Dataflow-based Heterogeneous Code Generator for IoT Applications

Gábor Paller[1], Endri Bezati[2], Nebojša Taušan[1], Gábor Farkas[1] and Gábor Élő[1]
[1]*Széchenyi István University, Information Society Research & Education Group, Egyetem tér 1. Győr, Hungary*
[2]*streamgenomics Sarl, EPFL Innovation Park, Bâtiment C, Lausanne, Switzerland*

Keywords: Dataflow, Code Generator, IoT, Eclipse, IDE, Orcc.

Abstract: The latest wave of connected digital systems, nowadays called Internet of Things (IoT) promises various gains, especially in terms of significant ease of data access. A large number of different platforms for IoT applications, however, makes their development difficult and time-consuming leading to projects that are failed to be on time, on budget or fully completed. To address the platform heterogeneity, this paper presents the ongoing work on development environment called Orcc-IoT. Orcc-IoT facilitates the development of IoT by combining dataflow modelling language, heterogeneous code generator and the library of ready-made IoT actors. The utilisation of Orcc-IoT in development is expected to increase the quality, and to reduce the development costs and time-to-market of IoT applications.

1 INTRODUCTION

The adoption of connected digital systems (commonly known as Internet of Things, IoT) is still slow due to their high complexity. Factors contributing to the complexity include security, energy efficiency, immaturity of communication technologies and heterogeneity of platforms. None of the hundreds of different IoT platforms reached the sufficient footprint so that a dominant platform can be identified. This heterogeneity makes application development a costly and time-consuming enterprise with the constant risk that the application must be ported from one platform to another.

The heterogeneous platform problem of IoT applications can also be argued with the findings from the scientific literature. For example, the (Taušan et al., 2016) study reveals heterogeneous platforms as one of the requirements for the development of modelling languages in embedded systems domain. The (Giang et al., 2015) study suggests that there are two dimensions of the platform heterogeneity in IoT application domain. These are:

- vertical - shows the amount of resources the device has. For example a certain device may have more computing power or memory than the other.
- horizontal - shows differences in services provided by the device. For example a certain device may have an embedded A/D converter or a certain network adapter (e.g. WiFi).

An overview of IoT challenges presented in (Gazis et al., 2015) reveals two aspects of device interoperation. The first is technical and denotes how to enable the interoperation of heterogeneous devices. Second is semantic and denotes how to enable the seamless exchange of information among heterogeneous devices. Besides technical and semantic interoperation of heterogeneous devices, the (Yaqoob et al., 2017) study also identifies the pragmatic aspect. The pragmatic aspect concerns with the capability to observe the intention of heterogeneous parties involved in the interoperation.

To address the heterogeneity problem of IoT application development, this paper presents the ongoing work on Integrated Development Environment (IDE) called Orcc-IoT. The Orcc project (Yviquel et al., 2013; Sourceforge, 2014) has started in 2010 and it targets signal-processing applications. Our project, Orcc-IoT builds on Orcc with additional features that are specific for the IoT problem domain. The features include the improvements of the graphical Data-Flow Language (DFL), novel code generator for heterogeneous platforms and the library of ready-made IoT components or actors.

A motivational example for the selection of dataflow paradigm for our DSL is presented in Figure 1. This example shows an Air Handling Unit (AHU) IoT system with sensors and actuators designed with

Paller, G., Bezati, E., Taušan, N., Farkas, G. and Élő, G.
Dataflow-based Heterogeneous Code Generator for IoT Applications.
DOI: 10.5220/0007496804260432
In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*, pages 426-432
ISBN: 978-989-758-358-2

the popular Niagara IoT workbench [1]. The AHU controls supply and return air flows and several characteristics of each flow are measured. The measurements are typically cyclically collected, analysed and actuator values are computed. The calculations often follow the flow of material in the system (in this case, the flow of air).

One example application for the AHU engine is depicted in Figure 2. This shows the flow of data from one processing element to another passing the control among the devices indicating that the dataflow paradigm can be used efficiently for these types of applications.

