INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Linguagem Específica de Domínio para a Integração de Sistemas Ciber Físicos

PAULO RENATO MENDES OLIVEIRA
Outubro de 2021

POLITÉCNICO
DO PORTO

[Domain Specific Languages for Cyber-Physical Systems Component Integration]

Renato Oliveira

A dissertation submitted in fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering

Supervisor: David Pereira, Auxiliary Researcher, CISTER, ISEP, P.Porto

Porto, October 13, 2021

Dedicatory

Dedicated to myself, my colleagues at Bosch, who I thank for lifting the heavy load during working hours so I could complete this work, to my supervisor, David Pereira for all the support and teachings, and to my girlfriend, Soraia Mesquita for being always there for me whenever I'm stressed out about having too much work to do.

Abstract

The present work is inserted in the area of Cyber-Physical Systems (CPS), with particular emphasis on the automotive domain. With the ever increasing globalization of this industry, and the increasing levels of production required to satisfy customer needs, a necessity arises to automatize many tasks which allow for the development of systems that can support the growth of the industry.

Developing complex CPS applications for the automotive domain is a complex task, due to the criticality inherent to this domain and the need for increased functionality, safety, and security. In order to properly address such complexity of the design and development process, the development of a new Domain Specific Language (DSL) is proposed. This DSL aims to provide a mostly automatized integration process between the different entities or components of a CPS application, abstracting away technical details of these components and enabling rigorous methods of verification, customization, and deployment.

To this end, the DSL provides different views of the system, where properties or constraints imposed in a view propagate to other views, increasing the set of systemwide properties that need to be ensured correct in order for the system to be considered safe.

Each of these views has language concepts associated, which get translated into target language, as the objective is to build a system or parts of a system that respect the restrictions imposed by those views and concepts.

The DSL was implemented using Meta Programming System (MPS) and artifacts were generated from a given specification of DSL concepts and utilized in a deployment environment utilizing a BananaPi-M1 board with a custom configured Linux distribution and kernel. Custom scripts were also deployed that help in the installation of the operating system as well as relevant packages to install on the first boot. Additionally, application layer artifacts were also deployed, using a message oriented architecture (publish/subscribe), executing on top of a Robot Operating System (ROS) instance.

Keywords: DSL, View, Integration

Resumo

Este trabalho está inserido no domínio dos Sistemas Ciber-Físicos, com um ênfase particular no domínio automóvel. Com o crescimento desta indústria e clientes com necessidades que requerem níveis de produção mais elevados, existe uma necessidade de simplificar e automatizar tarefas que fazem parte de processos de desenvolvimento de sistemas que consigam suportar o crescimento da indústria.

Tipicamente, o desenvolvimento deste tipo de sistemas é uma tarefa complexa, devido à criticalidade inerente a este domínio. Para conseguir mitigar este problema, propõe-se o desenvolvimento de uma DSL, que irá providenciar uma integração semi-automática entre as diferentes tecnologias que compõe este domínio, abstraindo os detalhes mais técnicos da integração entre estas tecnologias para providênciar uma experiência mais simplificada aos utilizadores.

Para atingir este objetivo, a DSL providencia diferentes vistas do sistema, sendo que cada uma dessas vistas captura um conjunto de propriedades diferente, que poderão ser partilhadas entre elas, com o objetivo de produzir um sistema que respeite essas mesmas propriedades. Cada uma destas vistas tem conceitos da linguagem associados, sendo que estes conceitos são traduzidos para a(s) linguagem(s) alvo.

A implementação desta DSL foi feita com recurso ao MPS que é uma sistema que permite desenvolver DSL's de maneira intuitiva. Este desenvolvimento, tipicamente, requer a adoção dos padrões de desenvolvimento enforçados pela ferramenta, nomeadamente, a criação de conceitos da linguagem que são na sua essência uma unidade que representa algo no domínio.

Para cada um destes conceitos é necessário definir a sua estrutura (propriedades, filhos, referências), o seu editor, que é a forma que o conceito irá ser apresentado ao utilizador da DSL e o seu componente de geração de texto, que dita como o conceito é traduzido para linguagem alvo. Adicionalmente, para cada conceito são definidas também quaisquer restrições de domínio que tenha, que na verdade poderão ser transformadas em restrições de sintaxe para o utilizador final, bem como quaisquer comportamentos relevantes para o domínio daquele conceito, comportamentos estes que muitas vezes ajudam na tradução para a linguagem alvo.

O objetivo é produzir um sistema com recurso a estas vistas, que seja extensível, configurável e manipulável, mas ao mesmo tempo, oferecer algum isolamento a partes criticas que poderão requerer acesso ininterrupto a recursos de hardware. Posto isto, propõe-se um caso de uso em que o objetivo será gerar configurações do Jailhouse, que é um hypervisor que é um hypervisor particional baseado em Linux. Nesta configuração será possível incluir também partições, que são blocos isolados no sistema que providênciam o acesso controlado a recursos de hardware. Este isolamento deverá ser configurável pelo que o caso de uso contempla a configuração de canais de comunicação entre partições. Adicionalmente, propõe-se a configuração de um sistema operativo, sendo que esta configuração envolve todo o processo de obter uma base, configurar o sistema de ficheiros, injetar scripts de

configuração de pacotes nesse sistema de ficheiros e também configurar o kernel da distribuição escolhida. Para terminar, o caso de uso contempla também a produção de uma camada aplicacional, executada em cima de ROS (Robot Operating System), que consiste num conjunto de aplicações com arquitetura orientada a mensagens Pub/Sub.

Dado que uma vista de sistema envolve um conjunto de conceitos que podem ou não ser partilhados com outras vistas, a partir de uma configuração especifica de conceitos com as suas propriedades definidas, é possível correr o gerador de linguagem e gerar então artefactos.

A partir de uma destas configurações, foram então gerados um conjunto de artefactos que foram utilizados para produzir uma prova de conceito, utilizando uma board BananaPi-M1, implantada com um conjunto de sistema operativo Linux e kernel, configurados pelo utilizador utilizando a DSL. Adicionalmente, foram gerados scripts que ajudam a esta configuração, instalação do sistema operativo e pacotes de software relevantes para ajudar na execução do caso de uso.

Contents

List of Figures

List of Tables

List of Source Code

List of Acronyms

ADAS	Advanced Driver Assistance Systems.
AST	Abstract Syntax Tree.
CPS	Cyber-Physical Systems.
DSL	Domain Specific Language.
FFE	Fuzzy Front-End.
IRQ	Interrupt Request.
IVSHMEM	Inter-Virtual Machine Shared Memory.
MPS	Meta Programming System.
NCD	New Concept Development.
NPD	New Product Development.
OS	Operating System.
ROS	Robot Operating System.
SCES	Safety Critical Embedded System.
UI	User Interface.
VA	Value Analysis.

Chapter 1

Introduction

1.1 Context and Problem

The present work is inserted in the area of Cyber-Physical Systems (CPS), with particular emphasis on the automotive domain. With the ever increasing globalization of this industry, and the increasing levels of production required to satisfy customer needs, a necessity arises to automatize many tasks which allow for the development of systems that can support the growth of the industry.

Developing complex CPS applications for the automotive domain is a complex task, due to the criticality inherent to this domain and the need for increased functionality, safety, and security. A considerable part of the development effort relies on the integration of third-party and vendor specific software components, thus involving the usage of plethora of different technologies, typically developed in isolation (and by different entities) and, later in the process, integration and verification efforts that delay the time to market of new solutions, and higher production costs. The different technologies typically involved in the development of automotive applications range from distinct modeling frameworks, code editors and programming languages, development frameworks, operating systems, COTS platforms, support for virtualization via hypervisors, among others. Such diversity of technologies leads to an integration process that can be very complex, tedious, error prone, and time-, money- and effort-consuming, according to Land and Crnkovic 2003.

As the complexity of these CPSs increases, our inability to rigorously capture the interactions between the physical and the computation components paves the way to software errors that can lead to serious vulnerabilities during operations. Ultimately, systems become unsafe, with disastrous failures that could not have been properly identified during the several stages of the application's development cycle, according to Sangiovanni-Vincentelli, Damm, and Passerone 2012.

1.2 Objective

In order to properly address the complexity of the design and development process, the development of a Domain Specific Language (DSL) is proposed. This DSL aims to provide a quasi-automatic [1] integration process between the different entities or components of a

[1] By quasi-automatic a scenario is envisioned where consistency checking is mostly made by the solution itself, automatizing the integration process

CPS application, abstracting away technical details of these components and enable rigorous methods of verification, customization, and deployment.

To this end, the DSL provides different views of the system, where properties or constraints imposed in a view propagate to other views, increasing the set of systemwide properties that need to be ensured correct in order for the system to be considered safe. By providing distinct views, our envisioned solution allows different system intervenients to seamlessly interact during the development process, simultaneously ensuring that the underlying infrastructure used for this interaction is verified, and that CPS applications and systems can be easily built, customized and deployed.

Each of these views can be deployed in a standalone state, meaning the output of the DSL can also be utilized to fill gaps within closed loop systems. In this context, deployment is related to the generation of code and/or artifacts that allow for the setup and execution of a specific target, which is related to a specific view of the full system.

The syntax of the generated code, of course, varies according to the target technology for which it aims. Additionally, in what concerns the DSL syntax, design-time validation shall be delivered, with rules defined for each pertinent system view, which shall feed visual cues to the user. This syntax validation includes error detection (context based, typos, inter-view connections, etc...) as well as correctness highlighting for keywords.

The DSL shall have auto-complete features for known keywords and related to the context, as well as pre-filled parts of the view definition, which shall also be based on context.

1.3 Value Analysis

According to Miles 2015, with the ever increasing competition between businesses, success depends largely on offering the customer the best value for the price asked. Value depends both on user and the context, with it becoming an imprecise word. It can be further defined as performance of its function, divided by its cost. In order to obtain the best outcome, a Value Analysis (VA) can be performed. Value analysis, starts with the Innovation Process, where the opportunity and approaches are identified and analyzed. Next, the Solution Value expands and clarifies the value proposition, followed by a Functional Analysis, in which solution functions are established with the purpose of achieving the project goals.

The Innovation process includes the innovation phase of the product life cycle, which consists the applying the Fuzzy Front-End (FFE) methodology, then, the New Product Development (NPD), and, lastly, the Commercialization stage, which is where the promotion and distribution of the product occurs, according to Koen et al. 2002.

As the name indicates, the initial stages of FFE are fuzzy, due to lack of knowledge about the product-to-be, which means that the conclusions that stem from that phase may not be predictable, which impacts the other stages of the innovation process. This lapse in the methodology is mitigated the a new model, named New Concept Development (NCD), since it is a more structured model, creating more predictability within the process itself.

With that being said, such analysis will be performed within the scope of the present work, following known and relevant methodologies to perform this analysis, namely the NCD (that closes some of the gaps that FFE has. The more project specific analysis will be performed within subsection 2.4.

1.4 Work Methodology

The work methodology followed a waterfall pattern, with constant artifact validation being required by the supervisor. Required changes and additions were performed, requiring more validation until a proof of concept was achieved. Additionally, as this work was part of a collaborative work between CISTER Research Centre and Vortex [2], various meetings discussing requirements and checking the progress of the development process took place. In these meetings, the members of the collaborative laboratory were present and had impact on the development process in what concerns DSL requirements and implementation validation.

1.5 Structure

In the present chapter, the context, objective of the present work and an introduction about value analysis was provided.

In Chapter 2 the context in which the present work is inserted is expanded and a study about the state of the art is presented.

In Chapter 3, the design challenges of the DSL are provided, explaining the chosen tool and its relevant intricacies.

In Chapter 4, the implementation of the DSL is documented.

In Chapter 5, experiments and results obtained while testing the DSL, are provided.

In Chapter 6, conclusions about the present work and future improvements on the DSL are documented.

Lastly, all the relevant appendixes that are relevant but are not located within the main document, are provided.

[2] https://www.vortex-colab.com/

Chapter 2

Context and State of the art

Within this chapter, a focus will be given to tools that allow for the development of Domain Specific Languages (DSLs), with respect to some assumptions, that will output artifacts related to the description in subsection 1.2.

2.1 Assumptions

Given the broad scope of the present work, naturally, some constraints regarding the candidate tools to develop the DSL have been set. These are listed as follows:

1. It is assumed a specific use-case;
2. Specific technologies and target systems;
3. Tool that allows for text-based code generation, either templated or pure text;
4. Tool that allows for some degree of control regarding usability [1];
5. Tool that allows for validation, extension, configuration;
6. For the integration aspect, it is assumed that different technologies are compatible with one another;

2.2 Context

Within the scope of the present work, there are two distinct view angles that can be analyzed. First, the context of DSLs, more specifically the necessary processes that make up the definition and purpose of such structures. On the other hand, there is the purpose for which a DSL is designed, which is separate from the process that leads to the creation of a DSL. This purpose has its own relevant parts, processes and interested parties. This document will reflect both, to a degree, with the main focus on the purpose for which the DSL is designed, since the process of creating and designing a DSL is shaped by its purpose and/or the tools that are used in its construction, and, since this is the case, this process will be illustrated on Chapter 3. The process described in this section is related to the output/target system.

[1] This may include an IDE output

2.2.1 Business Concept and Process

This subsection describes what the key business concepts of the present work are, as well as the basic process utilized to achieve this. The first business concept is the ability to quickly prototype parts of, or a full system in a seamless way, which allows for quick feasibility tests with quick results. The second key business concept is system integration, which contemplates that different parts of the system that may have dependencies with one another have the ability to reflect this within the DSL, providing this integration aspect and guaranteeing that configuration is consistent between different technologies. The third key business concept is validation, where, based on the rules of target technology as well as integration between different technologies, visual feedback is given to the user before code generation

In what concerns the process, a generic approach to each system view is desired. This means one process is followed, which shall be technology agnostic. This process is comprised by 4 phases:

1. Concept definition
2. Editor and Syntax definition
3. Rule validation
4. Text Generation

The Concept definition phase is where relevant business concepts are reflected and wrapped into a single node. The Editor and Syntax definition phases consists in shaping how a single concept is presented to the user. Rule validation allows for constraints can be applied to a subset of properties of the defined concept. Lastly, the Text Generation phase consists in defining a translation for the concept and its properties. This translation effectively is utilized by the code generator on a 1:1 basis.

A single system view is characterized by multiple concepts, and these vary according to the system view itself. Integration between system views respects the process illustrated above, however, it does require consistency validation based on the current context, with constraints applied to one system view possibly depending on a constraint from another system view, henceforth known as constraint dependency.

2.3 State of the art

2.3.1 DSL Requirements

In this subsection, some criteria which will be used to assess if a DSL is adequate to the project's needs will be defined.

The defined criteria in this section should be respected by the solution as a whole, i.e., not only the DSL but also the high level models, the code generation tools and finally, any existing middleware layers. With that being said, we define the criteria as such: (1) the solution must employ a generic, easy-to-use DSL; (2) it must support the definition of high level models by using the DSL; (3) it must allow for the definition of requirements/restrictions; (4) it must allow for the definition of annexes which further specify the model; (5) it must generate code which is a translation of high level entities into low level processes; As such,

we will go over each of these high level requirements and describe their specific objectives, when applicable.

Table 2.1: High Level Model Definition specification

Requirement	**Description**
High level model definition	- It should be possible to create/update/delete entities - It should be possible to connect entities - It should be possible to define properties for entities - It should be possible for models to inherit other models - It should be possible to re-use previously defined models

The requirement above stems from the need of a logic modeling process. It is more intuitive to define a generic model and then specify it, also known as 'top down approach'. This requirement is generally dependent on a User Interface (UI) in order to perform the described functions more intuitively.

Table 2.2: Restriction Definition specification

Requirement	**Description**
Restriction definition	- It should be possible to define restrictions for single entities - It should be possible to define group restrictions

The existance of restrictions allow the definition of a more specific, complete system. They are crucial when designing a Safety Critical Embedded System (SCES) as they are required to ensure the safety attribute the system should guarantee, but can be useful for other types of systems. Also, these restrictions can apply not only to the business logic, but also for the DSL editor itself.

Table 2.3: Annex Definition specification

Requirement	**Description**
Annex definition	- It should be possible to define new annexes for an existing DSL

This requirement exists because the target DSL business logic might not be aligned with the project's needs. If that is the case, it should be possible to adapt the DSL in order to consider it a candidate. We brand this adaptation as annex definition.

Depending on the degree of the misalignment between the DSL's business logic and the project's needs, a study has to be performed in order to assess if it is profitable to develop a new DSL which is better aligned with the target business logic instead of writing an annex for the existing DSL.

Table 2.4: Entity Translation specification

Requirement	Description
Translate High Level Entities to Low Level processes	- It should perform the transformation of high level concepts into real, palpable processes

The requirement above can be logically aggregated with the code generation aspects of the tool. It does not require code generation itself, however, its purpose is to, upon its use, output a list of guidelines for the code generation tool to follow.

With the requirements specified, the next subsection dives in some candidate tools that will be evaluated according to these requirements.

2.3.2 Existing solutions for CPS integration

There are a number of languages that help the development of complex CPS applications by enabling the integration between different components and systems. In this subsection we briefly overview some of such languages.

SysML

One of such languages is the *System Modelling Language* (SysML), which aims to provide cross-domain technology and system integration. SysML addresses issues associated with *multi-view modeling* (Shah, Schaefer, and Paredis 2009) and offers a unifying language between the various views of the system according to Shah, Schaefer, and Paredis 2009. SysML is based in the *Unified Modelling Language* (UML) and allows for the creation of a plethora of diagrams that output models, which can then be used to generate code.

SysML allows basic operation signatures to be defined at interfaces, and pre- and post-conditions to be specified textually (Bryans et al. 2014). This feature is tightly connected with the concept of *Design by Contract* (DbC) according toBryans et al. 2014. Contract verification, in the case of SysML, requires the translation of these contracts into the formal notation of the *COMPASS Modelling Language* (CML) (Bryans et al. 2014). As this is a somewhat manual process, and since it relies on non-formal, natural language for a textual description of the contracts, it does not enable the rigorous specification of system properties or/and constraints that need to be verified in order to ensure system correctness.

SysML cannot transform models into target code by itself, which introduces other tool dependencies, like *Enterprise Architect* (Bryans et al. 2014). Additionally, SysML is by design a diagram oriented language, which is not always the more convenient way of satisfying the ever increasing necessity of incorporating different applications in a single solution that typically relies on tweaking code to achieve such integration. With that being said, this candidate tool is not fit for use to achieve the envisioned objectives.

AADL

Another language that provides integration support among different components of a system is the *Architecture Analysis and Design Language* (AADL). AADL was designed for the

specification, analysis, integration and code generation, based on requirements that refer to performance-critical systems.

AADL allows the analysis of CPS systems prior to their development, supporting model driven development approaches throughout the system life cycle. Due to its focus on the embedded systems domain, it can be used to produce both software and hardware, making it a very versatile tool for designing systems.

AADL can be used to manage both system and software aspects within the same model, offering a single notation mechanism for both. The existence of a single model eases the analysis because there is only one representation of the system, eliminating possible ambiguous notions that may emerge. Therefore, in order to properly produce a model, any AADL user can extend the language by defining properties, which in turn are system specific characteristics.

