline 023 Rig 3 (2023), Isobus - Isobus Automation Principles*, Agricultural Industry Electronics Foundation, 2023.

[9] Agricultural Industry Electronics Foundation. "Tractor Implement Management (Tim) - Aef Online." https://www.aef-online.org/about-us/activities/tractor-implement-management-tim.html (accessed Jan 18, 2024).

[10] D. Smart and V. Brill, "Aef – High Speed Isobus – Technology Readiness for a Next Generation Network," in *LAND.TECHNIK 2022*, Düsseldorf, 2022, vol. 2395: VDI, in VDI-Berichte, 1 ed., pp. 551-558, doi: 10.51202/9783181023952-551.

[11] D. Smart and V. Brill, "High Speed Isobus, an Aef Project for Next Generation Ag Networking," in *LAND.TECHNIK 2019*, Düsseldorf, 2019, vol. 2361: VDI, in VDI-Berichte, 1 ed., pp. 91-106, doi: doi.org/10.51202/9783181023617.

[12] G. M. Sharipov *et al.*, "Smart Implements by Leveraging Isobus: Development and Evaluation of Field Applications," *Smart Agricultural Technology*, vol. 6, p. 100341, 2023.

[13] G. M. Sharipov *et al.*, "Communication between Agricultural Robot and Mechanical Weeding Machine Based on Iso 11783 Network," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 8902-8907, 2023.

[14] E. Tumenjargal *et al.*, "Development of Isobus Universal Terminal and Client Ecus for Agricultural Machinery," in *2018 ASABE Annual International Meeting*, 2018: American Society of Agricultural and Biological Engineers, p. 1.

[15] E. Tumenjargal, L. Badarch, H. Kwon, and W. Ham, "Embedded Software and Hardware Implementation System for a Human Machine Interface Based on Isoaglib," *Journal of Zhejiang University Science C*, vol. 14, no. 3, pp. 155-166, 2013.

[16] Open-Agriculture. "Agisostack-Plus-Plus Virtual Terminal Client Example." https://github.com/Open-Agriculture/AgIsoStack-plus-plus/blob/9503ee9e207030de92c641a78aef96966d29586a/examples/virtual_terminal/version3_object_pool/main.cpp (accessed Jan, 2024).