This flow-like nature of IoT data is indicated in scientific sources. Several realistic use cases in different application domains are presented in (Yasumoto et al., 2016) to argue for the dataflow paradigm. In the domain of high-speed computing, the dataflow design environments are most widely used, according to (Milutinović et al., 2015). Finally, (Giang et al., 2015) use distributed dataflow for their framework where each device executes only part of the full actor network. All these characteristics indicate that the dataflow paradigm can be used efficiently for IoT applications.

An approach alternative to the dataflow model is the component-based model, illustrated here by ThingML (Harrand et al., 2016). The component-based model concentrates on the components, their interfaces with each other and their functional description which is often state machine-based. The component-based model is not able to expose the parallelism existing in the system to the degree provided by the dataflow model. While IoT systems are rarely as demanding with regards to processing performance as signal processing applications which is the main use case for the dataflow model, the extreme scalability required by the large amount of endpoints (e.g. sensors, actuators, etc.) often makes server implementation challenging. Automatic generation of parallel implementation facilitated by the dataflow model may simplify these server implementations significantly.

This paper is structured as follows. Section 2 describes the base system we chose for our dataflow-based IoT design environment and how we propose to extend it. Section 3 discusses a key component of the extended system, the heterogeneous dataflow compiler. Conclusions and further work are proposed in section 4.

[1] https://www.tridium.com/

2 Orcc IoT EXTENSIONS

Today, there are literally hundreds of IoT platforms proposed. These software stacks offer data ingestion/sending from/to the endpoints, data management, visualisation and device management features, including alarms. Some are offered as installable software packages, some are cloud services. So far no dominant player emerged in the field, making application development a risky investment.

The approach to IoT application development, presented in this paper, is to take a high-level application model and generate code from this model for the target platforms. Following the considerations presented in section 1, we have chosen the dataflow paradigm as high-level application model and developed the DFL to represent the applications.

Instead of developing the environment from scratch, existing dataflow/component design environments were analysed. The following environments were studied: Node-RED, Internet of Things Workbench (IBM), Spacebrew, TASTE, Caméléon, Vorto, ANKHOR FlowSheet and Orcc. The analysis resulted in the selection of the open RVC-CAL Compiler (Orcc) (Yviquel et al., 2013; Sourceforge, 2014) as base development environment. The main advantages of Orcc over the evaluated systems are the following:

- It is completely open-source under the BSD license.
- Its compiler backends are able to generate code for a wide variety of targets from FPGAs to relatively high-end runtimes like Java. Orcc does not enforce any mandatory runtime unlike systems like Node-RED enforces web runtime. [2].
- It is based on the Eclipse platform that makes it highly extensible.
- Its data formats have been standardised (ISO/IEC, 2012). This standard describes the dataflow network's representation in XML format that Orcc implements.

After having selected Orcc as base system, a gap analysis was performed among the features required for the IoT dataflow and those offered by Orcc. The most relevant gaps were the following:

- Orcc backend code generation is strictly homogeneous. When code is generated for an application dataflow network, one backend is selected and all the code is generated by that backend. This does not satisfy the heterogeneity requirement of IoT systems. The IoT version of Orcc must be able