It is possible to extend the AADL language for a specific domain, via annexes. Language annexes enhance the core AADL language in order to provide an enrichment to the architecture description and definition. At the present time, there are four defined annexes: the (1) behavior annex, which adds behavior to components with state machines; the (2) errormodel annex which specifies fault and propagation concerns; the (3) ARINC653 annex which defines modeling patterns for avionics systems; and the (4) data-model annex which describes the modeling of specific data constraints within AADL. With that being said, it would be possible to design a DSL using the annex definition capabilities of AADL. We chose not to pursue this path due to its inherent focus on real-time performance-critical (timing, safety, schedulability, fault tolerant, security, etc.) systems. These properties, while useful in some domains, might not fit the scope of the solution we are trying to envision. Additionally, the reason for not using AADL stems from the difficulty of representing different views using this language.

EAST-ADL

EAST-ADL which exists as a framework to provide a support for the non-ambiguous description of in-car embedded electronic systems at each level of their development Debruyne, Simonot-Lion, and Trinquet 2004. It is designed to complement the *AUTomotive Open System ARchitecture* (AUTOSAR)[2] standard with descriptions at a higher level of abstraction than the ones supported by the default implementation. With the focus completely on the automotive domain, it is not sufficiently generic to support other types of CPS applications, which is one of the main objectives of the DSL proposed in the present work.

With all these issues accounted for, it brought the necessity to design a tailor-made system integration solution, by developing a dedicated DSL.

2.3.3 DSL Creation tools

In this section we will describe two candidate DSL development tools, namely the Xtext tool and the Jetbrains Meta Programming System (MPS).

[2]AUTOSAR is a standard for the automotive software architecture between suppliers and manufacturers

Xtext

Xtext is a framework that allows one to quickly develop tools for a textual language Eysholdt and Behrens 2010. It can be used to develop DSLs or General Purpose Languages (GPL). Additionally, it has support for configuration files and requirement documents.

Xtext takes advantage of the *Eclipse Modeling Framework (EMF)* and allows easy integration with tools from the Eclipse Modeling ecosystem, such as Model-to-Model or Model-to-Text transformation languages Eysholdt and Behrens 2010. Starting with a grammar definition, Xtext generates a parser, serializer and a smart editor for the language. All these generated artifacts can be configured or customized via dependency injection Eysholdt and Behrens 2010.

Unfortunately, considering the domain approached in the context of the present work, a need for the validation of various concerns emerges, those concerns being timing, safety, schedulability, fault tolerance, security, etc. As such, the default behaviour that Xtext provides does not meet the needs of our envisioned solution, as we would need to customize this behaviour to close this gap. In order to perform this customization, some proficiency with Xtend is required [3]. Xtend is used to aid Xtext on processing models and expressions built under the latter, and they are normally used together. With that being said, although using a combination of these tools is a possible strategy, it requires proficiency with both of them, which possibly implies increased staff costs and development time, therefore, this tool is discarded as a possible solution.

MPS and the MBEDDR extension

MPS is a metaprogramming system and also known a language workbench that allows the development of DSL, simultaneously creating models for such DSLs, generating code, perform model verification, define annotations and create documentation. MPS relies on a projectional editor, meaning that the Abstract Syntax Tree (AST) is changed after every input instead of having a parser that transforms a character sequence in that same tree. This brings some advantages, namely, language modularity and flexible notation capabilities.

This tool also allows for extensions to be created for a given language in a smooth way, allowing for a more modular language, by eliminating traditional parsing methodologies. Instead, MPS experiments with a non-textual representation of the program code, treating the AST as a set of nodes, with a set of attributes (properties, children, references), with these fully describing the program code.

The projectional editor allows for quick visualization of this tree of nodes, providing a more user-friendly experience, enforcing all the rules defined by the user, such as types, concept constraints, etc, which MPS checks as the code is written, allowing for more control over user input.[4]

Although MPS is adept at generating Java code, it can generate code for other languages by using the embedded capabilities of the tool, however, in this project, many of the technologies and the integration needed, requires generating code free of language constraints, which

[3] Xtend is a flexible and expressive dialect of Java

[4] Full description of the tool can be found in *MPS Description* n.d.

MPS allows too. Even though this is the case, MPS lacks concerns related to Cyber-Physical Systems (CPS) development, therefore, a relevant plugin to help mitigate those faults was also identified.

MBEDDR is an extensible C-based language and IDE for embedded software development Voelter et al. 2012 and the features provided by this plugin are candidates for use within this project due to their focus in embedded system development, even though the usage of C as the only translated code is not the objective. Although this is the case, MBEDDR can be used as a set of plugins of MPS, which introduces features that bring value to our project, e.g., state machine definition, formal analysis capabilities, among others.

The plugins that compose MBEDDR contain language definitions, namely MBEDDR C and its extensions, which are languages in terms of MPS, in addition to the many libraries, utilities, views, editors, etc.

With that being said, it is pertinent to briefly describe MBEDDR architecture, which can be found on the figure below.

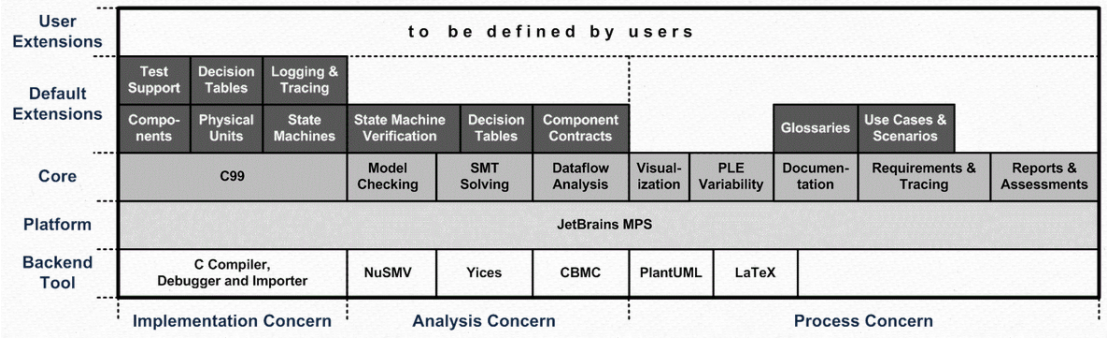

Figure 2.1: MBEDDR Architecture

,

MBEDDR's structure is comprised of 5 layers: (1) user extensions, which is where users can define languages and other domain specific features; (2) default extensions, which are the extensions already supported by MBEDDR; (3) core, which contains the bulk of MBEDDR's features like the C99 language standard, model checking features, requirement specification, documentation, etc.; (4) platform, which is the JetBrains MPS framework; (5) backend tool, which contains the basis (independent tools) for the entire tool like the C compiler, new symbolic model checker, etc.

Additionally, MBEDDR's architecture focuses on three distinct concerns, namely the implementation concern, the analysis concern and the process concern. The implementation concern addresses the development of applications based on C with the advantage that the user can extend the language and any of the existing extensions.

The analysis concern contemplates static analysis (formal verification) for some of the default extensions provided by the implementation concern. These analysis (based on symbolic model checking, SMT solving and C-level model checking) are performed by external tools, integrated in MBEDDR.

The process concern focuses on the facilities of integrating MBEDDR into development processes. These features apply to default and user-defined C extensions. It contains features such as requirements support which provides a language to describe requirements;

traces that can be attached to any program element, additional arbitrary data that can be added to a requirement, among other features.

With that being said, Meta Programming System will be the tool utilized to develop the DSL, as it fits all the requirements identified within subsection 2.3.1. Next, a more project specific VA will be partially employed, as a follow up of the introduction provided in Section 1.3.

2.4 Value Analysis

On this section a VA more related with the topic of the present work, will be performed.

The first step of the NCD model is named Opportunity Identification, where opportunities to solve a given problem are identified. As stated in Chapter 1, the problem to solve is connected to the complexity that revolves around integrating different technologies related to the Automotive Systems domain. As such, an opportunity to simplify these costly processes arises. The next step consists in analyzing the opportunity that was identified, and could be achieved using formal or informal methods. In this case, a SWOT analysis will be applied, in order to identify the strengths, weaknesses, opportunities and threats related to the opportunity. The SWOT analysis figure can be found below.

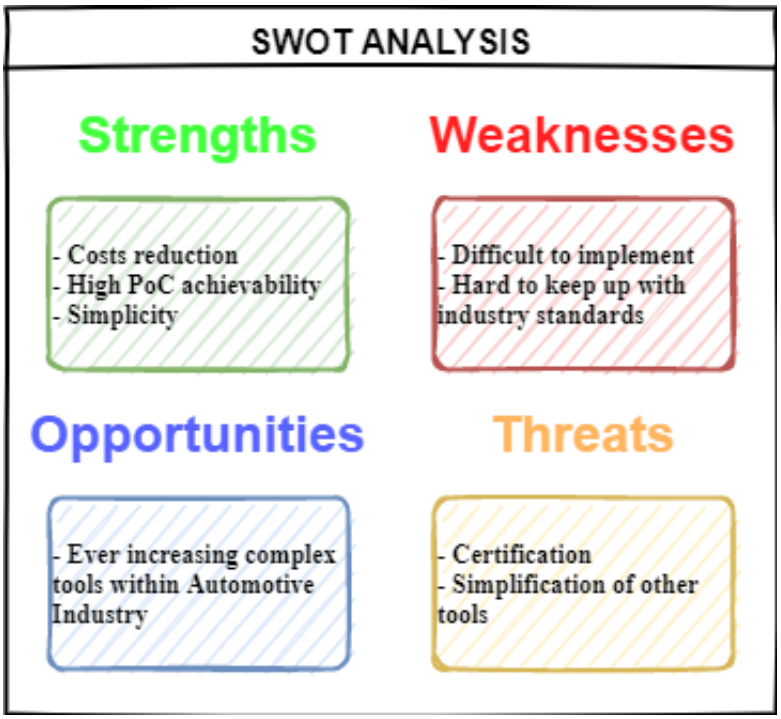

Figure 2.2: SWOT Analysis

Starting with the strengths, cost reduction is a very effective catalyst for technology adoptions various industries. In this case, costs would be reduced in software licenses for companies that do not use the tools to their full extent and could use a simpler alternative. Similarly, costs would be reduced in employee training, since the employee would have to

feel comfortable with various technologies to produce the same output that the output of this work would provide.

The next strength lies in the quickness of achieving a proof of concept quickly. Even if the objective is to use more complex tools to achieve the end result, a tool that would provide a glimpse into the future and viability of a project, may be desirable. This ties in with the next strength, which is simplicity.

For weaknesses, the complex implementation is hard to ignore; for each technology that the tool simplifies, knowledge must be obtained about that tool, and, accounting for all the existing tools within the same system when it concerns integrating them. The industry standards for the Automotive Industry are quickly evolving in what concerns which features a new vehicle is required to have. With that being said, the tool would have to keep up with these standards in order to be considered by companies.

Within opportunities, and as stated, the increasing complexity in requirements gives origin to even more complex tools, paving way to a simpler solution being taken into account for less complex projects.

Lastly, in what concerns threats, certification is a big issue since, depending on the kinds of technologies the tool incorporates, some kind of certification of the outputs provided would need certification due to industry standards. Additionally, there is always the possibility of simplification of existing tools that are now complex.

The next steps of NCD would be the Idea Genesis and Idea selection where alternative approaches would be envisioned and then the final idea selected. This does not apply in this case, as the approach of creating a DSL using the technology described Chapter 2.3 is the only viable one.

Chapter 3

DSL Design

In this chapter, the design of the intended DSL will be presented. This design includes concept definition, [1] editor specification (how a given concept is presented in the user-interface, i.e., syntax), text generation (how the translation is made from a given concept to target code) and constraint definition.

3.1 Tool

Before proceeding to a more in-depth design phase, it is important to understand existing limitations that are part of MPS, beyond what was identified in Chapter 2 as these can impact the design process.

3.1.1 The language concept component

This component is part of the initial process of language definition. Here, concept properties are defined. Properties can range from primitive variables to other child concepts. The design limitation here involves knowing what to promote from primitive variable to its own concept, i.e., this concept needs an abstract integer variable but then needs to translate it into something else at run-time. Another challenge is hierarchy definition, which involves defining which nodes are parents and children, for different concepts. This is directly connected to what will be referenced at subsection 3.1.3.

3.1.2 The editor component

The editor component is in charge of defining the syntax that the DSL user will be presented with. This syntax, can be word based, diagram based (using plugins) and there may be other presentation methods. In this case, it will be purely text based due to existing requirements.

3.1.3 The text generation component

The text generation component, also referred to as text-gen, which is a part of the process of concept-code translation, can be skipped with this part of the process being undertaken

[1] Concept in this context is related to the Meta Programming System tool itself, and its related to a language specific concept

by templates. These templates natively exist for Java target code, and, with the MBEDDR plugin, C target code. These templates perform a lot of the tasks that are manual when using the text-gen component, automatically. In the context of the present work, it is pertinent to mention that these templates are not usable due to the different technologies which may be comprised of, but not exclusively, the C language.

Referring again to the text-gen component, there are two ways to go about generating code. The first one involves preparing each language concept to translate its own properties and specification. The second method involves root language concepts collecting information about their child concepts and then translating the whole unit into usable code.

The first method provides modularity, meaning each concept is re-usable in different contexts. The second method provides more data localization, which means more control about verification/validation processes. Within the scope of the present work, the second method will be followed due to the different target technologies, and the specific use-case as each concept will have but one purpose and will not be reused in a different context.

3.1.4 The validation component

This component is in charge of validating. This validation occurs in design-time and run-time. In this case, design-time is when the user is writing code using the DSL. Run-time is when the user is generating code from the DSL into target code.

Within design-time validation there are syntax errors and context errors. Rules for relevant concepts, for the design time validation variant need to be prepared manually.

The run-time validation involves checking each concept node [2], consulting it to obtain data from its properties and then arrange that data into relevant data to be translated. This means an intermediate language that would sit between DSL code and target code is needed. From this intermediate language, target code is then fully written. In this case, the validation may occur either within the intermediate language layer or the target code layer, with the latter occurring due to target code language consistency check with the values provided by the intermediate language.

3.2 Specific use-case

In this section, we introduce the ideas supporting the novel DSL that is to be developed, and illustrate how it can be used to support the development of a simplified CPS application that contains the main characteristics one can expect in the modern and future generation of automotive CPS applications.

With the ever growing demand for cars to be equipped with more advanced, safe, and secure Advanced Driver Assistance Systems (ADAS), whose development costs and time-to-market have reduced, we foresee that many CPS applications will make use of high-performance computing hardware platforms equipped with several, potentially heterogeneous computing cores, as well as with GPUs specially tailored to improve the performance and

[2] One can think about the sum of written statements within the DSL as a tree with various nodes, which are the language concepts

energy efficiency of advanced computation features, like neural networks or image processing algorithms.

Also importantly, single OS solutions are also becoming quite limited, as complex CPS applications for the automotive sector clearly require higher degrees of isolation between components of the system since these components have different levels of criticality and security. Therefore, hypervisors that are able to manage several OSs within the same platform are gaining relevance, as they naturally enforce temporal and spacial isolation among components, and thus allow interference mitigation and provide a natural way to have components executing in an OS environment that provides support for predictable computation and operation over the physical environment (e.g., real-time operating systems satisfying AUTOSAR requirements), and for these to send data for being processed in standard Linux OSs where frameworks like the Robot Operating System (ROS) are used for coordination of components that are responsible for providing the "intelligence" required for vehicles with advanced ADAS features, for example.

With that being said, the specific use-case is comprised of an abstract instance of a system with the characteristics that were described in the previous paragraph, more specifically, a CPS application comprised of three partitions (each corresponding to a guest operating system, two of which are standard Linux and the remaining one hosting a RTOS), one hypervisor managing those partitions, all of which running on a high-performance computing platform Nvidia Jetson TX2. One of the Linux partitions will serve as the interface with the driver, the second Linux partition hosts ROS as its most outstanding application, and the RTOS running in the third partition is assumed to be in charge of managing the sensing of, and the actuation over the physical instruments of the vehicle. The high-level of this CPS system's architecture is illustrated in Figure 3.1.

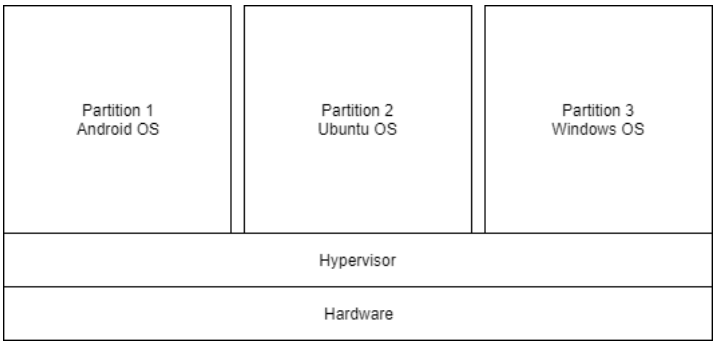

Figure 3.1: Solution assumption

3.2.1 DSL Characteristics

From the architecture of the above described example CPS application, four different views of that application can be identified: (1) a *platform view* (PV), where the hardware platform is chosen and the set of resources that have to be used in the application; a (ii) *hypervisor view* (HV), where one needs to configure how many partitions have to be defined and how they communicate with each other (if they communicate at all); an (iii) *operating system view* (OV) where the OS to be deployed in each partition is selected, and customized to reduce resource usage and potential interference with the applications that will be running on them; and an (iv) *application view* (AV), where the actual computational components

responsible for the functionality of the system are identified, their interactions are defined, and from which verification conditions can be derived.

Each of these views contains specific information which is necessary to be known by other views in order to establish strict boundaries on the support that each view can have or can provide to the other view to which it is directly connected. For instance, the HV must provide information to the OV indicating that only a subset of operating systems is supported, which in turn impacts on the set of components that can be deployed in each of the operating systems that may be chosen. Next, more details about the role of each view in the idealized DSL are provided, as well as the associated syntax of the language while showing how it can be applied to capture the properties of the running CPS application example.

Platform view

The scope of the PV is on defining the hardware platform to be adopted in for the target application, as well as specifying which of its resources are allowed for usage in the upper-level views. In terms of syntax, this represents a view block that is an instance of a Views.Platform template. Since that this example, assumes a NVidia Jetson TX2 as the target hardware platform, a refine statement is defined, whose argument is the name of this embedded board. In the general case, it is assumed that in order to use some hardware platform, some previous work on defining the corresponding stub must be done.

Next, the platform resources that are to be used or not are declared. To allow the usage of a resource, the statement allow is used, followed by the name of the resource (which is defined in the corresponding stub, and is hidden from the developer. For instance, in the example, access to the quad-core ARM A57 available in the Jetson TX2 board is allowed. The usage of ethernet, wireless, serial interfaces for communication, and hdmi support are also allowed. Finally, the usage of all the remaining available resources via the statement deny all is denied, whose intended semantics is that of disallowing access to any resource that was not marked as "allowed" in the view specification.

```
view DemoPV is Views.Platform {
  # Refines existing datasheet based specification for Jetson TX2
  refines JetsonTX2;
  # Allow ARM A57 cores
  allow cores = [1, 2 ,3, 4];
  # Allow Ethernet and WLAN support
  allow ethernet;
  allow wlan;
  # Allow hdmi output
  allow hdmi;
  # Allow serial connection
  allow serial;
  # Block all the rest
  deny other;
  # Define usable memory region for partitions
  region 0x000000 .. 0xFFFFFF : main;
}
```

The relevance of the PV is that it can be used to facilitate the construction of hypervisor and OS distributions, by deselecting drivers for whose corresponding resources are denies of usage, as well as associated services, thus leading to leaner distributions that occupies less memory space (which is very useful when deploying in embedded hardware platforms, which provide less computational resources when compared to regular computers); also, this information can be used to check unwanted access to memory regions that may map hardware resources, for instance, by programmers.

