Furthering the Goals of Multivendor Interoperability in ORAN: From Interfaces to Abstraction and Automation

Cirrus360 Corporation, Intel® Corporation, Vodafone

1. Executive Summary

O-RAN Alliance's charter is to transform Radio Access Networks (RAN) towards open, intelligent, virtualized and fully interoperable RAN.

In this paper, we introduce a **framework consisting of a domain specific language (DSL) to formally describe use cases, constraints, and multi-vendor hardware/software abstraction, supported by intelligent automation.** It captures the capabilities of the vendor components (silicon or software) and interfaces at a level of abstraction relevant to meeting RAN requirements. It also allows the deployment use cases and constraints to be formally represented. Automation is then applied to all these inputs to match, integrate, and optimize the requirements on the target hardware at the system integration level.

The framework we propose addresses efficient multivendor interoperability and deployment specific optimizations. It supports intelligent automation of all components of RAN and enables the goal of O-RAN Alliance.

This is achieved via **intelligent automation** and an **open-source RAN domain specific language (like Cirrus360 RDSL™ [1])** to facilitate interoperability, that is flexible enough to encompass a wide variety of use cases across vendors and industries. This extends the current O-RAN framework by advancing two important goals: software hardware disaggregation via abstraction and enabling a new level of control of multi-vendor components in the process of system integration.

Rapid proliferation of O-RAN deployments is driving a wide ecosystem of Software and Hardware vendors providing various open RAN solutions, delivering choice and flexibility for Operators. One of the challenges of this wider ecosystem is multivendor integration and optimization across a wide variety of use cases and traffic scenarios. These challenges are currently being addressed in many ways from open-source software solutions to standard interfaces from the O-RAN Alliance itself and have made progress towards delivering the Open RAN vision of an open disaggregated RAN ecosystem. Continuing this journey will require a new level of abstraction beyond simply interfaces and towards automated platforms that optimize for system level requirements as well as integrating multi-vendor components, as explained in the Vodafone whitepaper on System Integration [1], further reducing the cost and time required to introduce the latest features and technology to the O-RAN ecosystem.

Additionally, the enablement of the RAN Intelligent Controller (RIC) interface is a critical goal for O-RAN adoption. Unlocking the full potential of the RIC will require flexible insertion of new functionality on both sides of the RIC interface, and this will also benefit software hardware disaggregation and a more flexible system integration platform.

The framework introduced in this paper speeds up system integration, reducing the integration challenges of O-RAN's wide and diverse ecosystem by enabling a common language (RDSL™), that abstracts the interactions across the O-RAN interfaces into system and hardware constraints. Intelligent Automation can then be used to meet the system level KPI (Key Performance Indicator) and drive a robust deployable solution.

[1] Please reach out to Cirrus360 team (chaitali@cirrus3sixty.com) for a description and examples of RDSL™. RDSL™ has been defined by Cirrus360 and will be open-sourced in the near future.

2. Next Steps

The goal of this paper is to seek the support of operators and ecosystem partners behind the principle that ORAN must compete with highly integrated, single vendor solutions. We then posit that open box solutions based on a Domain Specific Language for description, with software/hardware disaggregation driven by automation, are a necessary step for the ORAN community to take to facilitate this goal. From there we can jointly work towards standardizing the abstraction concept as an essential element to enable such disaggregation and automation. The next steps in this process include:

1. Jointly developing a work item to be introduced to the ORAN Alliance.
2. Continue to collaborate on development of an Open DSL (such as RDSL™) to drive standardization within ORAN
3. Open-source basic tool chains, example RDSL code, formats for hardware discovery and system constraints files, and requirements to enable middleware adaptation as described in the Appendix in Section 7. Develop a community around the open box concept.

3. Challenges addressed and solution summary

The open RAN ecosystem and the O-RAN Alliance has made tremendous strides toward a more open and disaggregated Radio Access Network (RAN) through the introduction and detailed definition of interfaces that allow multiple vendors' products to communicate in a plug-and-play manner. This disaggregation of the RAN as shown in Figure 1 [2] opens new opportunities for new and existing vendors to innovate and provide cost effective solutions for a growing set of new use cases and markets.

However, the journey toward disaggregation and automation is not complete. This has become increasingly clear as open RAN deployments have progressed in the last two years [3]. In this whitepaper we outline the challenges now facing open RAN proponents and provide a path to solving these problems using a process of Intelligent Automation that goes beyond just Interface Definitions.

We argue that meeting the challenges outlined below requires a common language supported by automation that is flexible enough to encompass a wide variety of use cases across vendors and industries. In keeping with the goals of the O-RAN Alliance, the challenges we address in this paper are not related to radio performance such as cell capacity and massive MIMO algorithms. We focus on system implementation challenges of the RAN O-DU, as well as the O-RU and O-CU, such as hardware software disaggregation, power and cost, upgradability, software release maintenance and so on.

