International Journal of Core Engineering & Management
Volume-5, Issue-12, March-2019, ISSN No: 2348-9510

STEM DATA FLOW DESIGN FOR REAL TIME EMBEDDED SYSTEM

Binoy Kurikaparambil Revi
binoyrevi@live.com
Independent Researcher

Abstract

Data flow management in the real time embedded system with measurements and controls is a critical design factor that affects system parameters, software quality, and reusability. I am trying to explain the Stem Data Flow Design (SDFD) methodology that can be extremely useful in managing the data flow in a real time embedded system with a typical QML display and sensors. I am using the QML display as I have done my research using QML and QT framework, but it is not limited to QML and is applicable to most of the UI technology I can imagine.

I. INTRODUCTION

When designing a typical embedded system that has data inputs like the data from sensors, network, disk, and database, the data flow needs to be managed using a well-defined data architecture that can help software components to receive data quickly and transmit to destination as soon as possible. In many systems the component failure use cases play a critical role in designing the dataflow such as failure of one component should not affect the failure of another component and the failed components must be safely isolated. Other factors that are considered while designing the data flow are reusability and scalability. Reusability consideration can be within the same project development or across multiple projects. Stem Data Flow Design (SDFD) helps to design modular software components that can be reused for various projects. By using Stem Data Flow Design (SDFD), scalability can be achieved by scaling up or down a component without changing the architecture, which also increases the robustness of the software as it matures.

II. TYPICAL EMBEDDED SYSTEM DESIGN

Consider a software system as in the figure (1) below that reads the data from the sensors using a software thread and sends the data from the backend to the frontend display and let's assume the response from the frontend display flows down to the backend to execute control functions. Also, assume that this system also has an automated control function that executes controls without user input.

Now Let's analyze the dataflow from bottom to top. Data from the sensors are read by the sensor control thread and transferred to the backend C++ module. The data then flows all the way down to QML components and finally to the QML widgets. The most important key aspect of this design is that the software components are designed to tap on the upward data flows in a hierarchical order of components design. This is important because this design makes the overall architecture to use and create reusable components, enabling easy scalability and maintainability. By cutting the corners, one can easily plug a software component to an incorrect data point thinking that the code

International Journal of Core Engineering & Management
Volume-5, Issue-12, March-2019, ISSN No: 2348-9510

can be reduced or data transfer can be faster. But from my research and analysis this can cause very severe adverse effects. Example, in the diagram above, if a developer connect the QML Widget 4 directly to the Main.qml thinking it can receive data quickly, but it can lead to following issues:

1. Data displayed on the UI components may not be synchronized with other UI components.
2. It complicates signal and slot designs that can make the components difficult to reuse and maintain. Often end-up in the famous patching game.
3. Control data becomes unreliable and may lead to undesired outcomes.
4. Difficult to scale the UI functionality.