Hypervisor view

The goal of the HV is to allow the selection of the hypervisor and establish its configuration in terms of partitioning and sharing of resources, according to the requirements of the target application. In the example below, which is a view that implements the Views.Hypervisor, the Jailhouse hypervisor is selected via the statement refine Jailhouse. Next, three partitions are defined using the partition keyword, and to each of these partitions, the OS that is expected to be deployed in it is specified as well as what is the memory region in which the OS will be mapped to execute, and which resources each partition it will use. Since Jailhouse is a strong partitioning hypervisor (Baryshnikov 2016), i.e., each hardware resource can be associated with just one partition, we can make use of our view to verify isolation either in terms of memory regions on also guarantee that resources are not wrongly shared among partitions.

```
1  view DemoHyp is Views.Hypervisor {
2    # Bring to scope the selected hw platform and its usage
       constraints
3    import DemoPV;
4    # Refine existing Jailhouse hypervisor support
5    refines Jailhouse;
6    # Specify partition details
7    partition LinuxHMI with {
8     OS Linux;
9     #Specify the main memory region of this partition
10    region 0x000000 .. 0xAAAAA : main;
11    use cores[0], wlan, hdmi;
12   }
13   # Specify partition details
14   partition LinuxROS with {
15    OS Linux;
16    region 0xBBBBBB .. 0xCCCCCC : main;
17    use cores[1..2], ethernet;
18   }
19   # Specify partition details
20   partition RTOS with {
21    OS FreeRTOS;
22    region 0xDDDDDD .. 0xFFFFFF : main;
23    use cores[4], serial;
24   }
25   # Finally, we set up a set of inter-partition communication
       memory zones
26   channel inter_partition_1 with {
27     connect LinuxHMI and LinuxROS;
28     size 10MB;
```

```
29      mode read write to all;
30      protocol IVHSMEM;
31    }
32    channel inter_partition_2 with {
33      connect LinuxROS and RTOS;
34      size 10MB;
35      mode write to RTOS;
36      mode read to LinuxROS;
37      protocol IVHSMEM;
38    }
39  }
```

Another useful specification considered in the HV is that of specifying communication mechanisms between partitions. In the case of Jailhouse, that is possible via shared memory regions that the hypervisor manages. We specify those communication facilities via channel blocks, which relate partitions via the connect statement, after which the size of that communication channel is defined as well as the read/write modes associated with that channel. Also, it is assumed that there may be more than one kind of protocol being used to govern the inter-partition communication channel and for that, the protocol statement is considered, which in the case of Jailhouse is IVHSMEM.

Operating System view

The OV allows for the selection of concrete OS instances, according to what has already been specified in the HV. Not only the concrete OSs are chosen, but also extra packages that need to be installed are selected and the set of services that need to start during boot time are defined. The example below depicts the specification of the OS that shall be deployed in the LinuxROS and RTOS partitions defined in the HV. Again, the advantages of having the OV is that of having the possibility to generate customized configuration and build scripts that lead to OS instances with reduced size, discard applications and services that are not needed, which consequently reduces the changes of unwanted interference among applications packages in the OS and the concrete applications we are targeting to deploy in the final CPS application.

```
1  view LinuxROSImage extends Views.OV {
2   # Clone a pre-existing Ubuntu distribution with minimal software
      packages and services
3   refines Linux.Ubuntu.Minimal;
4   # State that is an instance for a particular partition
5   deploys LinuxROS;
6   # Set extra packages to be installed
7   install ros,...;
8   # Set application to be started at boot time
9   boot { ... }
10 }
```

```
1  view RTOSImage extends Views.OV {
2
```

```
3  refines RTOS.FreeRTOS;
4
5  deploys RTOS;
6
7  boot { ... }
8 }
```

In the above example, two OS images that shall be automatically built by the supporting tools of the DSL are specified. The first one states that a minimal Ubuntu Linux distribution shall be the base, and that it will be deployed on the LinuxROS partition, installing extra packages related to ROS, and boot a set of services that are necessary. The second wrapper simply states that the base OS is FreeRTOS and that it shall be automatically built and deployed on the RTOS partition.

Application View

The last view discussed in the present work is the AV. The goal of this view is similar to that of traditional development interfaces in model or DSL driven development environments. This view provides a set of predefined software components (or templates) to be refined and customized to the CPS application under development. Another aspect considered when specifying AVs is to establish verification conditions that need to be proved to ensure system correctness. To illustrate this, let us first look at the specification of two AVs which focus on a ROS component that needs to be deployed in the LinuxROS partition, and a task that shall be deployed in the RTOS partition. For the case of the ROS based application, an AV that defines which topics are going to be used is specified, as well as an AV that specifies the ROS application using that node.

```
1  view DemoTopic extends Views.AV {
2    # Clone the stub for ROS topics
3    refines ROS.Topics;
4    # ROS bricks to be deployed on previously
5    # specified LinuxROS image
6    deploys LinuxROS;
7    # Specify the set of topics to be considered
8    topic T1 with {
9      path = "path_to_ROS_topic_spec_file" ;
10     type = float ;
11   }
12 }
13
14 view DemoNode extends Views.AV {
15   # Clone the stub for ROS nodes
16   refines ROS.Nodes;
17   # Set the target deployment OS
18   deploys LinuxROS;
19   # Specify the nodes
20   node In_Node with {
21     body "path_to_ROS_node_source_code_file";
22     topics {
23       T1 mode read
```

```
    };
  }
}
```

In terms of specifying topics, that is performed via the writing of topic blocks, where the body parameter defines the source file that contains the actual code that will provide the ROS specification, and the type is the type associated with that topic. These two fields can be used together to derive verification conditions that ensure that any access to the topic is performed using the correct data structures (a sort of static type checking at the DSL level). This is particularly useful when several developers are implementing different nodes accessing to common topics, and typing problems can occur, so the DSL, when being processed, shall warn for those inconsistencies and deny the overall CPS application building process. In the case of the node specification, the approach is very similar, with the exception that we enforce reading/writing modes to associated topics, which can also be verified during analysis or build times of the application, using information already available in the topic's specification.

For specifying a task in within the DSL, the process is very similar to the specification of ROS nodes and topics. The difference relies on the refine argument, which in this case is a refinement of a task in FreeRTOS, and its associated scheduling parameters, which for now consist of only the task's priority and period. Like with the other AVs specified above, that task specification must also point to the concrete file where the code implementing its functional behavior is written.

```
view DemoTask extends Views.AV {
  # Clone the stub for FreeRTOS task nodes
  refines FreeRTOS.Tasks;
  # Set the target deployment OS
  deploys RTOS;
  # Task specification
  task ReadSensor {
    body "path_to_source_code_file" ;
    priority Pr;
    period Per;
  }
}
```

Next, the focus lies on the specification of verification aspects of the DSL. For that, the notion of an AV whose internal components are check and monitor blocks was conceived, where the former establishes formal specifications that shall be verified statically, whereas the latter considers a mechanism that generates monitors that will be coupled with the system and verify its properties during run-time. This is not to be confused with the verification and validation specified in subsection 3.1.4 as these refer to the DSL itself and the verification in the context of the current subsection refers to enabling the user to perform application and system verification, regarding the target system. What is idealized is that the tools that will be built to support development of CPS application with our DSL shall have specific blocks that allow users to define formal specifications and these specifications are checked against the associated verification tools. In the example that follows, some Linear Temporal Logic

(LTL) verification tool to check one property, and a Restricted Metric Temporal Logic with Durations (RMTLD) monitor generation framework that will build a monitor that has the execution semantics of the formula specified, are assumed. Intuitively, the LTL specification states that it is always true that when the ROS node In_Node reads from topic T1 it satisfies property ϕ then, somewhere in the future it must write something to the standard output that satisfies the property φ. Similarly, the specification of the monitor block intuitively means that the node In_Node reading from topic T1 cannot take more than 10 time units to finish.

```
view NodeVerification extends Views.AV {
  # Load all ROS verification stub
  refines ROS.Verification.*;
  # Include nodes and topics
  include DemoNode;
  include DemoTopic;
  # Intra Node Verification (static)
  check {
   module LTL {
    [G](if In_Node.Read.T1 sat ϕ
        then [F](In_Node.Write.stdout sat φ));
   }
  }
  # Property monitoring
  monitor {
   module RMTLD {
    # Check that a message publication takes no
    # more than 10 time units
    [G_{<10}](In_Node.Read.T1 sat true);
  }
}
```

Chapter 4

DSL Implementation

In this chapter, the implementation of the DSL will be presented. As the main language concepts were explained, this chapter will focus on the most technical aspects of the implementation, such as, language syntax and presentation, code generation and intermediate code translation. The first topic to be approached will be the *Language Structure*, followed by the *Language Editor*, *Language Constraints* and, lastly, the *Code Generation* topics. Each of these topics that assemble the language, harbor a number of different concepts of the language.

4.1 Language Structure

In this section the concepts that compose the language are enumerated and explained. These concepts are illustrated in the figures below.

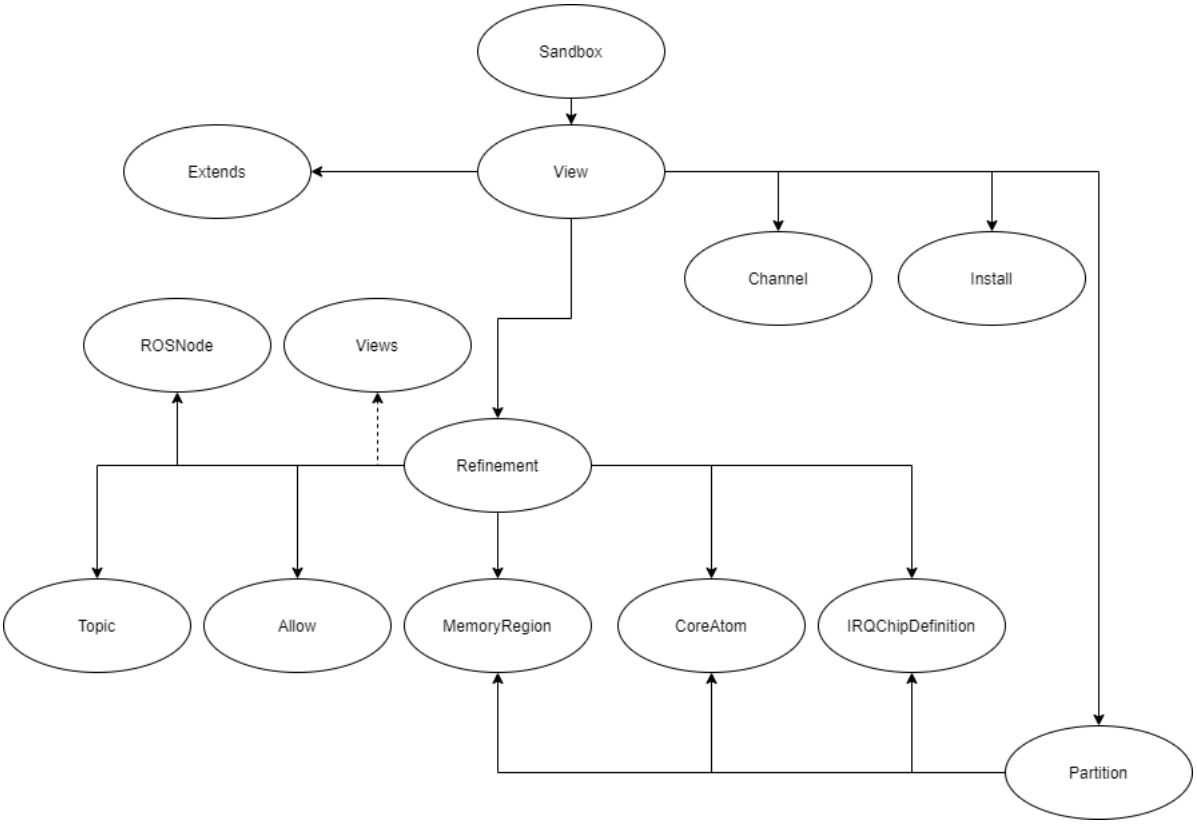

Figure 4.1: Language Concept Structure

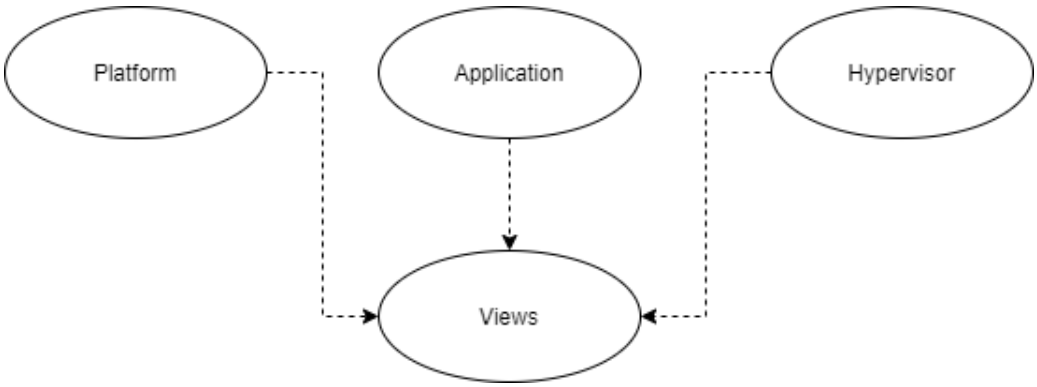

Figure 4.2: Language Concept Structure (2)

Next, follows an explanation of each of these structural concepts and a quantification of its importance to the purpose of this DSL. The analysis will consist of traversing the entirety of the language concepts depicted in the figures above.

4.1.1 Sandbox Concept

Table 4.1: Sandbox Concept Properties

Concept Name	Concept Description
Sandbox	Enables users to write instructions that will then be interpreted and translated into desirable output.

The *Sandbox* Concept is the root concept of the DSL. The name itself does not connect very well with the domain, however, this concept is merely an aggregator of the first real domain concept, which is called *View*. With that being said, this concept possesses a collection of Views. Below lies the concept structure.

```
concept Sandbox extends ClassConcept implements INamedConcept

Children:
View : View[1..n]
```

Listing 4.1: Sandbox Structure

The first thing to mention is the *extends* and *implements* targets specified. Although both of these are not related to the *DSL* whatsoever, they are used by *MPS* to provide certain features to the language, i.e., the *INamedConcept* provides a name property, which can be used to present the user with the name of the concept, if needed, otherwise, this property could be defined manually. Other interfaces or concepts can be utilized here, including the ones related to the DSL

Since this concept is an active concept[1] it possesses an Editor component, in order to format how it is presented to the end user.

```
Editor for Sandbox Concept

{ name }
[/
    (- % View % -)
/]
```

Listing 4.2: Sandbox Editor

In what concerns the *Text Gen* component for this concept, it possesses a definition different from all other concepts, because it is the root concept of the language, with this difference being its ability to define the output file name and extension. The actual text generation invoked within this concept, is basically the invocation of the *TextGen* component of other child concepts, in this case, *Views*.

```
# Obtains TextGen data from all views

(node)->void
  for (node<BaseConcept> view : node.View)
    append ${view} ;
    append \n ;
```

Listing 4.3: Sandbox TextGen

[1] Active concepts are those which are actively utilized by the end user

In this context, the node statement, refers to the current language node for which this text-gen component is being defined. Additionally, it is instanced, meaning the translation is performed for each instance of the language concept that is defined within a specific view.

4.1.2 View Concept

Table 4.2: View Concept Properties

Concept Name	Concept Description	Parent
View	Represents system properties related to specific layers	Sandbox

The *View* Concept allows the conception of parts of a system. It aggregates the most impactful language concepts. As explained in Chapter 3, we can project an aglomerate of system views. This concept is able to describe all of them, and in order to achieve that behavior, concepts like *Extends* were created, i.e., this view *Extends* a subset of properties of the system. To summarize, depending on the selected children (since some are optional) and their values, we achieve a description of a system layer.

```
concept View extends BaseConcept implements INamedConcept

Children:
Extends : Extends[0..1]
Refinement : Refinement[0..1]
Node : Node[0..1]
Install : Install[0..1]
Partition : Partition[0..n]
Channel : Channel[0..n]
Monitor : Monitor[0..1]
```

Listing 4.4: View Structure

As this concept is active, it possesses an Editor component that is used to present the concept to the end user in the format specified below:

```
Editor for View Concept

[- view { name } is % Extends % { -]
[- % Refinement % -]
[- % Install % -]
[-
  (- % Partition % -)
-]
[-
(- % Channel %  -)
-]
[- % Monitor % -]
[- } -]
```

Listing 4.5: View Editor

As stated, each of these children's purpose and properties shall be described, before traversing down the language tree.

Since we have no specific behavior associated to this concept, the *Text Gen* component merely obtains data of each of the child nodes. If this concept had behavior associated[2], as some will have, the *Text Gen* component would also reflect that behavior.

```
# Obtains TextGen data from all views

(node)->void
  if (node.Refinement.target != null) {
    append ${node.Refinement} ;
  }
  for (node<Partition> part : node.Partition) {
    append ${part} ;
  }
  if (node.Install != null) {
    append ${node.Install} ;
  }
  for (node<Channel> ch : node.Channel) {
    append ${ch} ;
  }
  if (node.Monitor != null) {
    append ${node.Monitor} ;
  }
```

Listing 4.6: View TextGen

Next, the *Extends* concept will be described.

4.1.3 Extends Concept and the Views Interface

Table 4.3: Extends Concept Properties

Concept Name	Concept Description	Parent
Extends	Allows the extension of a given target	View

The *Extends* Concept, in essence, enables extension of a given View. Below, the properties of this concept are illustrated.

```
concept Extends extends BaseConcept implements <None>

Children:
Target : Views[1]
```

Listing 4.7: Extends Structure

The Views target, in this case, its an interface. This interface is implemented by four targets, one for each system view, namely the Platform, Hypervisor, Operating System and Application. By declaring the Views concept as a child, we enable the Extends concept to have controlled access to each of the specified views. This control provided by the extension of such concepts is performed by each concept, i.e., each concept chooses what to share with the interface. This is useful because, merely by extending a view, we provide, by design, an environment where the user could do no wrong (i.e., extending one target then

[2]In this case, behavior refers to a set of instructions that allows us to manipulate information from that specific node

providing configurations for another target) since the properties available to operate refer to the extension target itself.

As this concept is active, it possesses an Editor component that is used to present the concept to the end user in the format specified below:

```
Editor for Extends Concept

[- % Target % -]
```

Listing 4.8: Extends Editor

Basically, this concept exclusively uses the editor of its child, which means it will present what each extension target defines. Next, we describe the *Refinement* concept, which is another child of the *View* concept and sibling of the *Extends* concept.

4.1.4 Refinement Concept and the Constants Interface

Table 4.4: Refinement Concept Properties

Concept Name	Concept Description	Parent
Refinement	Enables customization of a given set of properties related to a View	View

Before we describe the *Refinement* Concept, it is important to point out the purpose of the *Constants* Interface, since this is the first concept that implements it. The purpose of the *Constants* Interface is to provide and share information between the different concepts (for example, file names, file paths, search terms, file organizational patterns). This information is then used by various concepts in the process of translation to the target language.

The purpose of the *Refinement* Concept is to customize properties related to the refinement target. It is implemented in such a way that it depends on the *Extension target of the view* where *Refinement* is declared, meaning *Refinement* can provide different outputs not only defined by its target, but also by the current system view being described. This will be explained further within this subsection, on the *TextGen* component.