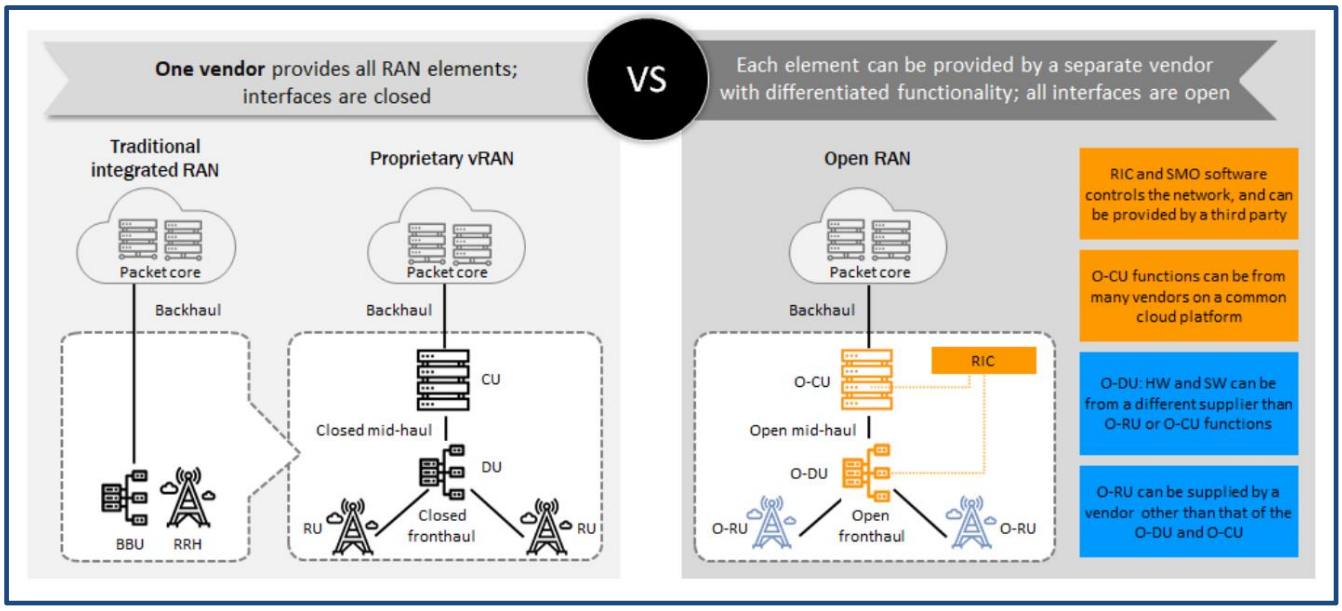

Figure 1: Open RAN system integration challenges. Graphic from 0[2]

The paper is organized as follows. We first summarize **four important future topics** for the O-RAN Alliance to address to increase the competitiveness of RAN deployments. We then summarize the **three technology pillars** that we need to develop to address these challenges. The remainder of the paper adds more details to the topics and technology pillars and describes how the technologies address the topics.

1. Future Topics that the O-RAN Alliance Must Address:

A. Seamless multi-vendor interoperability meeting deployment goals and enforcing system constraints in addition to open interface API
B. Rapid innovation by reducing cost of onboarding new and upgrade of existing solutions
C. More efficient usage of RAN hardware, leading to lower TCO.
D. Simplify System Level optimization via Automation

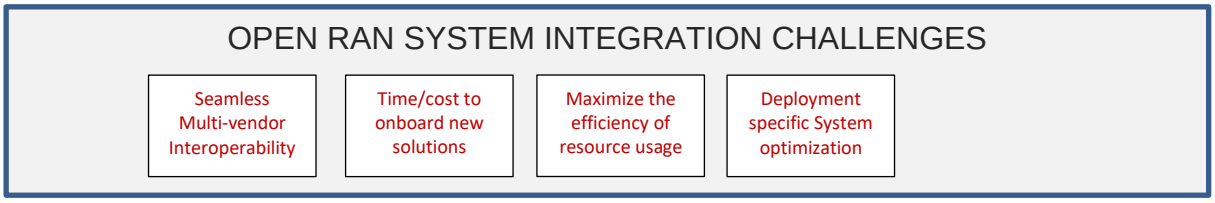

Figure 2: Open RAN system integration challenges

2. Three Pillars for a Solution for Automated Integration in ORAN:

1. ORAN Common Language for RAN requirements and IP agnostic abstraction of components
2. Automated and Explainable Optimization of CAPEX and OPEX
3. RIC level Machine Learned patterns used in automation