[2] https://nodered.org/

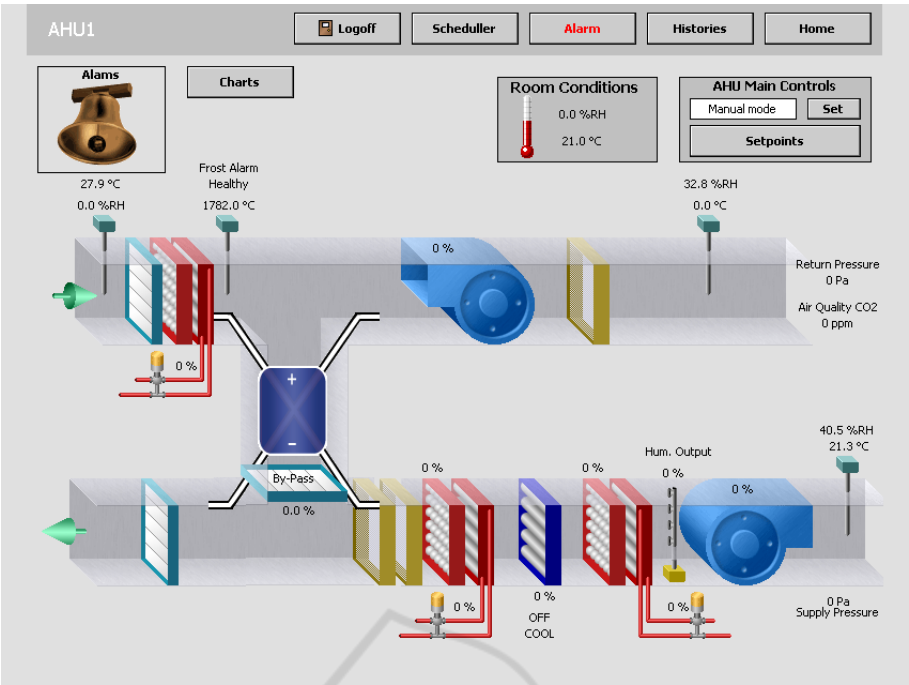

Figure 1: Air Handling Unit IoT system designed with Tridium Niagara (source: CC North Ltd. ©2015).

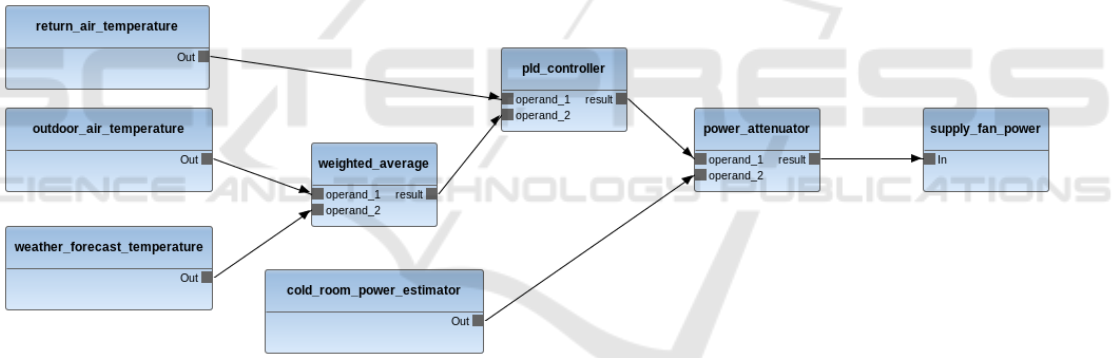

Figure 2: Example dataflow application for the AHU unit.

to assign parts of the dataflow graph to different backend code generators and generate code for each IoT system elements.

- The current public Orcc version is very strong in code generation for low-end devices like FPGAs or embedded computers.This is the result of dataflow paradigm's strong focus on high-speed computing. IoT system components include server backends, with a different set of programming languages. Orcc used to have a Java backend but it was obsoleted. We plan to include into the IoT version a Java backend with server flavour like Spring Boot [3].
- An Orcc application currently must define all the actors in the graph. The requirement is that IoT system designers be able to select a wide variety of actors so that IoT applications can be composed quickly. This requires an actor library support.

[3]https://spring.io/projects/spring-boot

3 HETEROGENEOUS CODE GENERATION FOR IoT

The heterogeneous compiler is the main difference between the traditional dataflow applications optimized for high-speed computing and the IoT applications. Heterogeneous dataflow has been a major research topic (e.g. (Bezati et al., 2014)) but prior research mainly concentrates on multiple processing

cores (of same type like multicore or of different types as CPU-GPU) integrated into the same computing hardware, often on the same chip. As discussed in section 1, IoT applications are inherently heterogeneous and distributed with communication links that can be unreliable (like short-range communication links) and/or slow (like low-power WANs).

Figure 3 demonstrates, how the heterogeneous code generator backend fits into the Orcc architecture. The heterogeneous backend knows, which parts of the dataflow application are assigned to which homogeneous backend, partitions the graph and hands parts of the graph to their associated homogeneous code generators.