Below, the properties of this concept are described.

```
concept Refinement extends BaseConcept implements Constants

Children:
allowList   : Allow[0..n]
memRegions  : MemoryRegion[0..n]
cores       : Cores[0..1]
irqChips    : IRQChipDefinition[0..n]
topics      : Topic[0..n]
nodes       : ROSNode[0..n]
```

Listing 4.9: Refinement Structure

Since this is an active concept, it possesses an Editor Component, listed below.

```
Editor for Refinement Concept

[- refines { target } ; -]
```

```
[- use % Cores % -]
(- % allowList % -)
[-
  (- % memRegions % -)
-]
[-
  (- % irqchips % -)
-]
[-
  (- % topics % -)
-]
[-
  (- % nodes % -)
-]
```

Listing 4.10: Refinement Editor

The *TextGen* component for this concept is listed below. There we verify the target platform, and this is very important because it indicates that there is a dependency on the hardware. This means that for each new hardware component, support for it needs to be added within the language, due to configuration specificity for each hardware component. In this case, we added compatiblity with the JetsonTX2 board, due to its compatibility with the *Jailhouse* hypervisor.

After checking if hardware corresponds to the expected, we then obtain a reference to the *Extends* node to see in which system view we are operating. In this case we have but one possible output, and the system view expected is the *Platform View*. If we needed a different behavior for another view, i.e., sharing other properties beyoned the Target property, we would need to add support for that in the same fashion.

```
# Obtains TextGen data from Refinement target as well as its properties

(node)->void
 if (node.target.equals(JETSON_PLATFORM)) {
  append {#### START_REFINEMENT_JETSONTX2 ####} ;
  append \n ;
  node<> extendsRef = node.parent.children.findFirst({~it => it.
    isInstanceOf(Extends); });
  node<Extends> ex = ((node<Extends>) extendsRef);
  if (ex.Target instanceof node<Platform>) {
    append ${ex.Target} ;
  }
  append \n ;
  linkedlist<string> configs = new linkedlist<string>;
  for (node<Allow> allowNode : node.allowList) {
    configs.add(allowNode.target);
  }

  for (string configLine : node.configure(configs, false)) {
    append ${configLine} ;
    append \n ;
  }

  append {#### END_REFINEMENT_JETSON_TX2 ####} ;
  append \n ;

 if (node.target.equals("ROS.Topics")) {
  append {#### START_REFINEMENT_ROS ####} ;
```

```
28  append \n ;
29  node<> extendsRef = node.parent.children.findFirst({~it => it.
      isInstanceOf(Extends); });
30  node<Extends> ex = ((node<Extends>) extendsRef);
31  if (ex.Target instanceof node<Application>) {
32    append ${ex.Target} ;
33  }
34  append {#### END_REFINEMENT_ROS ####} ;
35  append \n ;
```

Listing 4.11: Refinement TextGen

Within this concept, we have our first Behavior component. As the *TextGen* component prepares the intermediary language, it may need help parsing files and translating to the target language, hence the calling of the *node.configure* behavioral function, with node being the current language node being processed, and the configure statement being a behavioral function.

The purpose of this behavior within the *Platform View* is to take what pieces of the system (i.e., hardware) the user wants to allow within the kernel configuration, and map them to configuration entries within the kernel configuration file itself.

```
1
2  Generates KConfig from a list of allowed configs
3
4  @param config Linked list of configs that are allowed, i.e., = y
5  @param denyAll Deny all that are =y and not in the list of configs
6  @return a List with all configs
7
8  public LinkedList<string> configure(linkedlist<string> config, boolean
       denyAll) {
9    try {
10
11     Path p = FileSystems.getDefault().getPath(this.
       KERNEL_CONFIG_FILE_PATH, this.KERNEL_CONFIG_FILE_NAME);
12     Path out = FileSystems.getDefault().getPath(this.
       KERNEL_CONFIG_OUTPUT_FILE_PATH, this.KERNEL_CONFIG_OUTPUT_FILE_NAME);
13
14     List<string> list = Files.readAllLines(p);
15     LinkedList<string> listConfigs = new LinkedList(list);
16     LinkedList<string> finalList = new LinkedList(list);
17
18     for (string kernelCfgEntry : listConfigs) {
19       for (string configSingle : config) {
20
21         if (kernelCfgEntry.contains(configSingle.toUpperCase()) &&
       kernelCfgEntry.contains("is not set")) {
22           string newEntry = kernelCfgEntry.replace(this.KERNEL_NOT_SET,
       this.KERNEL_ALLOW_TERM).replace("# ", "");
23           int index = listConfigs.indexOf(kernelCfgEntry);
24           finalList.remove(index);
25           finalList.add(index, newEntry);
26           continue;
27         }
28
29         if (denyAll && kernelCfgEntry.contains(this.KERNEL_ALLOW_TERM)
       && !kernelCfgEntry.contains(configSingle.toUpperCase())) {
30           string newEntry = "# " + kernelCfgEntry.replace(this.
       KERNEL_ALLOW_TERM, this.KERNEL_NOT_SET);
```

```
31            int index = listConfigs.indexOf(kernelCfgEntry);
32            finalList.remove(index);
33            finalList.add(index, newEntry);
34          }
35        }
36      }
37
38      Files.write(out, finalList);
39      return finalList;
40    } catch (Exception ex) {
41      return null;
42    }
43  }
```

Listing 4.12: Refinement main behavioral function

Within this piece of code, we can see various Constants from the implemented interface being queried. With this being said, it is pertinent, before moving on to the next child of the *View* concept, explain the *Allow* concept, since it was referred to widely within the current concept.

4.1.5 Allow Concept

Table 4.5: Allow Concept Properties

Concept Name	Concept Description	Parent
Allow	Enables access to hardware. Impacts Linux kernel configuration.	Refinement

The *Allow* Concept is directly connected to the Platform View, described in Chapter 3.2. It provides access to hardware components, i.e., Wireless LAN, or HDMI. This access is provided by configuring kernel properties that are directly connected with those components. This means that the only property that this concept needs, can be encapsulated within a String.

```
1 concept Allow extends BaseConcept implements <none>
2
3 properties:
4 target : string
```

Listing 4.13: Allow Structure

The target property, is mapped to a simple string. It can, then, be directly compared with relevant kernel entries, i.e., HDMI would enable all kernel entries related with HDMI. The user can distinguish different HDMI kernel options by being providing a more specific target than just HDMI, for example HDMI-OUTPUT.

```
1 Editor for Allow Concept
2
3 [- allow { target } ; -]
```

Listing 4.14: Allow Editor

This component, because of its nature, is only utilized within the *Platform View*, therefore, it does not get behavior directly associated to it. Instead, its desired behavior is deferred to the Refinement Concept, as stated within the explanation of that concept.

4.1.6 MemoryRegion Concept

Table 4.6: MemoryRegion Concept Properties

Concept Name	Concept Description	Parent
MemoryRegion	Defines and controls memory regions for the Platform and Partitions	Refinement/Partition

The *MemoryRegion* Concept allows the definition of memory regions by the *Platform View*. Additionally, it allows the definition and control of memory regions by the *Hypervisor View*. This is because the hypervisor can partition memory contained within the system to create partitions inside that same system, to provide isolated environments to applications, if needed. The *Partition Concept* is analyzed on subsection 4.1.12.

Below, lie the properties of this concept.

```
concept MemoryRegion extends BaseConcept implements Constants

properties:
memStart   : string
virtStart  : string
size       : string
target     : string
```

Listing 4.15: MemoryRegion Structure

The memStart property describes the address where the memory reservation starts. The virtStart property, defines how this is mapped into virtual memory inside the system, i.e., it could have a different value inside each partition. The size property is related to the reserved memory size and the target property is related to the peripheral that requires such memory.

This is an active concept, therefore, possesses an Editor component, listed below.

```
Editor for MemoryRegion Concept

[- region { memStart } .. { virtStart } : { size } : { target } -]
```

Listing 4.16: MemoryRegion Editor

Next, follow the first case of Constraint definition within the DSL. The purpose of constraining this concept, is to make sure that when memory regions are defined within the *Hypervisor View*, they respect what was defined within the *Platform View*. These constraints are applied while the user is actively utilizing the language. We state that this property is valid, when the regions defined within the *Hypervisor View* are bounded (addresses + size) within the system memory defined on *Platform View*, for the same targets, and, for the total memory of the system. This process is listed below.

```
# Memory Constraints

```

```
property {memStart}
  is valid (propertyValue, node)->boolean {
    try {
      node<> viewRef = node.parent.parent;
      node<View> vRef = (node<View>) viewRef;
      if (vRef.Extends.Target.isInstanceOf(Hypervisor)) {
        node<> sandBoxRef = vRef.parent;
        node<Sandbox> sRef = ((node<Sandbox>) sandBoxRef);
        for (node<View> view : sRef.View) {
          if (view.Extends.Target.isInstanceOf(Platform)) {
            for (node<MemoryRegion> memReg : view.Refinement.memRegions)
    {
              if (node.target.equals(memReg.target)) {
                if (node.target.equals("RAM") || node.target.equals("
    IVSHMEM")) {
                  if (Long.parseLong(propertyValue.substring(2), 16) <
    Long.parseLong(memReg.memStart.substring(2), 16)) {
                    return false;
                  }
                  return true;
                }
                if (Long.parseLong(propertyValue.substring(2), 16) !=
    Long.parseLong(memReg.memStart.substring(2), 16)) { return false; }
                return true;
      if (vRef.Extends.Target.isInstanceOf(Platform) && node.memStart !=
     null) {
        for (node<MemoryRegion> memReg : vRef.Refinement.memRegions) {
          if (node != memReg) {
            long memStart = Long.parseLong(memReg.memStart.substring(2),
     16) + Long.parseLong(memReg.size.substring(2), 16);
            if (node.memStart != null && Long.parseLong(propertyValue.
    substring(2), 16) >= Long.parseLong(memReg.memStart.substring(2), 16)
     && Long.parseLong(propertyValue.substring(2), 16) < memStart) {
              warn Long.parseLong(propertyValue.substring(2), 16) + ":"
    + memStart;
    catch (Exception ex) {
      warn "", <no project>, ex;
      return false;
    }
    return true;
  }
```

Listing 4.17: memStart property constraint

```

property {virtStart}
  is valid (propertyValue, node)->boolean {
    try {
      node<> viewRef = node.parent.parent;
      node<View> vRef = (node<View>) viewRef;
      if (vRef.Extends.Target.isInstanceOf(Hypervisor)) {
        node<> sandBoxRef = vRef.parent;
        node<Sandbox> sRef = ((node<Sandbox>) sandBoxRef);
        for (node<View> view : sRef.View) {
          if (view.Extends.Target.isInstanceOf(Platform)) {
            for (node<MemoryRegion> memReg : view.Refinement.memRegions)
    {
              if (node.target.equals(memReg.target)) {
                if (Long.parseLong(propertyValue.substring(2), 16) <
    Long.parseLong(memReg.virtStart.substring(2), 16)) { return false; }
```

```
15
16      } catch (Exception ex) {
17        return false;
18      }
19      return true;
20    }
21
```

Listing 4.18: virtStart property constraint

```
1
2  property {size}
3    is valid (propertyValue, node)->boolean {
4      try {
5        if (node.target.contains("RAM") || node.target.contains("IVSHMEM")
       ) {
6          node<> viewRef = node.parent.parent;
7          node<View> vRef = (node<View>) viewRef;
8          if (vRef.Extends.Target.isInstanceOf(Platform)) { return true; }
9          node<> sandBoxRef = vRef.parent;
10         node<Sandbox> sRef = ((node<Sandbox>) sandBoxRef);
11         long total = 0L;
12         if (vRef.Extends.Target.isInstanceOf(Hypervisor)) {
13           for (node<View> view : sRef.View) {
14             if (view.Extends.Target.isInstanceOf(Hypervisor)) {
15               for (node<Partition> partition : view.Partition) {
16                 for (node<MemoryRegion> memReg : partition.MemoryRegion)
       {
17                   if (node.target.equals(memReg.target) && memReg !=
       node) {
18
19         for (node<View> view : sRef.View) {
20           if (view.Extends.Target.isInstanceOf(Platform)) {
21             for (node<MemoryRegion> memReg : view.Refinement.memRegions)
       {
22               if (node.target.equals(memReg.target)) {
23                 if (total + Long.parseLong(propertyValue.substring(2),
       16) > Long.parseLong(memReg.size.substring(2), 16)) { return false; }
24
25     } catch (Exception ex) {
26       return false;
27     }
28     return true;
29   }
30
```

Listing 4.19: size property constraint

```
1  property {target}
2    is valid (propertyValue, node)->boolean {
3      node<> viewRef = node.parent.parent;
4      node<View> vRef = (node<View>) viewRef;
5      if (vRef.Extends.Target.isInstanceOf(Hypervisor)) {
6        for (node<Partition> part : vRef.Partition) {
7          for (node<MemoryRegion> memReg : part.MemoryRegion) {
8            if (node != memReg && memReg.target != null && memReg.target.
       equals(propertyValue) &&
9            !(propertyValue.equals("RAM") || propertyValue.equals("IVSHMEM
       ") {return false; }
10     return true;
```

```
11  }
```

Listing 4.20: target property constraint

The figure below represents the error users find when going out of bounds in memory definitions. In a system with *0x4000* bytes of memory defined for RAM, the sum of the sizes of RAM defined by two partitions (*0x2000 + 0x3000*) exceeds the total size.

```
  partition LinuxROS with {
    OS linux ;
    region 0x100000000 .. 0x100000000 : 0x2000 : UART
    region 0x5000000000 .. 0x50000000 : 0x2000 : RAM
    irqchip 0x30000000 .. 288 : 0 << (332 - 320) : GIC
    device IVSHMEM , region 3
    use cores [ 0 , 1 , ] , wlan, hdmi
  }
  partition LinuxROS2 with {
    OS linux ;
    region 0x5000000000 .. 0x50000000 : 0x3000 : RAM
    << ... >>
    << ... >>
    use cores [ 2 , 3 , ] , wlan
  }
<< ... >>
  <no Monitor>
}
view DemoPV is Views.Platform {
  <no Import>
  refines JetsonTX2 ;
    use cores [ 0 , 1 , 2 , 5 , ]
    allow wlan ;
    allow hdmi ;
    region 0x100000000 .. 0x100000000 : 0x1000 : UART
    region 0x5000000000 .. 0x50000000 : 0x4000 : RAM
```

Figure 4.3: Unbounded memory error

The next concept to be described is the *CoreAtom* concept.

4.1.7 CoreAtom Concept

Table 4.7: CoreAtom Concept Properties

Concept Name	Concept Description	Parent
CoreAtom	Enables the use of CPU cores to the Hypervisor	Refinement/Partition

The *CoreAtom* Concept allows the user to define what cores are available in the system, from within the *Platform View*. These cores can then be utilized within the *Hypervisor View* by allocating them to partitions. This means that each partition will have its own, isolated processing power, and, therefore, guarantee a degree of isolation between partitions. Additionally, this means that each partition can run safe critical applications, as each partition would be using the allocated CPU cores to schedule relevant tasks safe from problems like task preemption that other partitions would probably cause if CPU cores were shared. Below, the properties of this concept are depicted.

```
concept CoreAtom extends BaseConcept implements Constants

properties:
coreIdentifier   : integer
coreEnabled      : boolean
```

Listing 4.21: CoreAtom Structure

This concept possesses two simple properties, the identifier, which is an integer, and a property that states if the core is actively being utilized. This concept is active and therefore it is utilized by the user. Below lies its editor component.

```
Editor for CoreAtom Concept

[- { identifier } , -]
```

Listing 4.22: CoreAtom Editor

This concept does not generate text directly, instead, it gets queried by other concepts which generate relevant code. However, this concept does have constraints associated. The purpose is to guarantee that the cores utilized within partitions when defining the *Hypervisor View* were actually declared within the *Platform View* definition.

```

property {identifier}
  is valid (propertyValue, node)->boolean {
    node<> viewRef = node.parent.parent.parent;
    node<View> vRef = (node<View>) viewRef;
    if (vRef.Extends.Target.isInstanceOf(Hypervisor)) {
      for (node<Partition> part : vRef.Partition) {
        for (node<CoreAtom> coreAtom : part.Cores.cores) {
          if (node != coreAtom && propertyValue == coreAtom.identifier)
   { return false; }
        }
      }
      node<Refinement> refinementRef = vRef.Refinement;
      if (refinementRef.Cores.cores.contains(node)) { return false; }
    }
    return true;
  }
```

Listing 4.23: identifier property constraint

If the identifier of these cores is not present in the *Platform View* definition, or if we state repeated cores within sibling partition definitions, an error is displayed to the user, as illustrated in the figure below.

```
    partition LinuxROS with {
      OS linux ;
      region 0x100000000 .. 0x100000000 : 0x2000 : UART
      region 0x5000000000 .. 0x50000000 : 0x1000 : RAM
      irqchip 0x30000000 .. 288 : 0 << (332 - 320) : GIC
      device IVSHMEM , region 3
      use cores [ 0 , 1 , 2 , ] , wlan, hdmi
    }
    partition LinuxROS2 with {
      OS linux ;
      region 0x5000000000 .. 0x50000000 : 0x3000 : RAM
      << ... >>
      << ... >>
      use cores [ 2 , 5 , ] , wlan
    }
  << ... >>
    <no Monitor>
  }
  view DemoPV is Views.Platform {
    <no Import>
    refines JetsonTX2 ;
      use cores [ 0 , 1 , 2 , 5 , ]
```

Figure 4.4: Core allocation error

The next concept to be analyzed is the IRQChipDefinition concept.

4.1.8 IRQChipDefinition Concept

Table 4.8: IRQChipDefinition Concept Properties

Concept Name	Concept Description	Parent
IRQChipDefinition	Defines Interrupt Request chip properties	Refinement/Partition

The *IRQChipDefinition* Concept largely follows the implementation followed by the MemoryRegion concept, explained on sub-section 4.1.6. In essence, from the *Platform View* definition, the user can define Interrupt Request chips memory addresses. These are used to send signals to the CPU to interrupt the current routine under execution and run a rountine called *Interrupt Handler*. These defined regions can then be used by partition definitions within the *Hypervisor View*, of course, bounded by some constraints.

This is a feature already supported by the *Jailhouse Hypervisor*, and the implementation for this language is exclusively for configuration purposes. As such, there is no way to oversimplify this for the end user, as we have to assume there is some prior knowledge about such feature in order to correctly produce a configuration entry about the matter.

Below lie the properties and children of the IRQChipDefinition concept.

```
1
2 concept IRQChipDefinition extends BaseConcept implements Constants
3
4 properties:
5 address   : string
6 pinBase   : string
7 target    : string
```

```
8
9  children:
10 pinBitmap : PinBitmap[1]
```

Listing 4.24: IRQChipDefinition Structure

The address property states the location in memory of this interrupt chip. Unlike the addresses defined under the bounds of the *MemoryRegion* concepts, this address does not have a range associated, because its utilization and operation is responsibility of the chip itself, under the hypervisor. As such, the purpose is to indicate only its location in memory and forward that to the hypervisor configuration.

The pinBase is a property that, when added to the pinBitmap is used to calculate the number for the interrupt line, associated to the *Jailhouse cell* under configuration.

