Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

aufgrund eines Beschlusses
des Deutschen Bundestages

Integrierte Design- und Entwicklungsumgebung für Aerospace

MZB-IDEA: Modular zertifizierbares Betriebssystem für Integrierte Design- und Entwicklungsumgebung für Aerospace

Dokument-Version	1.0
Datum	15.06.2023
Berichtszeitraum	01.04.2019-31.12.2022
Verbreitungsgrad	Öffentlich
Projekt	MZB-IDEA
Laufzeit	1.4.2019 – 31.12.2022
Förderkennzeichen	FZK20Y1712J

1 Kurzdarstellung

1.1 Aufgabenstellung

Die Luftfahrtindustrie steht vor der Herausforderung, sicher und energieeffizient eine wachsende Mobilität zu unterstützen. Dabei soll der Entwicklungsprozess für die Systemarchitektur zur Auslegung neuer fehlertoleranter, sicherheitskritischer Systemarchitekturen und neuer Systemkomponenten, unterstützen und somit die Beherrschung immer komplexer werdender Systeme erleichtern. Langfristig können somit Systementwicklungs- und die Fertigungskosten reduziert werden. Auch führt die Standardisierung von Schnittstellen und Definition gemeinsamer generischem Datenaustauschformate zu Erleichterungen der Wartung. Die Verbesserung der Entwicklung von sicheren eingebetteten Systemen, ermöglicht leistungsfähige und sichere Gesamtsysteme in der Zukunft, insbesondere weil im Bereich Cybersicherheit zukünftige Wartungsprozesse durch nötige Sicherheitsupdates beeinflusst werden.

Das Ziel des Gesamtvorhabens von IDEA war die Definition einer integrierten Entwicklungsumgebung für Avionik. Der Methoden- und Toolbaukasten soll für den gesamten Entwicklungsprozess von fehlertoleranten, sicherheitskritischen eingebetteten Systemen, auf die Belange (Zulassungsaspekte) der Luftfahrtindustrie in Einklang mit Flightpath 2050, zugeschnitten sein. Dies wurde unter anderem durch ein breites Spektrum an teilnehmenden Unternehmen (Systemhersteller/OEM, Zulieferer/TIER1, Dienstleister und Technologielieferanten/Hochschulen und Forschungseinrichtungen) gewährleistet.

Die Motivation für das Teilvorhaben IDEA-MZB ist folgende: Bisher wurde bei SYSGO MILS-Integration zumeist auf Projektebene, aber wenig aus System- und Toolsicht betrieben. Ziel des Teilvorhabens MZB-IDEA ist es, unsere Erfahrung in Partitionierung auch bei Systementwicklungs- und Toolprozessen einzubringen, unsere Position in den Querschnittsthemen Funktionaler- und Cybersicherheit zu stärken und die Projektschnittstellen für unser Entwicklungsumgebung zu verbessern.

1.2 Voraussetzungen, unter denen das Vorhaben durchgeführt wurde

Eine sichere, skalierbare, performante und wiederverwertbare Systemarchitektur ist der Grundstein der Gesamtsystementwicklung. Die Systemarchitektur muss sorgfältig ausgelegt und aus verschiedenen Perspektiven, wie z.B. Funktionale- und Cybersicherheit, geprüft und bewertet werden. Eine nahtlose, integrierte Verarbeitung der erfassten Daten in dem Gesamtsystementwicklungsprozess trägt zur Beherrschung der Komplexität bei. Die auf dem Markt vorhandenen Methoden und Werkzeuge zur Entwicklung von sicheren, skalierbaren und wiederverwendbaren Systemarchitekturen sind zwar sehr vielfältig aber nicht hinreichend abdeckend und integriert. Dies führt zu Methodenbrüchen und Daten-Inkonsistenzen, da für einzelne Entwicklungsschritte wieder auf klassische Methoden wie textuelle Anforderungen, manuelle Checklisten und Folienpräsentationen zurückgegriffen werden muss.