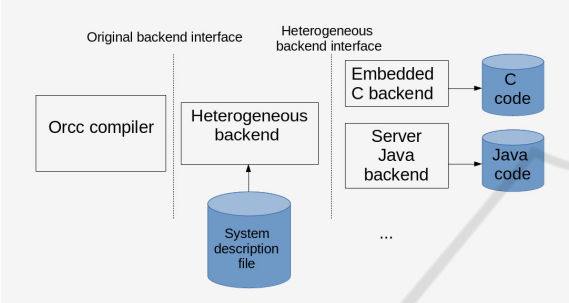

Figure 3: Heterogeneous code generator architecture.

3.1 Architecture

The basic principle of a heterogeneous platform is that they contain multiple processing elements with different kinds of computational units. We can describe the architecture of an heterogeneous platform with three components: *Processing element (PE)*, *Medium* and *Link*.

A PE models the processing element type, it is used to describe the main properties of a computational unit available on a given platform. A PE may refer to any element in the IoT architecture that is capable of being part of the dataflow network, i.e. can produce or consume data items (tokens in dataflow parlance). A PE therefore can be a sensor that can produce tokens, an embedded computer or IoT application gateway that can run part of the dataflow network or a server instance.

A Medium models a communication channel or a memory, it is used to describe the kind of communication that a PE communicates with another PE or an external communication channel. A Link describes the properties of the interface between a PE and a Medium.

A PE might be represented as nested PE in order to represent a multi-core or many-cores computing device. As an example the architecture of the

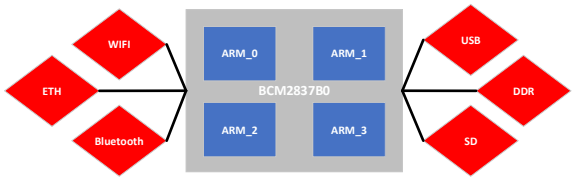

Figure 4: Architecture of the Raspberry Pi 3B+.

Rasberry Pi B+ is depicted in Figure 4. A Rasberry Pi B+ contains a Broadcom A-53 ARM processor (BCM2837B0) with four processing units and variety of media that can be interconnected with it. In addition, the architecture on Figure 4 represents the device as a standalone platform without any other platform connected to it. Furthermore, this architecture is sufficient for describing a Raspberry Pi that runs an operating system that controls all the interfaces and it provides four processing cores. A dataflow application can map actors to each ARM core.

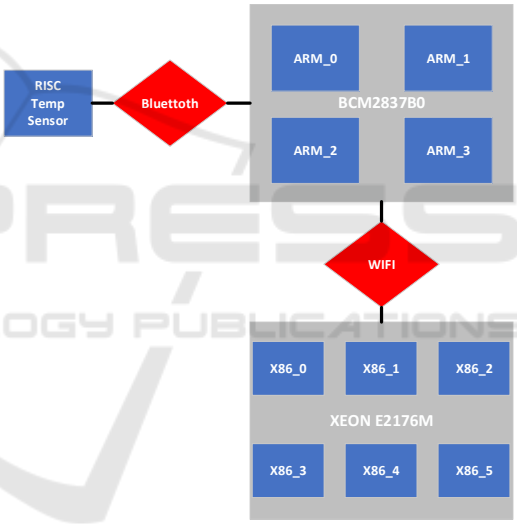

Figure 5: An heterogeneous architecture thats contains a server and an embedded device that receives the temperature data.

Figure 5 depicts an heterogeneous platform composed of a server that contains a Xeon processor with six PEs, an embedded Raspberry Pi that communicates with the server via WiFi and a RISC IOT device that captures the ambient temperature and sends its data to the Raspberry Pi through Bluetooth. The media present in Figure 4 but not used by this dataflow application is omitted in Figure 5.

Using this architecture representation the dataflow application will map its actors to each processing element and the actor FIFOs are mapped to links which are connected to the defined media.