Since this is an active concept, it possesses an editor component, described below.

```
1 Editor for IRQChipDefinition Concept
2
3 [- irqchip { address } .. { pin_base } : % pinBitmap % : { target } -]
```

Listing 4.25: IRQChipDefinition Editor

This concept, similarly to the *MemoryRegion* concept, does not possess a *TextGen* component as its properties are aggregated and translated by the *Hypervisor* concept. Instead, this concept has its own Constraints component, where its properties are bounded by rules.

Below, the constraints that bound this concept are displayed.

```
1
2  property {target}
3    is valid (propertyValue, node)->boolean {
4      node<> viewRef = node.parent.parent;
5      node<View> vRef = (node<View>) viewRef;
6      if (vRef.Extends.Target.isInstanceOf(Hypervisor)) {
7        for (node<Partition> part : vRef.Partition) {
8          for (node<IRQChipDefinition> irqDef : part.irqChips) {
9            if (node != irqDef && irqDef.target != null && irqDef.target.
     equals(propertyValue)) { return false; }
10     return true;
11   }
```

Listing 4.26: target property constraint

```
1
2  property {address}
3   is valid (propertyValue, node)->boolean {
4       node<> viewRef = node.parent.parent;
5       node<View> vRef = (node<View>) viewRef;
6       if (vRef.Extends.Target.isInstanceOf(Hypervisor)) {
7         node<> sandBoxRef = vRef.parent;
8         node<Sandbox> sRef = ((node<Sandbox>) sandBoxRef);
9         for (node<View> view : sRef.View) {
10          if (view.Extends.Target.isInstanceOf(Platform)) {
11            for (node<IRQChipDefinition> irqDef : view.Refinement.
     irqchips) {
12              if (node.target.equals(irqDef.target)) {
13                if (Long.parseLong(propertyValue.substring(2), 16) !=
     Long.parseLong(irqDef.address.substring(2), 16)) { return false; }
```

```
14                  return true;
15          if (vRef.Extends.Target.isInstanceOf(Platform) && node.address !=
    null) {
16              for (node<IRQChipDefinition> irqDef : vRef.Refinement.irqchips)
    {
17                  if (node != irqDef) {
18                      long addrStart = Long.parseLong(irqDef.address.substring(2),
     16);
19                      if (node.address != null && Long.parseLong(propertyValue.
    substring(2), 16) == Long.parseLong(irqDef.address.substring(2), 16))
     {
20                          warn Long.parseLong(propertyValue.substring(2), 16) + ":"
    + addrStart, <no project>;
21                          return false;
22              return true;
23          }
```

Listing 4.27: address property constraint

When utilizing this concept to define a partition, inside the *Hypervisor View*, the *target* and *address* properties must match what was defined for the same IRQChip within the *Platform View*. Failure to do so, will result in the following error, showed to the end user:

```
partition LinuxROS with {
  OS linux ;
  region 0x100000000 .. 0x100000000 : 0x2000 : UART
  region 0x5000000000 .. 0x50000000 : 0x1000 : RAM
  irqchip 0x30000001 .. 282 : 0 << (332 - 320) : GIC
  device IVSHMEM , region 3
  use cores [ 0 , 1 , ] , wlan, hdmi
}
partition LinuxROS2 with {
  OS linux ;
  region 0x5000000000 .. 0x50000000 : 0x3000 : RAM
  << ... >>
  << ... >>
  use cores [ 2 , 5 , ] , wlan
}
<< ... >>
  <no Monitor>
}
view DemoPV is Views.Platform {
  <no Import>
  refines JetsonTX2 ;
  use cores [ 0 , 1 , 2 , 5 , ]
  allow wlan ;
  allow hdmi ;
  region 0x100000000 .. 0x100000000 : 0x1000 : UART
  region 0x5000000000 .. 0x50000000 : 0x4000 : RAM
  region 0x275000001 .. 0x275000001 : 0x1000 : IVHSMEM  2
  irqchip 0x30000000 .. 288 : 0xfffffff 0xffffffa : GIC
```

Figure 4.5: IRQ Chip wrong memory address error

Next, an analysis of the *Topic* language concept will be performed.

4.1.9 Topic Concept

Table 4.9: Topic Concept Properties

Concept Name	Concept Description	Parent
Topic	Defines a ROS Topic to be broadcast	Refinement

The *Topic* Concept is a simple concept that defines two properties of a Robot Operating System topic. In essence, a ROS topic contains a strongly defined type and a path, as stated by Koubâa et al. 2017. With that being said, our concept implementation roughly follows the same pattern.

```
concept Topic extends BaseConcept implements ROS, INamedConcept

properties:
path       : string
type       : string
```

Listing 4.28: Topic Structure

This is an active concept, therefore possesses an editor component, listed below.

```
Editor for Topic Concept

[- Topic { name } with { -]
[- path: { path } -]
[- type: { type } -]
```

Listing 4.29: Topic Editor

As the properties of this concept are queried by the *Refinement* language concept, described on subsection 4.1.4, this concept does not possess a *TextGen* component or a *Behavior* component.

Next, the implementation of the *ROSNode* concept is described.

4.1.10 ROSNode Concept & the Permissions and TopicInNode Subconcepts

Table 4.10: ROSNode Concept Properties

Concept Name	Concept Description	Parent
ROSNode	Allows the definition of a ROS Node	Refinement

The *ROSNode* Concept enables the creation of a ROS Node from within the *Application View*. A ROS node is able to connect with other nodes to exchange information, normally utilized to publish or subscribe to a given ROS Topic. Below, are the properties of this concept.

```
concept ROSNode extends BaseConcept implements INamedConcept, ROS

properties:
body       : string
```

```
topics     : TopicInNode[0..1]

children:
```

Listing 4.30: ROSNode Structure

The TopicInNode and Permissions subconcepts[3] possess the following attributes:

```
concept TopicInNode extends BaseConcept implements INamedConcept

children:
flags      : Permissions[0..n]
```

Listing 4.31: TopicInNode Structure

```
concept Permissions extends BaseConcept implements INamedConcept

properties:
target          : string

children:
permission      : string[1..n]
```

Listing 4.32: Permissions Structure

The permissions subconcept (as part of the ROSNode concept) controls the operations able to be performed by the current node, to a given topic. This impacts the code generation for the ROS applications and configuration files.

As the *ROSNode* concept is active, it possesses an editor component, listed below.

```

Editor for ROSNode Concept

[- node { name } with { -]
    [- body: { body } -]
    [- topics: { -]
    [- % topics % -]
[- } -]

Editor for the Permissions node (utilized inside the ROSNode concept)

[- mode (- % permission % -) to { target } ; -]
```

Listing 4.33: ROSNode and Permissions Editor

This concept does not possess a *TextGen* component, all data from this concept is collected by the *Application* concept, described in subsection 4.1.16, to then be translated into the target language.

Since all child concepts of *Refinement* have been analyzed, the next concept to be explained will be the *Install* concept, which is a critical concept for the DSL.

[3] A subconcept is a language concept that is used primarily to organize how the language is shown to the end user, and it usually doesn't impact the business logic

4.1.11 Install Concept and the InstallTableLines subconcept

Table 4.11: Install Concept Properties

Concept Name	Concept Description	Parent
Install	Enables installation of a given unix based OS by providing the necessary configuration scripts	View

The *Install* Concept aids the user in conceiving the necessary artifacts to configure and install an operating system, having into consideration a kernel version, the hardware upon which the OS will be installed and packages to be installed upon boot of this new operating system. In essence, this concept uses the created artifacts to produce a custom root file system (rootfs), with our changes published into it, that is then transformed in an installation disk. With that being said, the context upon which this concept operates is the *Operating System View*. Below, the properties that constitute this concept are described.

```
concept Install extends BaseConcept implements Constants

properties:
OS            : string
kernelVersion : string
boardTarget   : string
preInstalled  : boolean

children:
targets : InstallTableLines[1..n]
```

Listing 4.34: Install Structure

The OS property refers to the target OS the user aims to install, i.e., *Ubuntu Server*. The kernel version, refers to the kernel configuration aspects described within subsection 4.1.4. Essentially, the OS View is tasked with the operations of actually obtaining the kernel sources, which is then configured by defined parameters, as described in subsection 4.1.4. The boardTarget property lets the user choose the hardware upon which the target OS shall be installed. The specified hardware does not have to match 1:1 with the hardware specified within the *Platform View*, because, in this case, the issue is to try to find compatibility issues between the specified OS and the target board, and also aiding in finding pre-installed images[4], which is what the preInstalled parameter is about. Even if the image is pre-installed, package installation scripts generated from this concept can still be applied manually, which is useful in cases where the advantages of a pre-installed image outweigh the disadvantages of configuring the entire image.

The InstallTableLines subconcept is tasked with enabling the user to define packages to be installed within the newly configured operating system, and it provides scripts that are injected within the root file system, to be executed by the user upon booting the new OS, as the necessary scripts are already available. One could argue that these packages could be cross compiled within the host machine and then be injected within the rootfs, however, the advantages of simply providing these scripts far outweigh the disadvantages created by the complexity of configuring cross compilation for a given host machine.

[4] A pre-installed image already has the OS installed and customized for a given board

This concept is active and therefore possesses an editor component, however, due to the size of this component, in order to provide an intuitive presentation to the user, it is listed in Appendix C.

The figure below illustrates how this concept is presented to the user.

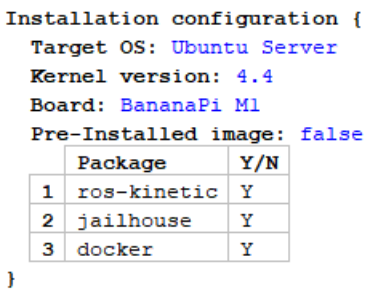

Figure 4.6: Installation concept presentation

This concept possesses both a *TextGen* component and a *Behavior* component. The *TextGen* component is listed below.

```
# Obtains TextGen data from Install target as well as its properties

append {#### START_INSTALL_INFORMATION ####} ;
append \n ;

linkedlist<string> configs = new linkedlist<string>;
configs.add("OS:" + node.OS);
configs.add("kernel:" + node.kernelVersion);
configs.add("target:" + node.boardTarget + ":" + node.preinstalled);

for (node<InstallTableLines> lines : node.targets) {
  configs.add("line:" + lines.descripton + ":" + lines.value);
}

for (string scriptLine : node.configureInstall(configs)) {
  append ${scriptLine} ;
  append \n ;
}

append {#### END_INSTALL_INFORMATION ####} ;
append \n ;
```

Listing 4.35: Install TextGen

From this component, the values extracted from properties are transformed into an intermediate language[5]. This intermediate language is then sent to the *Behavior* component to process the rest of the translation. The extracted properties that are transformed in the intermediate language, are depicted below.

```
#Intermediate language for install concept

OS:Ubuntu Server
kernel:4.4
```

[5]In this context, intermediate language consists of a step usually taken when translating from a DSL into a target language, which aids in the translation process

```
target:BananaPi M1:false
line:ros-kinetic:Y
line:jailhouse:Y
line:docker:Y
```

Listing 4.36: Install intermediate language

The *Behavior* component, as stated, takes this intermediary language and submits it to the behavioral function *configureInstall*. The logic necessary for this concept is illustrated within Appendix D due to its size. Within the behavioral function, there are some statements that seem static even before generating the script files. Although it may not be feasible to automate the entirety of the process, most of the download links, i.e., kernel downloads, are located within the Constants interface, but there are things that seem hard-coded within the function, although one could argue that it maybe not be stable as a long term solution, the purpose in the long run would be for the maintainers of this DSL to also maintain a secure repository for the supported assets of the DSL, that would maintain stable versions of, for example, the Jailhouse binaries or even the ROS and Docker packages, instead of cloning them from unstable repositories.

4.1.12 Partition Concept and the PCIDevices subconcept

Table 4.12: Partition Concept Properties

Concept Name	Concept Description	Parent
Partition	Defines a virtualized environment that may be isolated from other environments	View

The *Partition* Concept, as briefly stated on subsection 4.1.6, is essentially a virtualized environment, provided and managed by the hypervisor. This environment may be isolated from others alike, and in doing so, freedom from interference is achieved. This means that applications with different criticality levels can execute without preemptions or interruptions that stem from other applications. Of course, in a bare-metal setting this is verified more often that in a general setting where these applications execute on top of an Operating System. The properties for the *Partition* Concept are listed below.

```
concept Partition extends BaseConcept implements Constants

properties:
OS          : string

children:
cores       : Cores[1]
memoryRegion : MemoryRegion[1..n]
irqChips    : IrqChipDefinition[0..n]
pciDevices  : PciDevices[0..n]
```

Listing 4.37: Partition Structure

Most of these properties are concepts already visited, like the *Cores, MemoryRegion, IRQChipDefinition*. The PciDevices subconcept allows the definition of a PCI device to be configured within a certain memory region index. This is used to configure the *Jailhouse* hypervisor. Below lies the editor for this concept as well as the PCIDevices subconcept.

```
#Editor for Partition Concept

[- partition { name } with { -]
[- OS { OS } ; -]
[-
  (- % MemoryRegion % -)
-]
[-
  (- % irqChips % -)
-]
[-
  (- % pciDevices % -)
-]
[- use % Cores % , { cores } -]


#Editor for PCIDevices subconcept

[- device { type } , region { region } -]
```

Listing 4.38: Partition Editor

This concept does not possess either a *TextGen or Behavior* component, since the properties listed, are processed into target language by the *Hypervisor* concept, within subsection 4.1.15.

Next, the last child from the *View* Concept is analyzed, namely, the *Channel* Concept.

4.1.13 Channel Concept and the Connect subconcept

Table 4.13: Channel Concept Properties

Concept Name	Concept Description	Parent
Channel	Creates communication channels between partitions	View

The *Channel* Concept focuses on removing the limitations imposed by a partitioning hypervisor. While partitioning the physical resources is very useful in a critical setting, there are cases where the ability to exchange information between partitions within a controlled environment may be needed. Under the influence of the hypervisor, there are various ways to share information between partitions. They may communicat with each other directly, or use the hypervisor as middleware. This communication may be network based or memory based. This concept essentially allows the user to define such channels which then get forwarded to the hypervisor configuration. For the purpose of this work, the only supported setting is based on the Inter-Virtual Machine Shared Memory (IVSHMEM) protocol, which is memory based (Ren et al. 2016).

Below, the properties for this concept are listed.

```
concept Channel extends BaseConcept implements INamedConcept

properties:
size     : string
protocol : string
```

```
6
7 children:
8 connect     : Connect[1]
9 permission  : Permissions[1..n]
```

Listing 4.39: Channel Concept Structure

The size property, states the size of the shared memory region, it requires either a suffix of MB or KB, validated by a constraint component. The protocol states how this communication will occur. At the moment, only IVSHMEM is accepted as a valid protocol.

The connect property alludes to the *Connect* subconcept, which essentially defines which partition connects to a different partition. By doing so, we control how each partition is configured and we take into account the permission concept when performing the configuration generation, i.e., partition P1 may only read from partition P2, while P2 may read and write to P1.

The editor for this concept, as well as the *Connect* subconcept is shown below.

```
1 #Editor for Channel Concept
2
3 [- channel { name } with{ -]
4 [- % connect % -]
5 [- size { size } ; -]
6 [-
7   (- % permission % -)
8 -]
9 [- protocol { protocol } ; -]
10
11
12 #Editor for Connect subconcept
13
14 [- connect { from } and { to } ; -]
```

Listing 4.40: Channel Editor

As stated, the size needs to contain either a MB or a KB suffix, validated by the following constraint:

```
1 #Size suffix validation
2
3 property {size}
4   is valid (propertyValue, node)->boolean {
5     return propertyValue.substring(propertyValue.length() - 2,
    propertyValue.length()).contains("MB") || propertyValue.substring(
    propertyValue.length() - 2, propertyValue.length()).contains("KB");
6   }
```

Listing 4.41: Channel size constraint

This distinction is needed in order to perform conversions into hexadecimal bytes, which is the format needed by the hypervisor configuration file. This process will be listed on subsection 4.1.15 under the explanation of the *Behavior* component for the *Hypervisor* concept.

As all the child concepts from the *View* Concept were explained, the analysis moves on to three of the four pillars of the DSL, namely, the *Platform, Hypervisor and Application* concepts. The concept missing above is the *Install* Concept, that was analyzed in subsection

4.1.11, since it supersedes the *Operating System* Concept, because the latter has no other properties or behavioral components, at this point in time.

4.1.14 Platform Concept

Table 4.14: Platform Concept Properties

Concept Name	Concept Description	Parent
Platform	Queries nodes related to the Platform View, generates target language from their values, and TextGen components	Views Interface

The *Platform* Concept is a target for extension for a given View. The purpose of this concept is to gather information regarding all nodes that refer to platform configuration and transform the obtained values into relevant artifacts. With that being said, this concept possesses no relevant *Structure and Editor* components because it has no properties of its own.