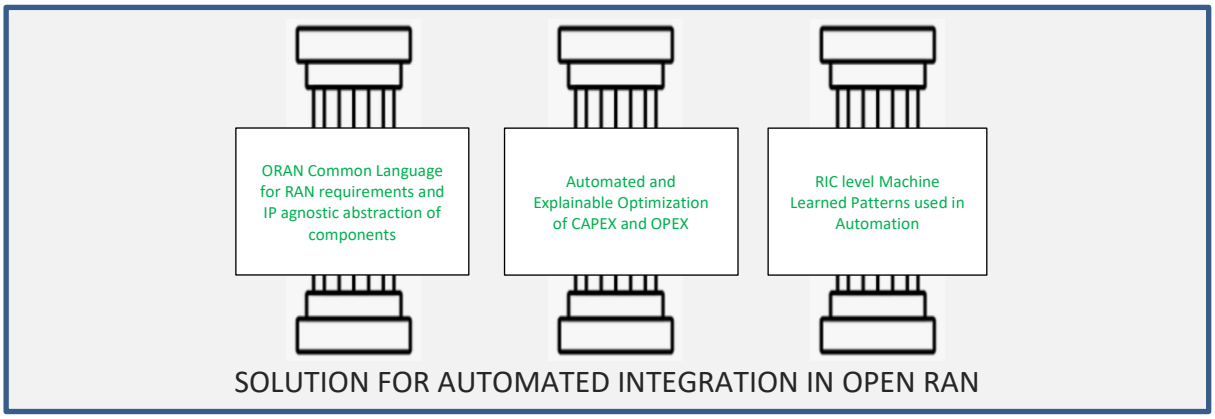

Figure 3: Three pillars for a solution for automated integration in Open RAN

Having identified these requirements and solutions, Cirrus360, Intel and Vodafone are developing technology in line with the three pillars. We currently have demonstrations of all pillars at some level of maturity. The critical challenges we outline are now becoming clear threats to the progress of open RAN. The time is right to engage an industry wide effort to address them as we describe, including making sure the ORAN Alliance puts the proper standardization in place to allow industry traction and acceleration. We invite all operators to engage in this process.

4. The Four Future Topics ORAN Must Address to Simplify Multi-vendor System Integration

We identify the main objectives that when achieved an operator, or enterprise integrator can enhance and optimize a multivendor solution in their network.

3. Seamless multi-vendor interoperability meeting deployment goals and enforcing system constraints in addition to open interface API

From mid 4G onwards RAN requirements have become increasingly complicated due to the addition of data applications. 5G has increased this complexity significantly with the addition of explicit classes of applications and the complexity will continue to grow rapidly through the life of 5G.

Complex and multifaceted requirements make it difficult to identify worst case scenarios in pre-deployment lab testing. In practice, intermittent errors in the field, called Heisenbugs, will appear, often in situations that are not particularly loaded. A common root cause is some combination of new user classes and sudden bursts of new user requests while others are at critical set up or tear down points. This means that simply testing highly loaded scenarios in the lab will not provide a robust solution in the field. A more formal methodology is required to describe allowed scenarios and to describe how they will interact. Even when a fault does not lead to a system crash it may both increase packet loss rates and cause violation of other user requirements such as latency, generally decreasing the QoS of the network.

All of the above are general problems for single vendor networks and become more difficult to deal with in multi-vendor networks [3]. In this white paper we address how to manage the issue though Open RAN with open standard interfaces and disaggregated hardware and software.

4. Rapid innovation by reducing cost of onboarding new and upgrade of existing solutions

In a multivendor environment onboarding any new hardware or software requires the integration of multiple vendor IP. A lack of a formal description of system level integration requirements leads to system level

incompatibilities between components, even if their interfaces are compliant. Individual vendor feature differentiation, one of the strengths of O-RAN, can exacerbate this problem. A Software based Open RAN Solution that allows for Continuous Development and Continuous Integration can mitigate this problem if it is integrated at the system level with a formal definition of requirements. Otherwise, the System Integrator will need to be closely involved in compatibility issues between vendor components, increasing Time To Market.

5. More Efficient usage of RAN hardware, leading to higher TCO RAN resource

The increasing diversity of applications supported in 5G provides room for optimization of OPEX and CAPEX in specific deployments. Small improvements in each site's deployment can lead to large savings at the network level. However, this requires deployment sensitive optimizations that are based on knowledge of the use of the RAN in the network. ORAN has addressed this issue with an external function, the RIC, which can monitor the rest of the RAN and provide optimization parameters. To do so the RIC app often needs new functionality to be added to the DU/RU/CU it is monitoring to collate and groom the analytics the RIC app needs. Additionally, the DU/RU/CU vendor may not have supplied the hooks necessary for a RIC app to manage the DU/RU/CU resources. So being able to add new analytics functionality and control ability to an existing vendor component, without disruption of its function is an important topic. This requires an open box component approach with a formal language abstracted modelling of the component at the system level.