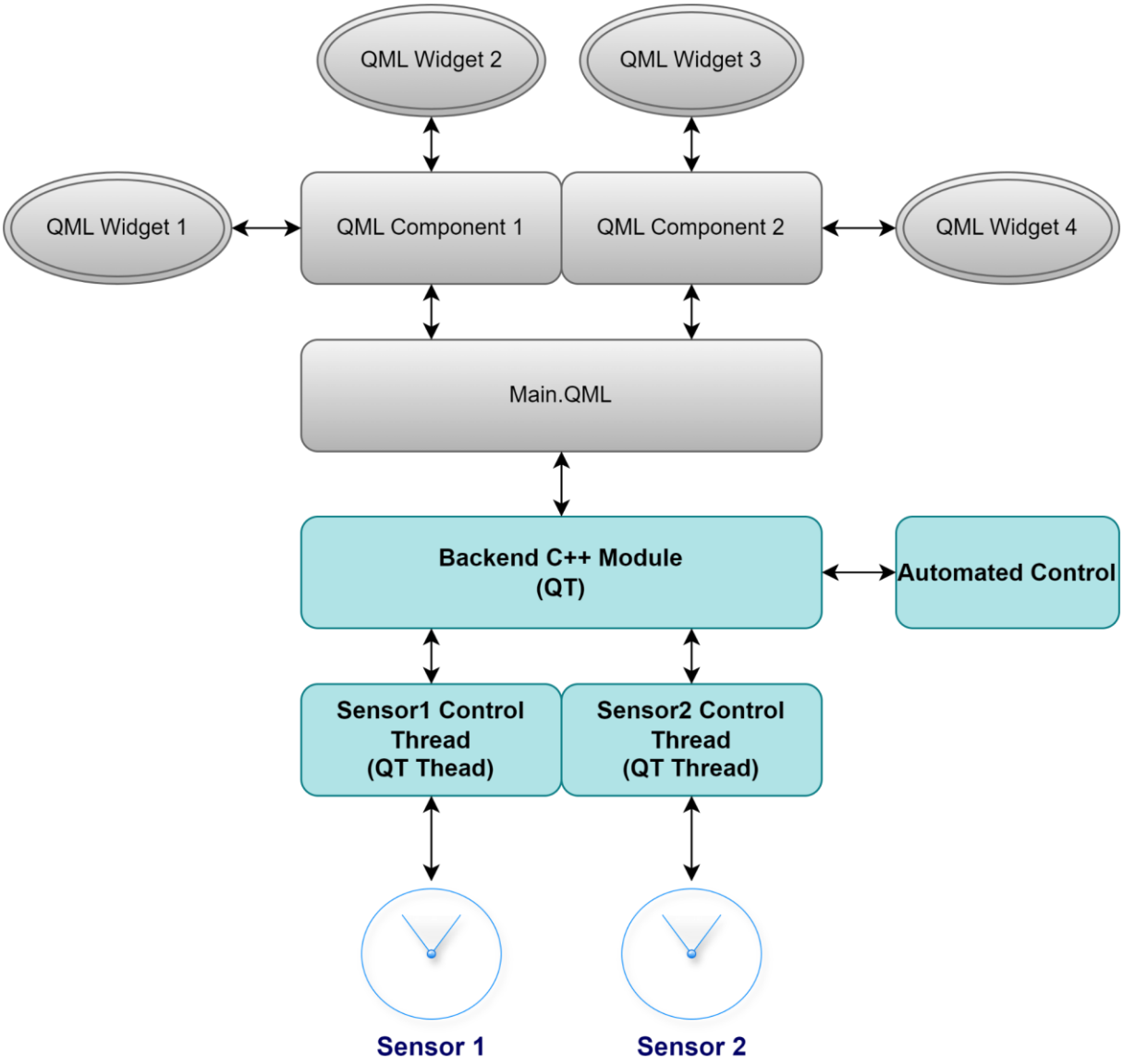

Figure 1: Typical Embedded System Design using QT framework

International Journal of Core Engineering & Management
Volume-5, Issue-12, March-2019, ISSN No: 2348-9510