3.2 Mapping Configuration

Orcc supports the mapping of actors to different PEs on a platform that supports POSIX threads for the C backend, but it does not provide a mechanism to represent the architecture of a platform or to generate code for heterogeneous platforms. The current mapping configuration in Orcc is called XCF for XML configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <Partitioning>
    <Partition id="1">
      <Instance id="source"/>
    </Partition>
    <Partition id="2">
      <Instance id="sink"/>
    </Partition>
  </Partitioning>
</Configuration>
```

Figure 6: Example XCF of ORCC C backend file with two partitions.

The XCF represented in Figure 6 indicates that the generated code has to place actor ***source*** to partition ***1*** and actor ***sink*** to partition ***2***. In the context of the Orcc C backend, it means that two threads on system that supports POSIX threads are going to be created and the actors ***source*** and ***sink*** are going to run in parallel. Furthermore this format does not specify which network is being used for this dataflow application.

This format in its current form can only be used for assigning runtime actors to different threads. Instead of using XCF just for partitioning, here we propose an extended version of the XCF format that can be used as input for Orcc to generate source code for heterogeneous platforms and can inherently describe the architecture of a platform.

As it can be seen, the XCF configuration file in its current format can not be used for heterogeneous computing and therefore the format has to be extended. First, the XCF needs to describe which is the top **Network** of the application that is going to be used. Thus, the XCF is coupled with a dataflow network or even multiple such networks. Second, it needs to give explicit information about the **Partitioning** of the application to an heterogeneous platform. The XCF file also needs to describe which PE this partition is going to be associated to and the backend that is going to be used to generate the code.

The XCF file also includes the notion of a host. A host can describe a platform that contains different PEs with the same processing core architecture (e.g a PC with multicore Intel processor) or different

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <Network file-format="xdf" qualified-id="AHU_Unit"/>
  <Partitioning>
    <Partition id="server_0" pe="x86_0" backend-id="Java Server" host="
        intel_server">
      <Instance id="weather_forecast_temperature"/>
      <Instance id="weighted_average"/>
      <Instance id="power_attenuator"/>
      <Instance id="cold_room_power_estimator"/>
    </Partition>
    <Partition id="server_1" pe="x86_1" backend-id="Java Server" host="
        intel_server">
      <Instance id="pid_controller"/>
    </Partition>
    <Partition id="air_temp_0" pe="ARM" backend-id="Java 0">
      <Instance id="return_air_temperature"/>
    </Partition>
    <Partition id="air_temp_1" pe="ARM" backend-id="Java 0">
      <Instance id="outdoor_air_temperature"/>
    </Partition>
    <Partition id="fan_power" pe="ARM" backend-id="Java 1">
      <Instance id="supply_fan_power"/>
    </Partition>
  </Partitioning>
  <Backends>
    <Backend id="Java Server" backend="Java Server"/>
    <Backend id="Java 0" backend="Java Vanilla">
      <Parameter key="Board" value="BeagleBone Black Wireless"/>
    </Backend>
    <Backend id="Java 1" backend="Java Vanilla">
      <Parameter key="Board" value="BeagleBone Black Wireless"/>
    </Backend>
  </Backends>
  <Media>
    <Interface id="wifi" medium="Wifi">
      <Parameter key="Server IP Address" value="192.168.0.1"/>
    </Interface>
  </Media>
  <Connections>
    <Fifo-Connection src="return_air_temperature" src-port="Out" dst="
        pid_controller" dst-port="operand_1" size="512" medium-id="
        wifi"/>
    <Fifo-Connection src="weighted_average" src-port="Out" dst="
        pid_controller" dst-port="operand_2" size="512" medium-id="
        wifi"/>
    <Fifo-Connection src="pid_controller" src-port="Out" dst="
        power_attenuator" dst-port="operand_1" size="512" medium-
        id="wifi"/>
    <Fifo-Connection src="outdoor_air_temperature" src-port="Out" dst="
        weighted_average" dst-port="operand_1" size="512" medium-
        id="wifi"/>
    <Fifo-Connection src="power_attenuator" src-port="result" dst="
        supply_fan_power" dst-port="In" size="512" medium-id="wifi
        "/>
  </Connections>
</Configuration>
```

Figure 7: AHU XCF file.

ones (e.g. the Zynq 7000 is composed with two ARM processors and an FPGA). Thirdly, it needs to provide a set of **Backends** and how they are configured. Fourthly, it needs to describe the different **Media** used for this configuration and its attributes. And finally, it needs to describe the **Connections** between the PE and the Media.