6. Simplify System Level optimization via Automation

Effective performance feedback requires Key Performance Indicators (KPIs) that can be collected and used, often within the RIC, to improve some aspect of the performance of the RAN. In a well-maintained network with evolving use cases KPIs are constantly being discovered and many are deployment dependent. For lifecycle maintenance and improvement, the operator needs to be able to add and subtract (because they also take resources) KPI collection functionality from DU, RU and CU deployments. This requires an open box component approach with a formal language abstracted modelling of the component at the system level.

5. Three Pillars for Automated Integration in ORAN

7. ORAN Common Language for RAN requirements and IP agnostic abstraction of components

This pillar addresses the need to be able to construct a complete RAN solution from multiple vendor IP using operator and deployment specific constraints. We propose a simple Domain Specific Language which we call the RAN Domain Specific Language (RDSL™) to achieve this goal.

Requirements for the RDSL™ include:

- The construction of the RAN must be described in a manner that is precise and unambiguous.
- The description must be hardware agnostic as much as is possible to allow the same RDSL™ application to be ported to different hardware SKUs of the same platform or even different platforms.
- The RDSL™ must allow for an automated implementation on the target platform. An abstract platform description and system constraint parameters are required. The system constraint parameters define constraints on the use of the hardware. One important example of system constraints would be timing of input and output of the platform allowing properly timed hook up of the DU to the RU. System parameters can also allow definition of memory use by the IP blocks so that efficient mapping of IP to hardware can be automatically optimized

To achieve these goals, we propose a declarative and immutable language. This is common in DSL [4] and allows target agnostic exposure of potential parallelism in both compute, memory, and interface resources in hardware. To this common framework we add an explicit awareness of time that is similar to that used in reactive systems such as Lingua Franca [5], but has been adapted to the specific needs of the Periodic Firm Real Time Systems that characterize RAN at CU, DU, and RU levels.

Such a description also precisely defines all of the worst-case paths through the construction allowing an automated tool to choose the best solution for a particular deployment on a particular target platform. System constraints are used to bound the construction to a feasible solution given memory sizes, system timings and compute resources. The RDSL™ we propose leads to a highly redundant description of the system where many possible corner cases are exposed. Because the description is declarative (by which we mean it describes the requirement but not how to implement it) and also immutable (by which we mean there are no time varying values in the description) an automated tool can safely reason about the best implementation. Though this may sound like a complicated programming paradigm, we have shown that the resulting language is simple and intuitive because we add an explicit definition of time and periodicity for the RAN functionality.

The declarative nature of the RDSL™ allows multiple sources of code from multiple vendors to be combined because these sources do not describe an implementation but only intent. The multiple sources can then be jointly optimized for implementation on a given platform.

In order to implement the constructive description of the RDSL™ on a platform we need an abstract description of the platform. We believe the current effort in O-RAN WG6 to define extensions to the AAL will allow for a suitable hardware abstracted platform description. So RDSL™ is very synergistic with the current O-RAN efforts and can be seen as the “missing piece” to allow full automation of RAN system integration.
To abstract the system, we break down the platform into high level components such as accelerator, processor, memory subsystem, interface and so on. We allow the platform provider to decide on the level of granularity with which to abstract their system and only require a simple constructive description language to show how these components hook together in terms of their real time usage requirements. In addition, we recommend a low complexity and high reliability scheduler for mapping functionality to the abstracted components, such as those commonly used in safety critical systems [6]. This allows cloud based offline analysis to be used to automate, for example,

- how functionality is mapped to processors and accelerators.
- how data management and data structure are organized to minimize power.
- finding optimal schedules that remove heisenbugs.
- minimizing the resources required to achieve the deployment goals.

We also use the system constraints of the RAN to define performance of functional blocks on the hardware component IP. RAN functionality vendors only need to publish timing and resource use requirements for their functions and these can be easily tested in abstraction without revealing the details of this vendor IP. Multiple vendors can now compete. The operator will identify functional components that meet the functional performance requirements of the RAN (for instance baseline SNR, dropped call probability etc). Improving system performance (for instance maximum number of users supported in the DU, latency of uplink decode, power requirements for a given traffic model) can be automated by allowing automation of the decision process of which available functional blocks can be used and how they should be connected.

In general, the operator can use automation to construct the full RAN system to meet the network *system performance* requirements given the components available to the RAN in a fraction of the time and cost of traditional manual methods. The disaggregation of functional performance and system performance is critical to the automation of RAN development.