The *TextGen* component listed below shows the process of translating the values obtained from nodes into intermediary language, which is used to create the platform artifacts.

```
# Obtains TextGen data from Platform target as well as its properties

(node)->void {
  linkedlist<string> allowStringList = new linkedlist<string>;
  node<> refinementRef = node.parent.parent.children.findFirst({~it =>
    it.isInstanceOf(Refinement); });
  node<Refinement> ref = ((node<Refinement>) refinementRef);
  for (node<Allow> allows : ref.allowList) {
    allowStringList.add(allows.target.toString());
  }
  string coresString = "cores[";
  for (node<CoreAtom> coresSingle : ref.Cores.cores) {
    coresString += String.valueOf(coresSingle.identifier) + ",";
  }
  allowStringList.add(coresString.substring(0, coresString.length() - 1)
    + "]");

  for (node<MemoryRegion> regionNode : ref.memRegions) {
    allowStringList.add("memReg:" + regionNode.target + ":" + regionNode
   .memStart + ":" + regionNode.virtStart + ":" + regionNode.size);
  }
  for (node<IRQChipDefinition> irqNode : ref.irqchips) {
    string pinbitmap = "";
    for (node<StringConcept> s : irqNode.pinBitmap.inputs) {
      pinbitmap += s.value + ",";
    }

    allowStringList.add("addr:" + irqNode.target + ":" + irqNode.address
    + ":" + irqNode.pin_base + ":" + pinbitmap.substring(0, pinbitmap.
   length() - 1));
  }
  node.saveIntermediaryLanguageToFile(allowStringList);
  for (string configLine : node.configurePV(allowStringList)) {
    append ${configLine} ;
```

```
30      append \n ;
31    }
32  }
```

Listing 4.42: Platform TextGen component

The intermediary language generated by this concept is illustrated below.

```
1
2  wlan
3  hdmi
4  cores[0,1,2,5]
5  memReg:UART:0x100000000:0x100000000:0x1000
6  memReg:RAM:0x5000000000:0x50000000:0x4000
7  memReg:IVHSMEM   2:0x275000001:0x275000001:0x1000
8  addr:GIC:0x30000000:288:0xffffffff,0xfffffffa
```

Listing 4.43: Platform intermediate language

Next, the *Behavior* component of this concept takes the values from the intermediary language and processes them to match the format of the hypervisor configuration files.

```
1
2  #Calculates the bitmask for the available platform cores
3  #@param coreConfig the configuration of the cores
4  #@return the hexadecimal representation of the cores for use in the
       hypervisor config
5
6  private string calculateCoreBitmask(string coreConfig) {
7
8    string coreConfigMain = coreConfig.replace(this.CORE_STRING_PREFIX, ""
       ).replace("]", "");
9    string[] coreConfigSplit = coreConfigMain.split(",");
10   string bitmask = "000000";
11   char[] arrBitmask = bitmask.toCharArray();
12   BitSet set = new BitSet(6);
13   for (string coreConf : coreConfigSplit) {
14     arrBitmask[Math.abs(Integer.parseInt(coreConf))] = '1';
15   }
16   int inc = 0;
17   for (int i = (arrBitmask.length - 1); i >= 0; i--) {
18     if (arrBitmask[i] == '1') {
19       set.set(inc++, true);
20       continue;
21     }
22     set.set(inc++, false);
23   }
24   return String.format("0x%01X", new Long(set.toLongArray()[0]));
25 }
```

Listing 4.44: Platform Bitmask Calculator for Core

The next function is the main configuration function for the hypervisor configuration files, which essentially is the basis of the platform.

```
1
2
3  #Main method that configures the platform view. It generates the
       hypervisor files for the require #hypervisor
```

```
#@return the list of configs for the hypervisor
#@param config the list of strings with the configuration. They are processed in the TextGen component of #this concept

public LinkedList<string> configurePV(linkedlist<string> config) {
  try {
    Path p = FileSystems.getDefault().getPath(this.HYPERVISOR_CONFIG_FILE_PATH, this.HYPERVISOR_CONFIG_FILE_NAME);
    Path out = FileSystems.getDefault().getPath(this.HYPERVISOR_CONFIG_OUTPUT_FILE_PATH, this.HYPERVISOR_CONFIG_OUTPUT_FILE_NAME);
    List<string> list = Files.readAllLines(p);
    LinkedList<string> listConfigs = new LinkedList(list);
    LinkedList<string> finalList = new LinkedList(list);

    for (string platformCfgEntry : listConfigs) {
      for (string configSingle : config) {
        if (configSingle.contains(this.CORES_SEARCH_TERM) && platformCfgEntry.contains(this.CORES_SEARCH_TERM_IN_FILE)) {
          int indexCpus = listConfigs.indexOf(platformCfgEntry);
          string newEntry = "\t\t" + calculateCoreBitmask(configSingle) + ",";
          finalList.remove(indexCpus + this.TRAILING_LINES_CORE_CONFIG);
          finalList.add(indexCpus + this.TRAILING_LINES_CORE_CONFIG, newEntry);
          continue;
        }
        if (configSingle.contains(this.MEMORY_REG_SEARCH_TERM)) {
          string[] memConfig = configSingle.split(":");
          // if file configuration contains memory target -- (UART OR SIMILAR)
          // [1] target
          // [2] phys start
          // [3] virt start
          // [4] size
          if (platformCfgEntry.contains(memConfig[1])) {
            int indexName = listConfigs.indexOf(platformCfgEntry);
            string physConfig = "\t\t" + this.PHYS_START_PREFIX + memConfig[2] + ",";
            string virtConfig = "\t\t" + this.VIRT_START_PREFIX + memConfig[3] + ",";
            string sizeConfig = "\t\t" + this.SIZE_PREFIX + memConfig[4] + ",";
            finalList.remove(indexName + this.TRAILING_LINES_PHYS_CONFIG);
            finalList.add(indexName + this.TRAILING_LINES_PHYS_CONFIG, physConfig);
            finalList.remove(indexName + this.TRAILING_LINES_VIRT_CONFIG);
            finalList.add(indexName + this.TRAILING_LINES_VIRT_CONFIG, virtConfig);
            finalList.remove(indexName + this.TRAILING_LINES_SIZE_CONFIG);
            finalList.add(indexName + this.TRAILING_LINES_SIZE_CONFIG, sizeConfig);
            continue;
          }
        }
        if (configSingle.contains(this.ADDR_SEARCH_TERM)) {
```

```
46            string[] addrConfig = configSingle.split(":");
47            if (platformCfgEntry.contains(addrConfig[1])) {
48              int indexName = listConfigs.indexOf(platformCfgEntry);
49              string addressConfig = "\t\t" + this.ADDRESS_PREFIX +
    addrConfig[2] + ",";
50              string baseConfig = "\t\t" + this.PINBASE_PREFIX +
    addrConfig[3] + ",";
51              string bitmapConfig1 = "\t\t" + this.PINBITMAP_PREFIX;
52              string[] bitmapConfig2Split = addrConfig[4].split(",");
53
54              finalList.remove(indexName + this.TRAILING_LINES_ADDR_CONFIG
    );
55              finalList.add(indexName + this.TRAILING_LINES_ADDR_CONFIG,
    addressConfig);
56              finalList.remove(indexName + this.
    TRAILING_LINES_PINBASE_CONFIG);
57              finalList.add(indexName + this.TRAILING_LINES_PINBASE_CONFIG
    , baseConfig);
58              finalList.remove(indexName + this.
    TRAILING_LINES_PINBITMAP_CONFIG_1);
59              finalList.add(indexName + this.
    TRAILING_LINES_PINBITMAP_CONFIG_1, bitmapConfig1);
60              for (int i = 0; i < bitmapConfig2Split.length; i++) {
61                finalList.remove(indexName + this.
    TRAILING_LINES_PINBITMAP_CONFIG_2 + i);
62                finalList.add(indexName + this.
    TRAILING_LINES_PINBITMAP_CONFIG_2 + i, i + 1 < bitmapConfig2Split.
    length ? "\t\t\t" + bitmapConfig2Split[i] + "," : "\t\t\t" +
    bitmapConfig2Split[i]);
63              }
64              continue;
65     Files.write(out, finalList);
66     return finalList;
67   } catch (Exception ex) {
68     return new LinkedList<string>();
69   }
70 }
```

Listing 4.45: Platform configuration function

Most of the configuration is built using values queried from the Constants interface, so it is easily adaptable to new configuration formats, i.e., different versions of the *Jailhouse* hypervisor. The results from generating text from this concept are described on the Results chapter on Chapter 5.

4.1.15 Hypervisor Concept

Table 4.15: Hypervisor Concept Properties

Concept Name	Concept Description	Parent
Hypervisor	Queries nodes related to the Hypervisor View generates target language from their values and TextGen components	Views Interface

The *Hypervisor* Concept is a target for extension for a given View. The purpose of this concept is to gather information regarding all nodes that refer to hypervisor partition configuration and transform the obtained values into target language artifacts. This concept possesses no relevant *Structure and Editor* components because it has no properties of its own.

Below, the *TextGen* component for this component is illustrated.

```
# Obtains TextGen data from Hypervisor target as well as its properties

(node)->void {
  node<> viewRef = node.parent.parent;
  node<View> vRef = ((node<View>) viewRef);
  linkedlist<node<Partition>> partitionPropsList = new linkedlist<node<
    Partition>>;
  for (node<Partition> partition : vRef.Partition) {
    partitionPropsList.add(partition);
  }
  linkedlist<node<Channel>> channelPropsList = new linkedlist<node<
    Channel>>;
  for (node<Channel> channel : vRef.Channel) {
    channelPropsList.add(channel);
  }
  for (string configLine : node.configureHV(partitionPropsList,
    channelPropsList)) {
    append ${configLine} ;
    append \n ;
  }
}
```

Listing 4.46: Hypervisor TextGen

It collects data from *Partitions and Channel* Concepts and then uses them to create the output artifacts, using behavioral functions. The intermediate language extracted by this component is illustrated below.

```


cores[0,1]
name:LinuxROS
memreg:UART:0x100000000:0x100000000:0x2000
memreg:RAM:0x5000000000:0x50000000:0x1000
addr:GIC:0x30000000:288:0 << (332 - 320)
device:IVSHMEM:3
channel:LinuxROS:LinuxROS2:10MB:LinuxROS:read,write:LinuxROS2:read:
    IVSHMEM 1
channel:LinuxROS:LinuxROS2:10KB:LinuxROS:read:LinuxROS2:read,write:
    IVSHMEM 2
cores[2,3]
name:LinuxROS2
memreg:RAM:0x5000000000:0x50000000:0x3000
channel:LinuxROS:LinuxROS2:10MB:LinuxROS:read,write:LinuxROS2:read:
    IVSHMEM 1
channel:LinuxROS:LinuxROS2:10KB:LinuxROS:read:LinuxROS2:read,write:
    IVSHMEM 2
```

Listing 4.47: Hypervisor Intemediary Language

This data is then sent to the behavioral functions listed below.

```
1
2 # Main method that configures the Hypervisor.
3 # This method is called from the TextGen view of this component.
4 # Partition and channel nodes are prepared within the same view.
5 #@param configChannel The list of communication channel nodes
6 #@param configParts The list of partition nodes
7 #@return A list of strings containing the already parsed hypervisor
      configuration file
8
9 public LinkedList<string> configureHV(linkedlist<node<Partition>>
      configParts, linkedlist<node<Channel>> configChannel) {
10   try {
11     Path p = FileSystems.getDefault().getPath(this.CELL_CONFIG_FILE_PATH
      , this.CELL_CONFIG_FILE_NAME);
12     List<string> list = Files.readAllLines(p);
13     LinkedList<string> configsList = new LinkedList(list);
14     LinkedList<string> partList = new LinkedList();
15
16     LinkedList<string> configStrings = new LinkedList<string>();
17     for (node<Partition> part : configParts) {
18       string coresString = this.CORE_STRING_PREFIX;
19       for (node<CoreAtom> coresSingle : part.Cores.cores) {
20         coresString += String.valueOf(coresSingle.identifier) + ",";
21       }
22       configStrings.add(coresString.substring(0, coresString.length() -
      1) + "]");
23       configStrings.add(this.CONFIG_STRINGS_NAME_PREFIX + part.name);
24       for (node<MemoryRegion> memReg : part.MemoryRegion) {
25         configStrings.add(this.CONFIG_STRINGS_MEMORY_PREFIX + memReg.
      target + ":" + memReg.memStart + ":" + memReg.virtStart + ":" +
      memReg.size);
26       }
27       for (node<IRQChipDefinition> irqNode : part.irqChips) {
28         configStrings.add(this.CONFIG_STRINGS_ADDR_PREFIX + irqNode.
      target + ":" + irqNode.address + ":" + irqNode.pin_base + ":" +
      getStringFromArray(irqNode.pinBitmap.inputs, ","));
29       }
30       for (node<PciDevices> pciDevices : part.pciDevices) {
31         configStrings.add(this.CONFIG_STRINGS_DEVICE_PREFIX + pciDevices
      .type + ":" + pciDevices.region);
32       }
33       for (node<Channel> channel : configChannel) {
34         string channelConfg = this.CONFIG_STRINGS_CHANNEL_PREFIX +
      channel.connect.from + ":" + channel.connect.to + ":" + channel.size
      + ":";
35         for (node<Permissions> permission : channel.permission) {
36           channelConfg += permission.target + ":";
37           channelConfg += getStringFromArray(permission.permission, ",")
       + ":";
38         }
39         channelConfg += channel.protocol;
40         configStrings.add(channelConfg);
41       }
42       string fileName = this.CELL_CONFIG_OUTPUT_FILE_NAME + part.name +
      ".c";
43       Path out = FileSystems.getDefault().getPath(this.
      CELL_CONFIG_FILE_PATH, fileName);
44       LinkedList<string> partSingle = processPartition(configStrings,
      configsList);
```

```
45          partList.addAll(partSingle);
46          Files.write(out, partSingle);
47        }
48        saveIntermediaryLanguageToFile(configStrings);
49        return partList;
50      } catch (Exception ex) {
51        LinkedList<string> finalList = new LinkedList();
52        finalList.add(ex.toString());
53        return finalList;
54      }
55    }
```

Listing 4.48: Hypervisor Main Behavior

The function above processes all nodes retrieved by the *TextGen* Component. It calls the *ProcessPartition* function which extracts and parses all data from a given partition. This function is listed within Appendix E due to its size.

This function properly configures each partition with all the required properties, such as channels, memory regions, permissions, etc.

The next behavioral function was mentioned on the explanation of the *Channel* Concept, and it refers to the translation of the size property of the communication channel into a format accepted by the hypervisor configuration file.

```
1
2  # Helper method to convert a decimal size (i.e., 10MB) into an
       hexadecimal byte representation
3
4  #@param actualSize The decimal size
5  #@param unit The memory unit (KB,MB)
6  #@return The hexadecimal representation of the size in bytes
7
8  private string convertToHexByte(string actualSize, string unit) {
9    long size = Long.parseLong(actualSize);
10   if (unit.equalsIgnoreCase("MB")) {
11     long multiplier = 1024 * 1024;
12     size *= multiplier;
13   }
14   if (unit.equalsIgnoreCase("KB")) {
15     size *= 1024;
16   }
17   return Long.toHexString(size);
18 }
```

Listing 4.49: Hypervisor Convert to hexadecimal byte

The function below, helps translating the selected active CPU cores from the *Core* Concept to a format understood by the hypervisor configuration file, since it requires a bitmask[6].

```
1
2  # Private method that calculates the core value for the hypervisor
       configuration file.
3  #@param coreConfig a string containing the selected cores in the
       following format: cores[x,y,z]
4  #@return The hexadecimal representation of the selected cores
5
```

[6] A Bitmask is an aglomerate of boolean flags

```
6  private string calculateCoreBitmask(string coreConfig) {
7    string coreConfigMain = coreConfig.replace(this.CORE_STRING_PREFIX, ""
       ).replace("]", "");
8    string[] coreConfigSplit = coreConfigMain.split(",");
9    string bitmask = "000000";
10   char[] arrBitmask = bitmask.toCharArray();
11   BitSet set = new BitSet(6);
12   for (string coreConf : coreConfigSplit) {
13     arrBitmask[Math.abs(Integer.parseInt(coreConf))] = '1';
14   }
15   int inc = 0;
16   for (int i = (arrBitmask.length - 1); i >= 0; i--) {
17     if (arrBitmask[i] == '1') {
18       set.set(inc++, true);
19       continue;
20     }
21     set.set(inc++, false);
22   }
23   return String.format("0x%01X", new Long(set.toLongArray()[0]));
24 }
```

Listing 4.50: Hypervisor Bitmask Calculator from Core

The results obtained from the text generation process from this concept will be presented on Chapter 5. To end this chapter, the last concept under analysis will be the *Application* Concept.

4.1.16 Application Concept

Table 4.16: Application Concept Properties

Concept Name	Concept Description	Parent
Application	Queries nodes related to the Application View generates target language from their values and TextGen components	Views Interface

The *Application* Concept follows the same logic as the two previous concepts, with its purpose being the collection of properties related to the *Application View*. Similarly, this concept possesses no properties of its own, therefore, it doesn't employ *Structure or Editor* components. With that being said, the property collection process is implemented within the *TextGen* component of this concept.

```
1  # Obtains TextGen data from Application target as well as its properties
2
3  (node)->void {
4    linkedlist<string> cfgStringList = new linkedlist<string>;
5    node<> refinementRef = node.parent.parent.children.findFirst({~it =>
       it.isInstanceOf(Refinement); });
6    node<Refinement> ref = ((node<Refinement>) refinementRef);
7    for (node<Topic> topics : ref.topics) {
8      cfgStringList.add(topics.path + "," + topics.type);
9    }
10   for (string configLine : node.configureTopics(cfgStringList)) {
11     append ${configLine} ;
```

```
12      append \n ;
13    }
14  }
```

Listing 4.51: Application TextGen

After gathering the relevant data, it uses the *Behavior Component* and its functions in order to process the data into target language/artifacts.

```
1
2  #Configures ROS Topics into target C language code
3  #@param TopicData: The list of topic properties
4
5  public LinkedList<string> configureTopics(linkedlist<string> topicData)
       {
6    try {
7      Path pub = FileSystems.getDefault().getPath(this.
       ROS_TOPIC_OUTPUT_FILE_PATH, this.ROSTOPIC_IN);
8      Path sub = FileSystems.getDefault().getPath(this.
       ROS_TOPIC_OUTPUT_FILE_PATH, this.ROS_SUB_IN);
9      Path out = FileSystems.getDefault().getPath(this.
       ROS_TOPIC_OUTPUT_FILE_PATH, this.ROS_TOPIC_OUTPUT_FILE_NAME);
10     Path sOut = FileSystems.getDefault().getPath(this.
       ROS_TOPIC_SUB_OUTPUT_FILE_PATH, this.ROS_TOPIC_SUB_OUTPUT_FILE_NAME);
11     Path pAux = FileSystems.getDefault().getPath(this.DOCKERFILE_OUT, "
       dockerfile.txt");
12     Path outAux = FileSystems.getDefault().getPath(this.DOCKERFILE_OUT,
       this.DOCKERFILE_NAME);
13     List<string> list = Files.readAllLines(pub);
14     LinkedList<string> listConfigs = new LinkedList(list);
15     LinkedList<string> finalList = new LinkedList(list);
16     for (String data : topicData) {
17       string path = data.split(",")[0];
18       string type = data.split(",")[1];
19       for (String configLine : listConfigs) {
20         if (configLine.contains("ros::Publisher")) {
21           string typeLocal = "  ros::Publisher chatter_pub = n.advertise
       <std_msgs::" + type + ">" + "(\"" + path + "\", 1000);";
22           int index = finalList.indexOf(configLine);
23           finalList.remove(index);
24           finalList.add(index, typeLocal);
25     Files.write(out, finalList);
26     List<string> listDockerFile = Files.readAllLines(pAux);
27     Files.write(outAux, listDockerFile);
28     list.clear();
29     list = Files.readAllLines(sub);
30     LinkedList<string> listConfigsSub = new LinkedList(list);
31     LinkedList<string> finalListSub = new LinkedList(list);
32     for (String data : topicData) {
33       string path = data.split(",")[0];
34       string type = data.split(",")[1];
35       for (String configLine : listConfigsSub) {
36         if (configLine.contains("Callback")) {
37           string typeLocal;
38           switch (type) {
39             //add more types as needed
40             case "float" :
41               typeLocal = "void chatterCallback(const std_msgs::Float::
       ConstPtr& msg)";
42               break;
```

```
43              default :
44                typeLocal = "void chatterCallback(const std_msgs::String::
    ConstPtr& msg)";
45            }
46            int index = finalListSub.indexOf(configLine);
47            finalListSub.remove(index);
48            finalListSub.add(index, typeLocal);
49      Files.write(sOut, finalListSub);
50      finalList.addAll(finalListSub);
51      return finalList;
52    } catch (Exception ex) {
53      return new LinkedList<string>();
54    }
55  }
```

Listing 4.52: Application main behavioral function

The results produced by the text generation of this concept will be presented in Chapter 5.

Chapter 5

Experiments and Results

In this chapter, the experiments conducted and the results obtained from the utilization of the DSL described within the present work, will be described since tests were conducted through the development process, in order to better adapt that same process and shape the end result.

As stated previously, the objective of the DSL is to translate the language nodes provided by it, into target language or artifacts. With that being said, these artifacts vary depending on their target. Some artifacts are scripts meant to be run under a Unix based OS, while others are C or C++ language files. For each one of the artifacts with need of output, there is a base artifact needed for input, excluding scripts, which are built standalone. This is because the text generation process parses these input files and performs changes accordingly. While the DSL can adapt most of the search strings or structure simply by altering the *Constants Interface*, these files are meant to be stable, as the development process was shaped by these files.

5.1 Produced Artifacts and used Hardware

The experiments conducted within the present work, focused on trying to set up a given system using the different *Views* the DSL provides. These systems do not need to be connected to one another, i.e., the artifacts produced by each *View* would be modular. For example, we are able to configure a kernel configuration file for personal use, or, set up an hypervisor configuration file. During the presentation of this work, these concepts were presented as dependent on each other to some degree, but this is only the case for validation purposes, since the production of artifacts is not restricted by the constraints specified for each *View*. While one use case may consist of building a system built by the artifacts produced by the *Platform View, Hypervisor View, OS View and Application View*, other use cases may include any combination of these views.