Figure 7 represents the XCF file of the AHU unit dataflow application of Figure 2. The dataflow application is mapped onto 5 partitions: *server_0*, *server_1*, *air_temp_0*, *air_temp_1* and *fan_power*. The *server_0* contains four instantiated actors and server_1 contains one instantiated actor. Both those partitions run in parallel on the same machine as indicated by the host *intel_server*. All other partitions run in different embedded platforms and contains one instanti-

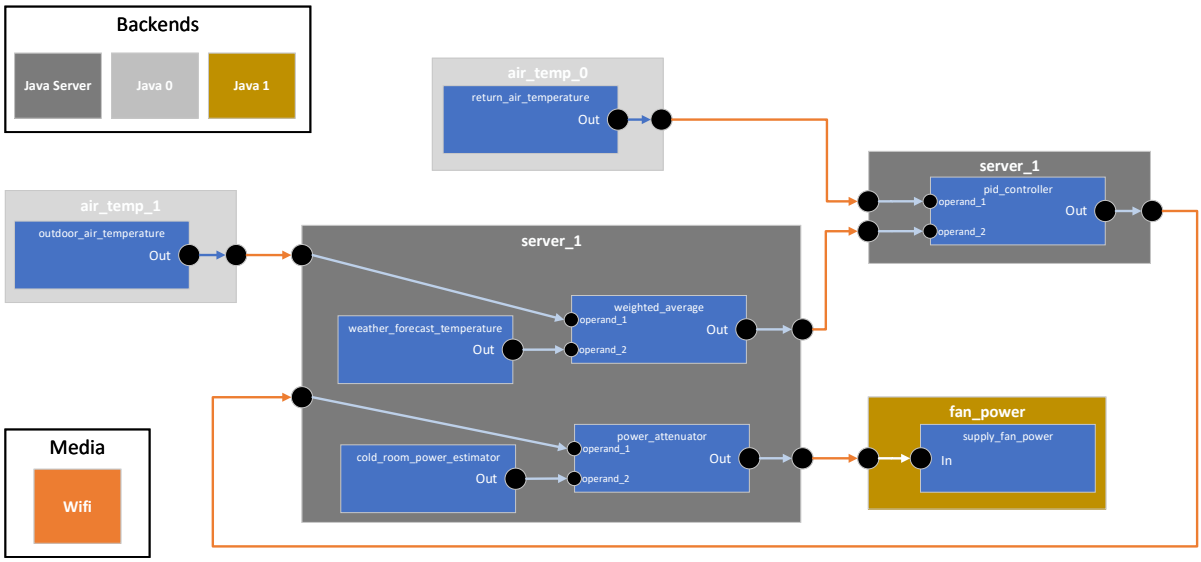

Figure 8: Visualization of the XCF defined in Figure 7.

ated actor. Three backends have been defined: one for Java server-based applications and two for plain java code generation. Here we should mention that the proposed XCF format permits the same backend with different parameters such as the backends with ids *Java 0* and *Java 1*. All the PEs communicate through one medium called "Wifi". The ip address of this medium can be passed as a parameter, as indicated in line 34 of Figure 7. The three embedded devices are connected to the server via FIFOs under the medium "Wifi", as defined in lines 40, 41 and 42 of Figure 7. Finally, the connections of the *result* port of *weighted_average* and *result* port of *pid_controller* are omitted and should not be defined in the XCF because there are internal FIFOs and *server_0* and *server_1* partitions are Java threads that run in parallel.