8. Automated and Explainable Optimization of CAPEX and OPEX

In this section we describe how automation can be employed to explore rapidly the space of potential RAN solutions by automating construction of the real time components (DU, CU and potentially RU) from collections of simple unit functions. This exploration can be performed to optimize for one or more soft

constraints such as power, latency, resources used and so on. This is a new manifestation of automation in the RAN as it is for integration of the DU/CU/RU to meet the particular goals of a deployment. It is a challenging problem because of the need to implement a high availability real time solution. The operator can employ such a tool, to explore the performance of a particular RAN for different deployments without needing to understand the details of the hardware or be an expert in embedded system development. Results for differently constrained synthesis runs will allow optimized solutions for different RAN deployments, such as urban versus rural, or factory versus suburban. The operator can focus on exploring tradeoffs that maximize the overall network deployment rather than having to choose from one of a small number of fixed designs that may not provide a good solution at the network level. As the RAN lower layers represent a majority of the OPEX and CAPEX costs of deployments, automation of construction of these components in the most efficient way is therefore a critical aspect of the success of ORAN in providing fine grained analysis and synthesis for specific deployment scenarios. We have developed and demonstrated such a tool on RAN deployment examples.

Changes to the RAN deployment is traditionally slowed due to concerns about heisenbugs and timing failures as an unintended consequence of changing runtimes and resource use of IP within the RAN. With automation any of these kinds of failures, even if they are not visible in test and verification runs, are flagged by the automation tool, allowing the system integrator to adjust the system deployment constraints or add mitigation strategies for any rare corner cases. With this assurance in place the system integrator can become more aggressive in improving the system level performance of the RAN.

Automation will also allow for a continuous tracking and searching for heisenbugs and timing failures in the RAN. These can be fixed as they are found. We discuss this more in the Machine Learning section of this paper. Automation of the process allows for an abstract digital twin of the RAN deployment to be created. Any new constraints or changes in understanding of the RAN environment can be fed back into this digital twin and a more optimal and performant RAN solution produced for deployment. The deployment could also be automated using a standard Container as a Service (CaaS) methodology so that the RAN could continue to learn its environment.

If the automation tool does not find a feasible solution given the system constraints provided by the operator, automated techniques can be applied to provide explanations for why the RAN will not work correctly with the current requirements and platform. These can be used by the operator or their chosen system integrator to make intelligent decisions on how to modify the system, either by relaxing the system requirements, reducing capacity, or adding hardware. In our experience, much of the art of system integration is in deciding what to do in the event that too much is asked of the available resources. Automation of explainability is therefore a critical tool for ORAN.

9. RIC level Machine Learned patterns used in automation

One source of new constraints for the automation is from machine learning and analytics in the RIC. Currently new apps in the RIC are constrained by the available analytics from the blackbox DU [7] as shown in Figure 4 (a). This limits innovation and diversity of vendor for RIC apps [7]. Our proposal unlocks the value of the RIC by enabling the addition of new analytics modules in the DU/RU/CU to support new RIC app capability as shown in Figure 4 (b). As the construction and functionality of the RAN is defined in RDSL™ the operator can add RIC enabling modules into the RAN as they see fit and can switch on and off monitors in the RAN that feed the RIC with data for analysis. The RIC in turn provides the automation tool with updated environment constraints and these are used to re-optimize the solution automatically. So, automation of RAN deployment becomes a key enabler of the value of the RIC and therefore part of the key value chain of ORAN itself.

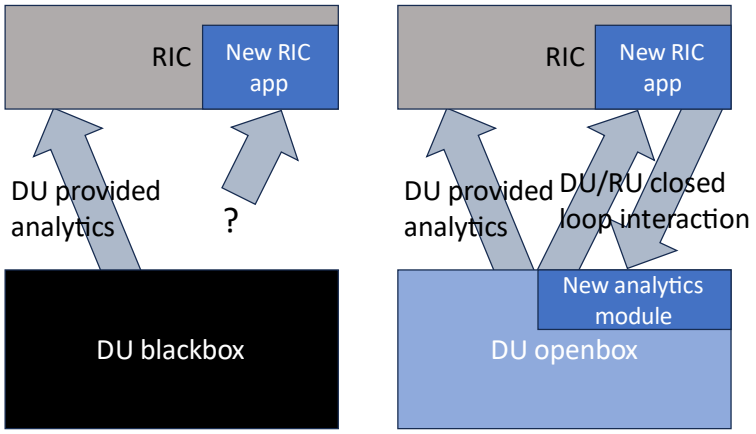

Figure 4: Evolution of RIC support with openbox DU

6. Addressing the Objectives with the Three Pillars

In this section we summarize how the three pillars of the solution can address the four topics. This summary therefore provides a simple motivation for the technology described in the three pillars.