With that being said, experiments conducted, specifically consisted of:

1. Setting up a Custom Kernel Configuration
2. Setting up a Custom Hypervisor Configuration (Jailhouse)
3. Compiling the Jailhouse Hypervisor[1]
4. Deploying scripts (to execute under Ubuntu Desktop 18.02) to Download OS images

[1] The Jailhouse hypervisor was not deployed due to lack of compatible hardware

5. Deploying scripts (to execute under the configured Unix based OS) to set up packages in the fresh OS
6. Building the Custom OS images using artifacts from 1), 4) and 5)
7. Deploying Custom Publish/Subscribe ROS applications
8. Deploying scripts to configure and deploy Docker Images to execute ROS and the applications
9. Executing scripts deployed on 5) and 9)

Below is the list of the *hardware* utilized to perform the experiments:

1. Windows 10 Desktop Core I7 4910HQ 1.9Ghz CPU to operate DSL and generate artifacts (approx. 2 second generation time)
2. Ubuntu 18.02 Desktop Core-2-Duo 2.5Ghz CPU to execute scripts and necessary compilations (kernel, Jailhouse, generate disk image, approx. 25-30 minutes process)
3. BananaPi M1 Cortex A7 Dual-Core 1.0Ghz to install the custom made OS and execute scripts (package downloads, executing Docker and ROS Publish/Subscribe applications (approx. 5 mins process)

In the next section, the artifacts generated from each *View* described using the DSL are illustrated.

5.2 View Artifacts

5.2.1 Platform View Artifacts

Given the following *Platform View* specification:

```
1  view DemoPV is Views.Platform {
2
3  refines JetsonTX2 ;
4      use cores [ 0 , 1 , 2 , 5 , ]
5      allow wlan ;
6      allow hdmi ;
7      region 0x100000000 .. 0x100000000 : 0x1000 : UART
8      region 0x5000000000 .. 0x50000000 : 0x4000 : RAM
9      region 0x275000001 .. 0x275000001 : 0x1000 : IVHSMEM  2
10     irqchip 0x30000000 .. 288 : 0xffffffff 0xfffffffa : GIC
11
12 }
```

The generation process produces the hypervisor configuration file located in Appendix F as well as a kernel configuration file located in Appendix G. Note that both of these only contain the relevant changes made to the file, due to the size of the real output.

5.2.2 Hypervisor View Artifacts

Given the following *Hypervisor View* specification:

```
view DemoHV is Views.Hypervisor {
  refines Jailhouse;

  partition LinuxROS with {
    OS linux ;
    region 0x100000000 .. 0x100000000 : 0x2000 : UART
      region 0x5000000000 .. 0x50000000 : 0x1000 : RAM
    irqchip 0x30000000 .. 282 : 0 << (332 - 320) : GIC
    device IVSHMEM , region 3
    use cores [ 0 , 1 , ]
  }
}
```

The artifacts generated from this view specification is Jailhouse cell configuration. Changes to the configuration file located in Appendix H.

5.2.3 Operating System View Artifacts

Having into account the following Operating System View specification:

```
view DemoOS is Views.OS {

    Installation configuration {
    Target OS: Ubuntu Server
    Kernel version: 4.4
    Board: BananaPi M1
    Pre-Installed image: false
      Package     Y/N
    1 ros-kinetic Y
    2 jailhouse   Y
    3 docker      Y
    }
}
```

Running the text generation tool on this view returns the artifact illustrated under Appendix B.

5.2.4 Application View Artifacts

Given the following *Application View* specification:

```

view DemoApplication is Views.Application {

refines ROS.Topics ;
    Topic T1 with {
        path: /ros/topics
        type: float
    }

    node ROS_inNode with {
```

```
11          body: /ros/topics
12          topics: {
13              mode write to T1 ;
14          }
15      }
16  }
```

The artifacts generated from this view are found in Appendix A.

The following illustration depicts the execution of the artifacts obtained from the current view:

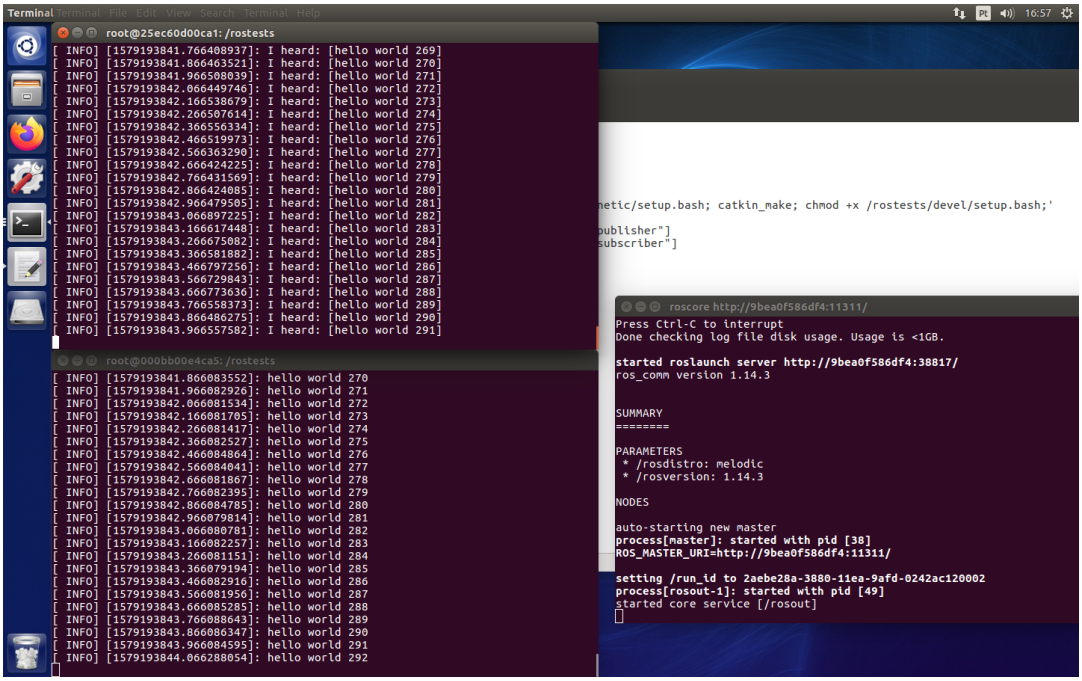

Figure 5.1: ROS Application execution

Chapter 6

Conclusions and other contributions

The work produced under the context of this Master's thesis was extremely fulfilling. As I've worked first hand with technologies used within the domain of Automotive Systems, developing a language that simplifies the process of creating and deploying a system built under a tool that can attenuate the complexities of other existing tools, or, condense services that are part of multiple tools in one place, was very challenging. As the complexity of the technologies connected with the automotive domain, grows, a need for easy to use solutions for smaller problems, might prove a valuable asset for companies and developers alike, lowering costs in technology-specific trainings and Software licences.

The objectives stated in the present work were achieved. A Domain Specific Language that allows the description of a system from the viewpoints of different parts of the system was created. The DSL provides control over user input when needed, but is also flexible in cases where such restrictions may not apply.

This DSL facilitates the definition and deployment of such systems, albeit at a level unfit for professional use, in my own opinion and at this point in time, as the subject and complexity desired to operate at such levels takes time and effort to achieve.

With that being said, the created *DSL* has much room for growth, in terms of supported platforms, supported Software packages with stable repositories, customized top level applications, monitoring, security, formal verification, with those being valid points for future work on the subject.

For contributions, besides the obvious contribute of the Domain Specific Language, a scientific article titled "A Domain Specific Language for Automotive Systems Integration" was published and presented in the 45th Annual Conference of the IEEE Industrial Electronics Society (IECON 2019), with the response being extremely positive. This states that the present work was validated by the community in what concerns the value it can bring to the Automotive scene.

Bibliography

Baryshnikov, Maxim (2016). "Jailhouse hypervisor". B.S. thesis. České vysoké učenıe technické v Praze. Vypočetnıe a informačnıe centrum.

Bryans, Jeremy et al. (2014). "SysML contracts for systems of systems". In: *2014 9th International Conference on System of Systems Engineering (SOSE)*. IEEE, pp. 73–78.

Debruyne, Vincent, Françoise Simonot-Lion, and Yvon Trinquet (2004). "EAST-ADL—An architecture description language". In: *IFIP World Computer Congress, TC 2*. Springer, pp. 181–195.

Eysholdt, Moritz and Heiko Behrens (2010). "Xtext: implement your language faster than the quick and dirty way". In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, pp. 307–309.

Koen, Peter A et al. (2002). *Fuzzy front end: effective methods, tools, and techniques*.

Koubâa, Anis et al. (2017). *Robot Operating System (ROS)*. Vol. 1. Springer.

Land, Rikard and Ivica Crnkovic (2003). "Software Systems Integration and Architectural Analysis - A Case Study". In: pp. 338–347. isbn: 0-7695-1905-9. doi: 10.1109/ICSM.2003.1235441.

Miles, Lawrence D (2015). *Techniques of value analysis and engineering*. Miles Value Foundation.

MPS Description. https://www.jetbrains.com/mps/concepts/. Accessed: 2021-02-15.

Ren, Yi et al. (2016). "Shared-memory optimizations for inter-virtual-machine communication". In: *ACM Computing Surveys (CSUR)* 48.4, pp. 1–42.

Sangiovanni-Vincentelli, Alberto, Werner Damm, and Roberto Passerone (2012). "Taming Dr. Frankenstein: Contract-based design for cyber-physical systems". In: *European journal of control* 18.3, pp. 217–238.

Shah, Aditya A, Dirk Schaefer, and Christiaan JJ Paredis (2009). "Enabling multi-view modeling with SysML profiles and model transformations". In: *International Conference on Product Lifecycle Management*. Citeseer, pp. 527–538.

Voelter, Markus et al. (2012). "mbeddr: an extensible C-based programming language and IDE for embedded systems". In: *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. ACM, pp. 121–140.

Appendix A

Robot Operating System PUB/SUB applications

```
#### START_REFINEMENT_ROS ####

#### Publisher ####

#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "publisher");
  ros::NodeHandle n;
  ros::Publisher chatter_pub = n.advertise<std_msgs::float>("/ros/topics
    ", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}

#### Subscriber ####

#include "ros/ros.h"
#include "std_msgs/String.h"
void chatterCallback(const std_msgs::Float::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}
int main(int argc, char **argv)
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;
```

```
43 void chatterCallback(const std_msgs::Float::ConstPtr& msg)
44   ros::spin();
45   return 0;
46 }
47
48 #### END_REFINEMENT_ROS ####
```

Appendix B

Linux Kernel Download and Install

```
1
2 #### START_INSTALL_INFORMATION ####
3
4 #!/bin/bash
5 SECONDS=0
6
7 #configure linux kernel
8
9 echo "Downloading linux kernel version 4.4"
10 wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.4.191.tar.xz
11 tar -xvf linux-4.4.191.tar.xz
12 cp bananapi_defconfig linux-4.4.191/arch/arm/configs
13 cd linux-4.4.191
14 #can copy kconfig generated from DSL
15 echo "Compiling kernel..."
16 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -j4  bananapi_defconfig
17 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -j4  uImage LOADADDR=0
       x40008000 dtbs modules
18 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -j4  modules_install
       INSTALL_MOD_PATH=/home/renato/Downloads/BPI/FS
19 cp arch/arm/boot/uImage ../
20 chmod 777 ../uImage
21 cd ..
22 echo "Kernel took $SECONDS seconds to compile. Resetting timer..."
23 SECONDS=0
24
25 #configure disk image
26
27 mkdir copy
28 cp ../BPI/ubuntusrv.imgubuntusrv.img ./copy
29 mv ./copy/ubuntusrv.img ./testbpi.img
30 rm -r copy
31 echo "Setting up loopback device"
32 sudo losetup -P /dev/loop53 testbpi.img
33 mkdir tmpfs
34 mkdir tmpfs/partition1
35 mkdir tmpfs/partition2
36 echo "Mounting fake partitions"
37 sudo mount -t vfat /dev/loop53p1 tmpfs/partition1
38 sudo mount -t ext4 /dev/loop53p2 tmpfs/partition2
39 sync
40 sudo rm -r tmpfs/partition1/
41 sudo rm -r tmpfs/partition2/
42 echo "Copying filesystem"
43 sudo cp -r ../BPI/BPI-ROOT/ tmpfs/partition1
44 sudo cp -r ../BPI/BPI-ROOT2/ tmpfs/partition2
```

```
mkdir tmpfs/partition2/scripts
sudo mkdir tmpfs/partition2/scripts/rosapp
sudo mkdir tmpfs/partition2/scripts/rosapp/src
sudo mkdir tmpfs/partition2/scripts/rosapp/src/app
sync
sudo rm tmpfs/partition1/bananapi/bpi-all/linux4/extlinux/zImage
echo "Copying kernel image"
sudo cp zImage tmpfs/partition1/bananapi/bpi-all/linux4/extlinux
sudo umount /dev/loop53p1
sudo umount /dev/loop53p2
sudo losetup -d /dev/loop53
echo "Took $SECONDS seconds to create image file"

#ROS installation

echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
    /etc/apt/sources.list.d/ros-latest.list
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-
    key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
sudo apt-get update
sudo apt-get install ros-kinetic-desktop-full
sudo rosdep init
rosdep update
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc>
source ~/.bashrc

#Jailhouse installation

git clone https://github.com/siemens/jailhouse.git
make install

#Docker installation

sudo apt-get install apt-transport-https ca-certificates curl  gnupg-
    agent  software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
    add -
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository \n   "deb [arch=amd64] https://download.docker.
    com/linux/ubuntu \n   $(lsb_release -cs) \n   stable"
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io

#ROS Image for Docker

sudo docker pull ros
sudo docker network create rosnet
/bin/bash -c 'docker run -it --net rosnet --name master ros roscore'

#Prepare ROS environment

sudo cp publisher-new.cpp tmpfs/partition2/scripts/rosapp/src/app
sudo cp subscriber-new.cpp tmpfs/partition2/scripts/rosapp/src/app

#### END_INSTALL_INFORMATION ####
```

Appendix C

Install concept Editor component

```
<default> editor for concept Install
  node cell layout:
    [/
      Installation configuration {
      [- Target OS: { OS } -]
      [- Kernel version: { kernelVersion } -]
      [- Board: { boardTarget } -]
      [- Pre-Installed image: { preinstalled } -]
      [-
        table {
          vertical c<{
            "Package"
            "Y/N"
          }> {
            horizontal r<query {
              getHeaders Test (node, editorContext)->join(string | EditorCell | node<> | Iterable) {
                return node.targets.select({~it => Integer.toString(it.index + 1); });
              }
              insert new header (node, index)->void {
                node.targets.add new(<default>);
              }
              on delete: (node, index)->void {
                if (node.targets.size > 1) {
                  node.targets.removeAt(index);
                }
              }
            }> {
              query {
                shared variables << ... >>
                initialize       <no sharedInit>
                column count     (node)->int {
                                   return 2;
                                 }
                row count        (node)->int {
                                   node.targets.size;
                                 }
                cell             (node, columnIndex, rowIndex, editorContext)->join(node<> | string | EditorCell |
                                     {
                                   node<InstallTableLines> singleTarget = node.targets[rowIndex];

                                   switch (columnIndex) {
                                     case 0 :
                                       return editorContext.createCell(singleTarget, -> { descripton });
                                     case 1 :
                                       return editorContext.createCell(singleTarget, -> { value });
                                     default :
                                       throw new RuntimeException("invalid index");
                                   }
                                 } as vertical list
```