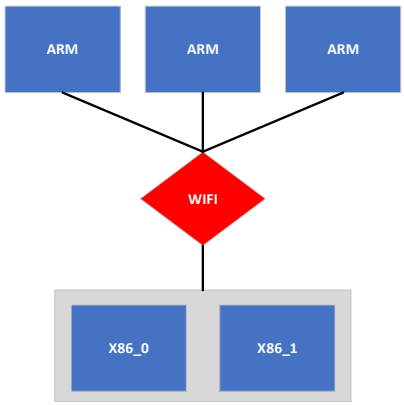

Figure 9: V the XCF defined in Figure 7.

Figure 8, visualizes the XCF of the Figure 7. The five partitions are subnetworks of the original network of Figure 2. Three backends have been defined with different colors, every parition is colored with a correspoding backend. For this example only one Media has been used. The same color as the Media has been used for depicting the interconnection among the partitions. Finally, given the proposed XCF file structure, the architecture of a platform can be extracted directly as depicted in Figure 9. Thus, this extended XCF file describes how different heterogeneous platforms are interconnected and configured but also represents the architecture of the overall heterogeneous systems.

4 CONCLUSIONS AND FURTHER WORK

This paper presented an ongoing work to extend an open-source dataflow environment with IoT features. We argue that the dataflow application model suits a wide variety of IoT systems, providing a solution to the IoT heterogeneity platform problem, bridging the gap between the physical engineering system (like an industrial process) and its associated IoT process chain and even improving the security. There are some differences, however, between the usage of dataflow model in its established, high-speed computing use cases and the proposed IoT use cases. The most important difference is the inherent heterogeneity of IoT systems and the presence of the short-range and wide-area network links that come with this heterogeneity. Our proposed extensions address this issue.

The feasibility of Orcc-IoT's dataflow approach

and heterogeneous code generation for IoT will be demoed in an air conditioning system for aircraft simulators. Orcc-IoT will be published as open-source software under the original Orcc license (BSD).

REFERENCES

Bezati, E., Thavot, R., and et al., G. R. (2014). High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms. *J Real-Time Image Proc (2014) 9: 251*.

Gazis, V., Goertz, M., Huber, M., Leonardi, A., Mathioudakis, K., Wiesmaier, A., and Zeiger, F. (2015). Short paper: Iot: Challenges, projects, architectures. In *2015 18th International Conference on Intelligence in Next Generation Networks*, pages 145–147.

Giang, N. K., Blackstock, M., Lea, R., and Leung, V. C. (2015). Developing iot applications in the fog: A distributed dataflow approach. In *2015 5th International Conference on the Internet of Things (IOT), 2015 Oct., Seoul, South Korea*. IEEE.

Harrand, N., Fleurey, F., Morin, B., and Husa, K. E. (2016). Thingml: A language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, pages 125–135, New York, NY, USA. ACM.

ISO/IEC (2012). *ISO/IEC 23001-4, Information technology MPEG systems technologies Part 4: Codec configuration representation*.

Milutinović, V., Salom, J., Trifunovic, N., and Giorgi, R. (2015). *Guide to DataFlow Supercomputing: Basic Concepts, Case Studies, and a Detailed Example*. Springer.

Sourceforge (2014). Orcc : Dataflow programming made easy, http://orcc.sourceforge.net/.

Taušan, N., Markkula, J., Kuvaja, P., and Oivo, M. (2016). Choreography modelling in embedded systems domain: Requirements and implementation technologies. In *4th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2016*, pages 75–86, Rome. IEEE.

Yaqoob, I., Ahmed, E., Hashem, I. A. T., Ahmed, A. I. A., Gani, A., Imran, M., and Guizani, M. (2017). Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wireless Communications*, 24(3):10–16.

Yasumoto, K., Yamaguchi, H., and Shigeno, H. (2016). Survey of real-time processing technologies of iot data streams. In *Journal of Information Processing Vol.24 No.2 195202 (Mar. 2016)*. IPSJ.

Yviquel, H., Lorence, A., Jerbi, K., Cocherel, G., Sanchez, A., and Raulet, M. (2013). Orcc: Multimedia development made easy. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 863–866, New York, NY, USA. ACM.