10. Seamless multi-vendor interoperability meeting deployment goals and enforcing system constraints in addition to open interface API

Automation of the construction of the RAN for a specific deployment, with IP abstracted components, allows a formal, abstracted digital twin description of the RAN that can be searched and analyzed for the heisenbugs and timing errors that lead to performance loss and system failure. Failures observed in the field can be used to update the model for improved analysis. Explainability can be used to decide how to deal with automated fixes if they contradict performance goals. Many critical issues that currently require a red team from the vendor (now multi-vendor in ORAN making this an even more complicated effort) to fly in and deal with, can now be dealt with directly an immediately by the operator using automated analysis and explainability.

11. Rapid innovation by reducing cost of onboarding new and upgrade of existing solutions

Onboarding and upgrading are automated, leading to a dramatic reduction in cost and time. Machine Learning provides constant improvement of the model and therefore continuous upgrade opportunities can be identified. Abstraction of IP allows multivendor IP to be quickly swapped into the system to improve functional performance or to add new performance features.

12. More Efficient usage of RAN hardware, leading to lower TCO

RDSL™ allows for a precise description of the OPEX and CAPEX goals and automation can be used to continuously strive for the optimal OPEX and CAPEX even as traffic and features evolve over time. Explainability allows the operator to understand the tradeoffs of OPEX and CAPEX versus algorithmic improvements. A small improvement in channel capacity may be delayed due to its detrimental impact on CAPEX for instance. All this can be analyzed automatically and presented in an explainable manner at the appropriate level within the operator decision making process.

13. Simplify System Level optimization via Automation

The abstract model of the RAN allows changes to be made to components that are currently black boxes in the ORAN model. These changes are automated while still supporting multivendor IP. Performance issues

observed in the field can be analyzed through automation on the digital twin and new deployments can be quickly decided on. These deployments can be very site specific.

References

[1] Vodafone, "Open RAN System Integration Whitepaper," 24 May 2022. [Online]. Available: https://www.vodafone.com/news/technology/vodafone-driving-greater-efficiency-open-ran.

[2] Deutsche Telekom, Orange, Telecom Italia (TIM), Telefónica, Vodafone, "BUILDING AN OPEN RAN ECOSYSTEM FOR EUROPE," November 2021. [Online]. Available: https://www.vodafone.com/sites/default/files/2021-11/building-open-ran-ecosystem-europe.pdf.

[3] G. Brown, "Reviewing Vodafones Open RAN System Integration White Paper," 6 July 2022. [Online]. Available: https://www.lightreading.com/open-ran/reviewing-vodafones-open-ran-system-integration-white-paper.

[4] K. Olukotun, "High Performance Domain Specific Languages using Delite," 2012. [Online]. Available: https://ppl.stanford.edu/papers/CGO2012-1.pdf. [Accessed 2022].

[5] M. Lohstroh, "Toward a Lingua Franca for Deterministic Concurrent Systems," 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3448128. [Accessed 2022].

[6] D. Kim, "Table driven proportional access based real-time Ethernet for safety-critical real-time systems," 2001. [Online]. Available: https://asu.pure.elsevier.com/en/publications/table-driven-proportional-access-based-real-time-ethernet-for-saf. [Accessed 2022].

[7] X. Foukas, B. Radunovic, M. Balkwill and Z. Lai, "Taking 5G RAN Analytics and Control to a New Level," [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2022/12/JanusTechnicalReport.pdf.

7. Appendix: Applying RDSL™ and Automation to Intel® FlexRAN™

The automation framework described in this paper has been applied to the Intel® FlexRAN™ solution using Cirrus360's RAN automation platform (Gabriel™) and RDSL™. As shown in Figure 5, the 5G NR protocol real time behavior and flows have been represented to Gabriel™ using RDSL™ as well as an abstract description of the Intel® Xeon Gold along with the Intel® vRAN Accelerator ACC100. Several tests were conducted spanning sub-6 and mMIMO scenarios with a range of target deployment constraints such as latency.

The automation platform analyzed and explored the solution space to find a feasible schedule of the software on the given hardware, such that the target deployment constraints were met, while the solution was optimized for maximum power savings opportunities. Using this methodology, the automation framework was able to significantly increase the power savings opportunity in the optimized FlexRAN™ solution compared to the manually optimized default configuration of FlexRAN™.