Figure C.1: Install concept Editor component

Appendix D

Install concept behavioral function

```
1
2  public LinkedList<string> configureInstall(linkedlist<string>
       configsList) {
3    saveIntermediaryLanguageToFile(configsList);
4    LinkedList<string> ret = new LinkedList();
5    LinkedList<string> retScripts = new LinkedList();
6    Path out = FileSystems.getDefault().getPath(this.
       INSTALL_CONFIG_FILE_PATH, this.INSTALL_CONFIG_FILE_NAME);
7    Path outScripts = FileSystems.getDefault().getPath(this.
       INSTALL_CONFIG_FILE_PATH, this.SCRIPTS_INJECTION_FILE_NAME);
8    ret.add("#!/bin/bash");
9    for (string configLine : configsList) {
10     if (configLine.contains("target:")) {
11       // if easy-deploy is enabled  (downloads  official image)
12       if (configLine.contains("BananaPi") && configLine.contains("true")
       ) {
13         ret.add("wget http://cdimage.ubuntu.com/releases/bionic/release/
       ubuntu-18.04.3-preinstalled-server-arm64+raspi3.img.xz?_ga
       =2.261429517.1054093792.1567174104-1482131256.1553622037");
14         ret.add("xzcat ubuntu-18.04.3-preinstalled-server-arm64+raspi3.
       img.xz | sudo dd of=mmcblk0 bs=32M");
15         ret.add("sync");
16         continue;
17       }
18       if (configLine.contains("BananaPi")) {
19         ret.add("SECONDS=0\n");
20         ret.add("#configure linux kernel");
21         string kVersion = configsList.findFirst({~it => it.contains("
       kernel"); }).split(":")[1];
22         ret.add("echo \"Downloading linux kernel version " + kVersion +
       "\"");
23         linkedlist<string> kDownloadLinks = new linkedlist<string>;
24         string[] links = this.KERNEL_VERSION_DOWNLOADS.split(this.
       KERNEL_VERSION_DOWNLOADS_SEPARATOR);
25         for (string link : links) {
26           warn link, <no project>;
27           kDownloadLinks.add(link);
28         }
29
30         string downloadLink = kDownloadLinks.findFirst({~it => it.
       contains(kVersion); }).split("_")[1];
31         ret.add("wget " + downloadLink);
32         warn downloadLink + "", <no project>;
33
34         linkedlist<string> splitDownloadLink = new linkedlist<string>;
35         for (string node : downloadLink.split("/")) {
```

```
36            splitDownloadLink.add(node);
37          }
38
39          string fileName = splitDownloadLink.get(splitDownloadLink.size -
     1);
40
41          ret.add("tar -xvf " + fileName);
42          string folderName = fileName.split(".tar")[0];
43          ret.add("cp " + this.BANANAPI_KCONFIG + " " + folderName + "/
    arch/arm/configs");
44          ret.add("cd " + folderName);
45          ret.add("#can copy kconfig generated from DSL");
46          ret.add("echo \"Compiling kernel...\"");
47          ret.add(this.KERNEL_COMPILE_TOOLCHAIN + " " + this.
    BANANAPI_KCONFIG);
48          ret.add(this.KERNEL_COMPILE_TOOLCHAIN + " ulmage LOADADDR=0
    x40008000 dtbs modules");
49          ret.add(this.KERNEL_COMPILE_TOOLCHAIN + " modules_install
    INSTALL_MOD_PATH=" + this.INSTALL_MOD_PATH);
50          ret.add("cp arch/arm/boot/ulmage ../");
51          ret.add("chmod 777 ../ulmage");
52          ret.add("cd ..");
53          ret.add("echo \"Kernel took $SECONDS seconds to compile.
    Resetting timer...\"");
54          ret.add("SECONDS=0\n");
55          ret.add("#configure disk image");
56          ret.add("mkdir copy");
57          // may substitute for wget -- online images for bpi
58          ret.add("cp ../BPI/ubuntusrv.img" + this.IMAGE_BASE + " ./copy")
    ;
59          ret.add("mv ./copy/" + this.IMAGE_BASE + " ./" + this.IMAGE_OUT)
    ;
60          ret.add("rm -r copy");
61          ret.add("echo \"Setting up loopback device\"");
62          ret.add("sudo losetup -P /dev/loop53 " + this.IMAGE_OUT);
63          ret.add("mkdir tmpfs");
64          ret.add("mkdir tmpfs/partition1");
65          ret.add("mkdir tmpfs/partition2");
66          ret.add("echo \"Mounting fake partitions\"");
67          ret.add("sudo mount -t vfat /dev/loop53p1 tmpfs/partition1");
68          ret.add("sudo mount -t ext4 /dev/loop53p2 tmpfs/partition2");
69          ret.add("sync ");
70          ret.add("sudo rm -r tmpfs/partition1/*");
71          ret.add("sudo rm -r tmpfs/partition2/*");
72          ret.add("echo \"Copying filesystem\"");
73          ret.add("sudo cp -r ../BPI/BPI-ROOT/* tmpfs/partition1");
74          ret.add("sudo cp -r ../BPI/BPI-ROOT2/* tmpfs/partition2");
75          ret.add("mkdir tmpfs/partition2/scripts");
76          ret.add("sudo mkdir tmpfs/partition2/scripts/rosapp");
77          ret.add("sudo mkdir tmpfs/partition2/scripts/rosapp/src");
78          ret.add("sudo mkdir tmpfs/partition2/scripts/rosapp/src/app");
79          ret.add("sync");
80          ret.add("sudo rm tmpfs/partition1/bananapi/bpi-all/linux4/
    extlinux/zlmage");
81          ret.add("echo \"Copying kernel image\"");
82          ret.add("sudo cp zlmage tmpfs/partition1/bananapi/bpi-all/linux4
    /extlinux");
83          ret.add("sudo umount /dev/loop53p1");
84          ret.add("sudo umount /dev/loop53p2");
```

```
85          ret.add("sudo losetup -d /dev/loop53");
86          ret.add("echo \"Took $SECONDS seconds to create image file\"");
87          continue;
88        }
89      }
90      if (configLine.contains("line:")) {
91        if (configLine.contains("ros-")) {
92          retScripts.add("#ROS installation");
93          retScripts.add("echo \"deb http://packages.ros.org/ros/ubuntu $(
    lsb_release -sc) main\" > /etc/apt/sources.list.d/ros-latest.list");
94          retScripts.add("sudo apt-key adv --keyserver hkp://ha.pool.sks-
    keyservers.net:80 --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
    ");
95          retScripts.add("sudo apt-get update");
96          retScripts.add("sudo apt-get install " + configLine.split(":")
    [1] + "-desktop-full");
97          retScripts.add("sudo rosdep init");
98          retScripts.add("rosdep update");
99          retScripts.add("echo \"source /opt/ros/kinetic/setup.bash\" >>
    ~/.bashrc>");
100         retScripts.add("source ~/.bashrc");
101         continue;
102       }
103       if (configLine.contains("jailhouse")) {
104         retScripts.add("#Jailhouse installation");
105         retScripts.add("git clone https://github.com/siemens/jailhouse.
    git");
106         retScripts.add("make install");
107         continue;
108       }
109       if (configLine.contains("docker")) {
110         retScripts.add("#Docker installation");
111         retScripts.add("sudo apt-get install apt-transport-https ca-
    certificates curl  gnupg-agent  software-properties-common");
112         retScripts.add("curl -fsSL https://download.docker.com/linux/
    ubuntu/gpg | sudo apt-key add -");
113         retScripts.add("sudo apt-key fingerprint 0EBFCD88");
114         retScripts.add("sudo add-apt-repository \\n   \"deb [arch=amd64]
     https://download.docker.com/linux/ubuntu \\n   $(lsb_release -cs) \\
    n   stable\"");
115         retScripts.add("sudo apt-get update");
116         retScripts.add("sudo apt-get install docker-ce docker-ce-cli
    containerd.io");
117         // retScripts.add("sudo docker run hello-world");
118         retScripts.add("#ROS Image for Docker");
119         retScripts.add("sudo docker pull ros");
120         retScripts.add("sudo docker network create rosnet");
121         retScripts.add("/bin/bash -c 'docker run -it --net rosnet --name
     master ros roscore'");
122         retScripts.add("#Prepare ROS environment");
123         retScripts.add("sudo cp publisher-new.cpp tmpfs/partition2/
    scripts/rosapp/src/app ");
124         retScripts.add("sudo cp subscriber-new.cpp tmpfs/partition2/
    scripts/rosapp/src/app ");
125         continue;
126       }
127
128       // injects side scripts into main file system
```

```
129         ret.add(37, "cp " + this.SCRIPTS_INJECTION_FILE_NAME + " tmpfs/
        partition2/scripts");
130       }
131     }
132     try {
133       Files.write(out, ret);
134       Files.write(outScripts, retScripts);
135 
136     } catch (Exception ex) {
137       <no statements>
138     }
139     ret.addAll(retScripts);
140     return ret;
141 
142 }
```

Appendix E

Process Partition Behavioral function

```
1
2  #Private method that parses a single partition
3  #@param configStrings The list of strings for a SINGLE partition,
       created in the caller method.
4  #@param configStrings Each string is created with appended prefix to
       identify the section of the #configuration file which it targets
5  #@param configsList The list of configurations read from the
       hypervisorconfig.c file. Used as basis for the #new configuration
       file
6  #@return The list of strings with the correct configuration for that
       partition
7
8  private LinkedList<string> processPartition(LinkedList<string>
       configStrings, LinkedList<string> configsList) {
9    string currentPartName = "NULL_PART";
10   LinkedList<string> finalList = new LinkedList(configsList);
11   for (string hypervisorCfgEntry : configsList) {
12     for (string configSingle : configStrings) {
13       if (configSingle.contains(this.CORES_SEARCH_TERM) &&
       hypervisorCfgEntry.contains(this.CORES_SEARCH_TERM_IN_FILE)) {
14         int indexCpus = configsList.indexOf(hypervisorCfgEntry);
15         string newEntry = "\t\t" + calculateCoreBitmask(configSingle) +
       ",";
16         finalList.remove(indexCpus + this.TRAILING_LINES_CORE_CONFIG);
17         finalList.add(indexCpus + this.TRAILING_LINES_CORE_CONFIG,
       newEntry);
18         continue;
19       }
20       if (configSingle.contains(this.NAME_SEARCH_TERM) &&
       hypervisorCfgEntry.contains(this.NAME_SEARCH_TERM_IN_FILE)) {
21         int indexName = configsList.indexOf(hypervisorCfgEntry);
22         string newEntry = "\t\t" + this.NAME_SEARCH_TERM_IN_FILE + "\""
       + configSingle.split(":")[1] + "\",";
23         finalList.remove(indexName);
24         finalList.add(indexName, newEntry);
25         currentPartName = configSingle.split(":")[1];
26         continue;
27       }
28       if (configSingle.contains(this.MEMORY_REG_SEARCH_TERM)) {
29         string[] memConfig = configSingle.split(":");
30         // if file configuration contains memory target -- (UART OR
       SIMILAR)
31         // [1] target
32         // [2] phys start
33         // [3] virt start
34         // [4] size
```

```
35          if (hypervisorCfgEntry.contains(memConfig[1])) {
36            int indexName = configsList.indexOf(hypervisorCfgEntry);
37            string physConfig = "\t\t" + this.PHYS_START_PREFIX +
    memConfig[2] + ",";
38            string virtConfig = "\t\t" + this.VIRT_START_PREFIX +
    memConfig[3] + ",";
39            string sizeConfig = "\t\t" + this.SIZE_PREFIX + memConfig[4] +
     ",";
40            finalList.remove(indexName + this.TRAILING_LINES_PHYS_CONFIG);
41            finalList.add(indexName + this.TRAILING_LINES_PHYS_CONFIG,
    physConfig);
42            finalList.remove(indexName + this.TRAILING_LINES_VIRT_CONFIG);
43            finalList.add(indexName + this.TRAILING_LINES_VIRT_CONFIG,
    virtConfig);
44            finalList.remove(indexName + this.TRAILING_LINES_SIZE_CONFIG);
45            finalList.add(indexName + this.TRAILING_LINES_SIZE_CONFIG,
    sizeConfig);
46            continue;
47          }
48        }
49        if (configSingle.contains(this.ADDR_SEARCH_TERM)) {
50          string[] addrConfig = configSingle.split(":");
51          if (hypervisorCfgEntry.contains(addrConfig[1])) {
52            int indexName = configsList.indexOf(hypervisorCfgEntry);
53            string addressConfig = "\t\t" + this.ADDRESS_PREFIX +
    addrConfig[2] + ",";
54            string baseConfig = "\t\t" + this.PINBASE_PREFIX + addrConfig
    [3] + ",";
55            string bitmapConfig1 = "\t\t" + this.PINBITMAP_PREFIX;
56            string[] bitmapConfig2Split = addrConfig[4].split(",");
57            finalList.remove(indexName + this.TRAILING_LINES_ADDR_CONFIG);
58            finalList.add(indexName + this.TRAILING_LINES_ADDR_CONFIG,
    addressConfig);
59            finalList.remove(indexName + this.
    TRAILING_LINES_PINBASE_CONFIG);
60            finalList.add(indexName + this.TRAILING_LINES_PINBASE_CONFIG,
    baseConfig);
61            finalList.remove(indexName + this.
    TRAILING_LINES_PINBITMAP_CONFIG_1);
62            finalList.add(indexName + this.
    TRAILING_LINES_PINBITMAP_CONFIG_1, bitmapConfig1);
63            for (int i = 0; i < bitmapConfig2Split.length; i++) {
64              finalList.remove(indexName + this.
    TRAILING_LINES_PINBITMAP_CONFIG_2 + i);
65              finalList.add(indexName + this.
    TRAILING_LINES_PINBITMAP_CONFIG_2 + i, i + 1 < bitmapConfig2Split.
    length ? "\t\t\t" + bitmapConfig2Split[i] + "," : "\t\t\t" +
    bitmapConfig2Split[i]);
66            }
67            continue;
68          }
69        }
70        if (configSingle.contains(this.DEVICE_SEARCH_TERM)) {
71
72          string[] deviceConfig = configSingle.split(":");
73          // [1] type
74          // [2] region
75          string typeString = this.CONST_PCI_DEVICE_IDENTIFIER_IVHSMEM.
    split("=")[0];
```

```
76          if (hypervisorCfgEntry.contains(typeString) && this.
    CONST_PCI_DEVICE_IDENTIFIER_IVHSMEM.split("=")[1].contains(
    deviceConfig[1])) {
77            int indexName = configsList.indexOf(hypervisorCfgEntry);
78            // REGION
79            finalList.remove(indexName + this.TRAILING_LINES_REGION_CONFIG
    );
80            finalList.add(indexName + this.TRAILING_LINES_REGION_CONFIG, "
    \t\t\t\t" + this.REGION_PREFIX + deviceConfig[2] + ",");
81            continue;
82          }
83        }
84        if (configSingle.contains(this.CHANNEL_SEARCH_TERM)) {
85          // [1] from
86          // [2] to
87          // [3] size
88          // [4] target permission
89          // [5] permissions
90          // [6] target permission 2
91          // [7] permissions 2
92          // [8] protocol
93          string[] channelConfig = configSingle.split(":");
94          if (channelConfig[1].equals(currentPartName) || channelConfig
    [2].equals(currentPartName)) {
95            if (hypervisorCfgEntry.contains(channelConfig[8])) {
96              int indexName = configsList.indexOf(hypervisorCfgEntry);
97              string size = "\t\t\t" + this.SIZE_PREFIX + " 0x" +
    converToHexByte(channelConfig[3].substring(0, channelConfig[3].length
    () - 2), channelConfig[3].substring(channelConfig[3].length() - 2,
    channelConfig[3].length())) + ",";
98              string flags = "\t\t\t" + this.FLAGS_PREFIX;
99              for (int i = 5; i < 8; i += 2) {
100               string[] permissionsSplit = channelConfig[i].split(",");
101               if (channelConfig[i - 1].equals(currentPartName)) {
102                 for (string perm : permissionsSplit) {
103                   switch (perm) {
104                     case "read" :
105                       flags += this.FLAGS_JAILHOUSE_READ;
106                       break;
107                     case "write" :
108                       flags += this.FLAGS_JAILHOUSE_WRITE;
109                       break;
110                 flags += this.FLAGS_JAILHOUSE_SHARED;
111             finalList.remove(indexName + this.
    TRAILING_LINES_CHANNEL_SIZE_CONFIG);
112             finalList.add(indexName + this.
    TRAILING_LINES_CHANNEL_SIZE_CONFIG, size);
113             finalList.remove(indexName + this.
    TRAILING_LINES_FLAGS_CONFIG);
114             finalList.add(indexName + this.TRAILING_LINES_FLAGS_CONFIG,
    flags);
115             continue;
116   return finalList;
117 }
```

Appendix F

JetsonTX2 Jailhouse Configuration

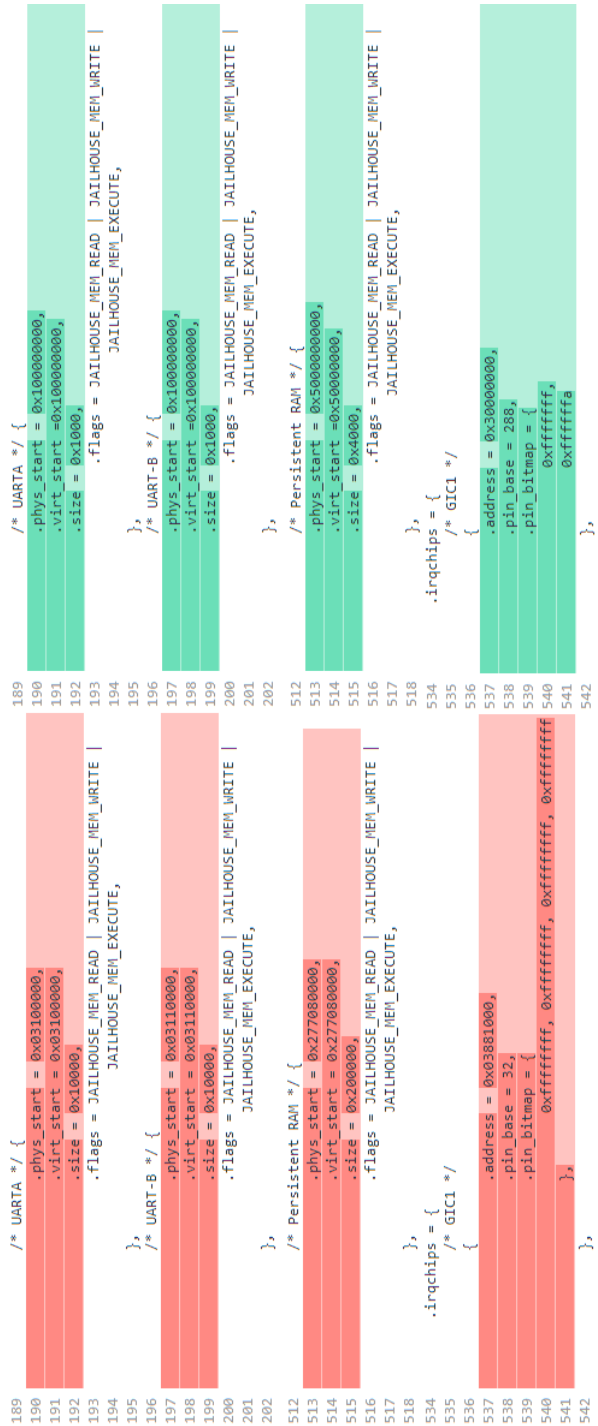

Figure F.1: Differences between hypervisor configurations

Appendix G

Kernel Configuration file

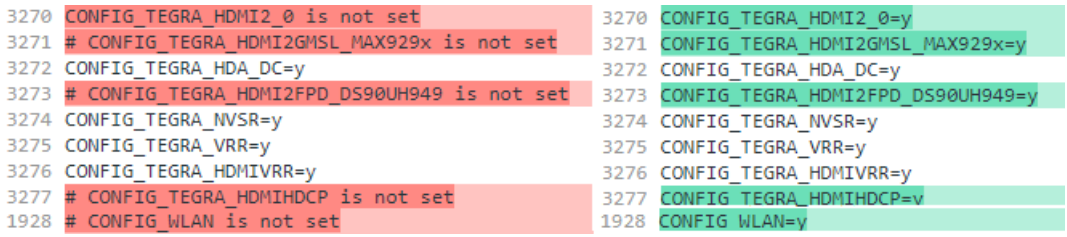

Figure G.1: Differences between kernel configurations

Appendix H

Partition Configuration changes

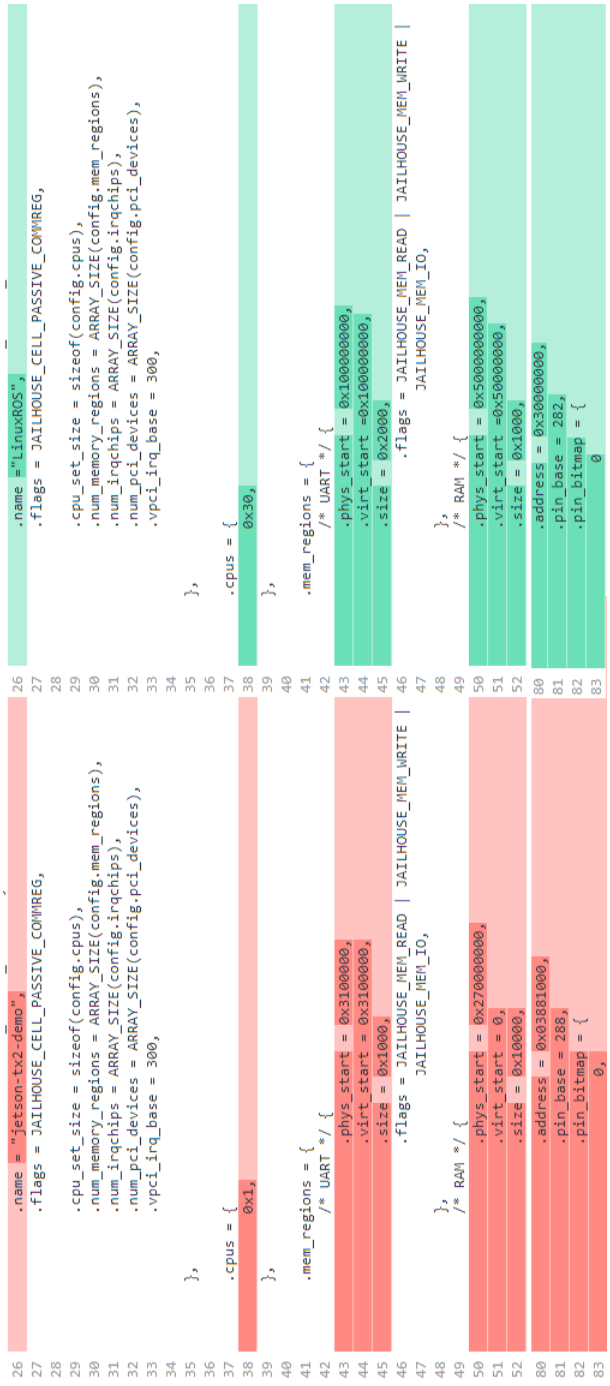

Figure H.1: Jailhouse partition (cell) configuration diff