III. STEM DATA FLOW DESIGN(SDFD)

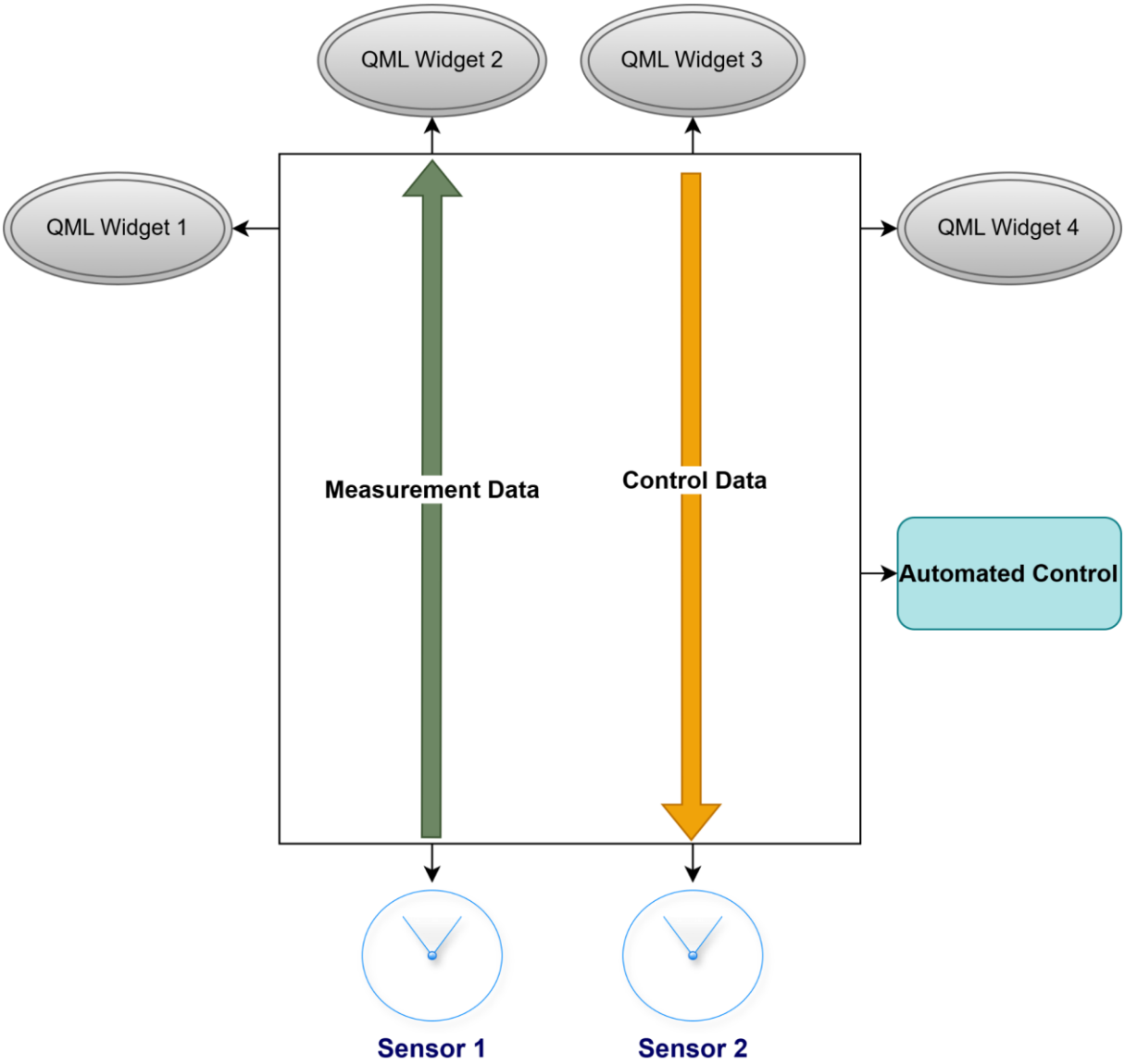

Figure 2: Stem Data Flow Design in a Typical Embedded system

Now let's analyze the Control data flow in Stem Data Flow Design. The control commands flow from the top of the hierarchy to the bottom. The beauty of this design is that the control commands flowing from the top can be vetted or validated as it flows down through the component hierarchy. This can be vetted because the components below in the component hierarchy may have data or conditions to perform the validation. For example, a command from the QML widget such as a user input flows down the component hierarchy. However, due to automated control execution, the backend C++ module can abort or pause the user command.

Another robust factor that Stem Data Flow Design(SDFD) brings to the design is that measurement data and control data are two different tunnels in the Stem Data Flow Design. This means that the

outcome of a vetted command execution is measured and sent through the component hierarchy from button to top keeping the data in the software components up to date and consistent. In other words, Stem Design Flow Design provides high stability, system reliability and robustness.
Stem Data Flow design also provides removal and reusability of components without much impact on the other system components. For example, the automated control component can be removed easily without even disturbing the measurement and control data tunnels in the Stem Data Flow. Also, automated control components can be added to another system with minimal integration effort.

IV. CONCLUSION

The Stem Data Flow Design(SDFD) is a smart approach to the overall architecture of a new embedded system product. This design methodology makes the overall all system design more modularized so that it is easy to add or update new components. Considering robust interfaces this design provides, it is helpful and easy to reuse the software components in multiple projects. As the components are not tightly coupled, an enhancement or a bug fix on one component may not have a large impact on other components. This helps the overall system software to get mature and robust with time. Finally the most important advantage with the Stem Data Flow Design(SDFD) is that it is easy to debug to find the root cause of a problem and provide appropriate fixes compared to just delivering temporary patches and fixes that may break other components.

REFERENCES

1. I-Ling Yen, J. Goluguri, F. Bastani, L. Khan and J. Linn, "A component-based approach for embedded software development," Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002, Washington, DC, USA, 2002, pp. 402-410, doi: 10.1109/ISORC.2002.1003805.
2. Dewayne E. Perry and Alexander L. Wolf. 1992. Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes 17, 4 (Oct. 1992), 40–52.
3. Aoyama, M., 1998, April. New age of software development: How component-based software engineering changes the way of software development. In 1998 International Workshop on CBSE (pp. 1-5).