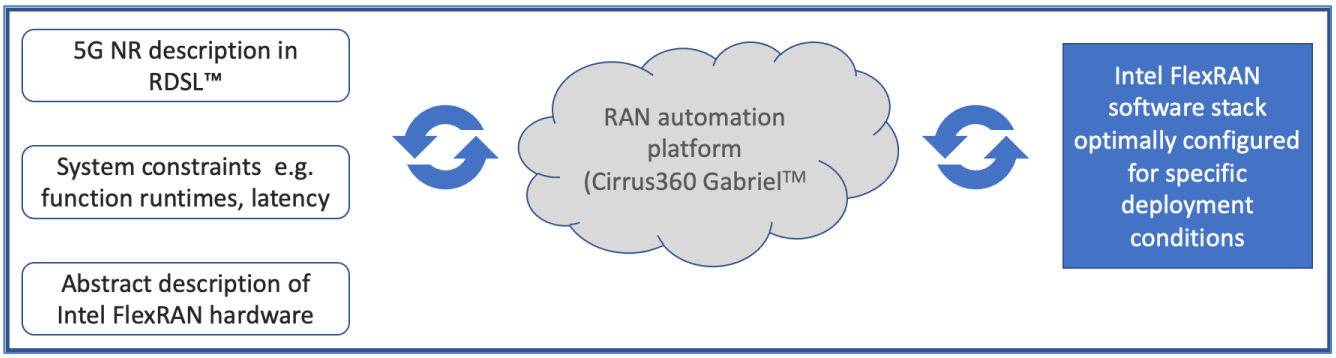

Figure 5: RDSL™ and Intelligent Automation from Cirrus360 applied to Intel® FlexRAN™

As an example of how significant changes can be represented clearly within the declarative language of RDSL™ we show two code fragments in Figure 6. The first fragment implements a flow (a specific type in RDSL™) which processes streams (also a type) of data such that several channel compensation operations are declared independently. In the other fragment, all of the channel compensation occurs within a single logical function. This algorithm difference is often seen when optimizing for high Doppler (the top RDSL™ fragment) versus medium to low doppler. Apart from swapping out these two flow definitions, nothing else was changed in the RDSL™ description of uplink 5G mMIMO processing. Gabriel processed the RDSL™ declaration of the uplink functionality, along with constraints and the abstract description of the hardware, as shown in Figure 5. For a small enough latency constraint on uplink processing, we saw that the first flow could be scheduled but the second could not. When the latency constraint was relaxed so that both had a feasible solution, this was achieved in the second flow by pushing more equalization and subsequent processing towards the end of the latency budget and rearranging the other functions to accommodate this.

```
%*******************************************************************************************
%PUSCH DMRS Channel Estimation for a single User group per beam
%ulsh_DMRS_xSymbol_In is a vector of DMRS symbols for this beam
%perUEGrp_ChEst_info_In is the ChestInfo common to all UE
%perUEGrp_ChEst_Out is the CHEST for the UE group after symbol post processing
%*******************************************************************************************
flow puschDmrs_chest_perSymbol
        ulsh_DMRS_xSymbol_in    :stream[MAX_NUM_DMRS_SYMB]{type = in}
        perUEGrp_ChEst_info_in  :stream{type = in}
        perUEGrp_ChEst_out      :stream[NUM_SYMBOLS_IN_SLOT]{type = out}

        %written specifically for 4 symbol groups and 2 DMRS for now
        chestCompensation(ulsh_DMRS_xSymbol_in[1], perUEGrp_ChEst_info_in, perUEGrp_ChEst_out[1])
        chestCompensation(ulsh_DMRS_xSymbol_in[1], perUEGrp_ChEst_info_in, perUEGrp_ChEst_out[2])
        chestCompensation(ulsh_DMRS_xSymbol_in[2], perUEGrp_ChEst_info_in, perUEGrp_ChEst_out[3])
        chestCompensation(ulsh_DMRS_xSymbol_in[2], perUEGrp_ChEst_info_in, perUEGrp_ChEst_out[4])
```

In this RDSL™ example four channel compensation operations are called out explicitly, each can be scheduled by automation constrained by the availability of their inputs. Each input to be compensated is taken from a tensor stream of inputs to the flow.

In this RDSL™ example a single channel compensation operation operates on the complete set of input symbol streams in the tensor stream. The schedule of the compensation is therefore constrained by the availability of all streams and must run on a single computational unit within the hardware.

```
%*******************************************************************************************
%PUSCH DMRS Channel Estimation for a single User group per beam
%ulsh_DMRS_xSymbol_In is a vector of DMRS symbols for this beam
%perUEGrp_ChEst_info_In is the ChestInfo common to all UE
%perUEGrp_ChEst_Out is the CHEST for the UE group after symbol post processing
%*******************************************************************************************
flow puschDmrs_chest_perSymbol(i,j)
        ulsh_DMRS_xSymbol_in    :stream[MAX_NUM_DMRS_SYMB]{type = in}
        perUEGrp_ChEst_info_in  :stream{type = in}
        perUEGrp_ChEst_out      :stream{type = out}


        chestCompensation(i, ulsh_DMRS_xSymbol_in, perUEGrp_ChEst_info_in, perUEGrp_ChEst_out)
```

Figure 6: Example RDSL™ fragments

The conclusion, as we have seen in many examples implemented using FlexRAN™ reference software and RDSL™, is that a change in algorithm can be easily accommodated using the methodology described in this paper, even when a significant amount of rescheduling of functions is required in the FlexRAN™ reference software to meet latency or other constraints. This allows the FlexRAN™ reference software to be optimized for very specific deployment requirements, potentially saving power and cost in the network. For the example in Figure 6, the two different flows can both be part of the RDSL™ declaration and the relative amount of each type can be added as a constraint, so balancing high and low doppler support for a specific deployment. Such fine granularity deployment tradeoffs are only possible with the automation approach described in this paper.

8. Appendix: Onboarding new Software and Hardware Vendors

The onboarding process consists of three steps:

1. Identify or develop a middleware capable of supporting the scheduling of SoC (System on Chip) resources
2. Provide envelope data on functions in RAN Software Development Kit (SDK)
3. Provide an abstract .xml description of the hardware at the middleware level

Middleware

Gabriel provides a Configuration and Control Layer (CCL) file to the middleware that describes a collection of operations on the resources described in the hardware abstraction file. So, middleware needs to be able to control the operation of unit functions on the hardware to perform specific operations at specific times. How this is achieved by the middleware is up to the middleware developer (for instance the SoC provider). *For standardization purposes only the format of the description file needs to be defined. We have successfully used an extensible xml format and have written generators for this format allowing a simple creation and modification process for hardware abstraction.* This approach to hardware abstraction is already in use for AAL in ORAN standardization.

SDK

Envelope data for SDK functions defines basic runtime and memory use bounds for the function running on a defined resource in the hardware abstraction file. As the resource is abstracted, no detailed information on the type of resource is required. For instance, it might be an FPGA block or a CPU but the optimization process does not care except that it needs the runtime envelop of the function for that resource. A very basic and extensible set of rules needs to be defined to allow any optimizer to read envelope data.

Hardware Abstraction

Starting with the current AAL abstraction we can extend the AAL to encompass the rest of the SoC. Hardware abstraction needs to be extended to include performance information, such as latency models for interfaces. Some of this work is already ongoing in ORAN today.

9. Appendix: Systems Optimization versus Performance Optimization for RAN

System integration is the process of combining multiple IP blocks to form a larger system that meets specific system level goals. These goals can be divided into radio performance goals and system implementation goals. Examples of radio performance goals are:

- Cell throughput at a given SNR.
- Support of 3GPP specific features such as MIMO, beamforming, LDPC modes and their impact on system performance for different traffic types and loads.
- End to End latency requirements.
- Radio parameter tuning to optimize radio performance, which may in turn be performed by Machine Learning algorithms in the RIC.
- Rate of SRS, RACH and other network maintenance signaling.

All of these goals are focused on maximizing user satisfaction, user density and meeting certain critical algorithmic performance criteria for correct operation. They must be achieved regardless of the implementational details of the RAN. For this reason, we will informally call them “3GPP requirements” Examples of system implementation goals are:

- Power Constraints within the chassis and power minimization to meet OPEX goals.

- Implementation of upgrades on a variety of existing deployed hardware with minimum onboarding time and cost.
- Simplicity of, and bug traceability of, software releases.
- Integration of multivendor software IP on a single existing hardware platform.
- Integration of a single software upgrade across multivendor and multi-SKU hardware platforms.
- Maximizing cell capacity in terms of number of cells that can be run on the hardware.
- Integration of Functionality of one vendor with another vendor's functionality. For instance:
  - The data access timing requirements of Vendor A RU being incompatible with vendor B DU.
  - Latency budgeting across multivendor IP that was developed before latency requirements were set, to achieve latency but also optimize for hardware and power requirements.
  - Integration of CU and DU on the same hardware platform where the two IP share resources, such as the DDR interface.

All of these goals focus on the development of a flexible, maintainable and low-cost network that can react quickly to new feature requirements while minimizing heisenbugs and other unplanned faults. They allow the system integrator to constantly optimize their network in a very fine-grained manner to minimize OPEX and CAPEX. For this reason, we will informally call them "ORAN requirements". Achieving them is critical for the success of the Open RAN concept.

In this paper we focus on the ORAN requirements. Without further discussion we posit that 3GPP requirements can be achieved in a multivendor environment through the development of better algorithms *provided* ORAN requirements are already achieved.