Die Entwicklung findet auf den unterschiedlichen Auslegungs- und Integrationsebenen (System-Software, Software-Hardware) heutiger Luftfahrtsysteme im Wesentlichen isoliert statt. Ein Austausch von Daten (z.B. Modellen) geschieht nur in zu großen Abständen und unter

großen technischen und formalen Problemen, wie z.B. der unklare formale Status, die Anpassung der Daten an unterschiedliche Entwicklungsumgebungen und Standards sowie unterschiedliche Anforderungen an den Detaillierungsgrad der Modelle auf den unterschiedlichen Ebenen. So ist z.B. auch die Standardisierung der APIs von MILS-Systemen keineswegs abgeschlossen. Daher können potentielle Vorteile aktueller Verfahren nicht hinreichend genutzt werden. Heute existieren weder geeignete Modelle für den Austausch von Daten, noch sind die eingesetzten Tools vollumfänglich funktional geeignet, um diesen Transfer zu ermöglichen. Die Definition eines adäquaten Datenmodells zur Verwendung in allen Domänen ist die wesentliche Voraussetzung für eine domänenübergreifende Kooperation und Integration.

1.3 Planung und Ablauf des Vorhabens

Die folgende Tabelle gibt Planung und Ablauf der für das Teilvorhabens relevanten Teilprojekte an:

Teilprojekt	Name	Zeitraum
TP1	Produktentwicklungsprozess	M1-M15
TP2	Domänenschnittstellen	M1-M24
TP3	Anwendungsfälle	M13-M45
TP4	Querschnittsthemen	M6-M45

1.4 Wissenschaftliche und technische Ausgangslage

PikeOS ist ein Betriebssystem, welches explizit für den Einsatz in sicherheitskritischen Umgebungen entwickelt wird. Es ist zudem für Zertifizierbarkeit ausgelegt, sodass es bereits im Luftfahrt- und Automobilbereich sowie in der Bahntechnik zum Einsatz kommt.

Die zu erwartende Komplexität zukünftiger Luftfahrtanwendungen mit einer Vielzahl verteilter, teilweise autonomer Systeme unterschiedlichen Absicherungslevels sowie einer Zunahme von Anzahl und Komplexität der Schnittstellen erschweren dramatisch die Analyse und die Sicherstellung der Absicherung der Systeme gegen nicht autorisierte, bösartige Eingriffe von außen. Auf dem Markt existieren bereits viele Standards, Prozesse, Methoden und Werkzeuge, die die Anforderungen an die Cyber-Sicherheit stellen, die Bedrohungsszenarien modellieren und die Entwicklung sicherer Systeme unterstützen. Diese sind jedoch stark auf Web-basierte Systeme zugeschnitten, behandeln meistens nur einen Aspekt (Cyber-Sicherheit) und gehen nicht auf die Bedürfnisse der Luftfahrt ein (Funktionale, fehlertolerante und sichere Systeme). Weiterhin bieten die vorhandenen Methoden und Werkzeuge keine Unterstützung für integrierte Methoden und Entwicklungsumgebungen von sicheren Systemen.

1.5 Zusammenarbeit mit anderen Stellen

SYSGO arbeitete im Verlauf des Projektes mit der europäischen Behörde für Flugsicherheit (EASA) in einem Workshop zu Zertifizierungsaspekten zusammen.

2 Eingehende Darstellung der Arbeiten und Ergebnisse

2.1 TP1 Produktentwicklungsprozess

Wir haben die Separation-Kernel-Architektur und für eine effiziente DAL-A und Security-Zertifizierung relevanten Aspekte (Standards, Methoden und Werkzeuge für System-Modellierung und Analyse) analysiert, mit Schwerpunkt auf Interpretation von Interferenzen, Konfigurationsanalyse, wobei Performance und WCET stets mit im Auge behalten wurden. SYS hat dabei mit (z.B. in IDEA-Workshops) mit allen anderen Partnern zusammengearbeitet.

In TP1.1 hat SYSGO bei *Standards* die ISO 15408 (Common Criteria for Information Technology Security Evaluation / CC) also z.B. Bedrohungsanalyse vorgestellt, einschließlich Praxiserfahrung bei SYSGO, z.B. das Sicherheitsziel von PikeOS. Wir haben auf Beispiele für Verwendung in der Avionik hingewiesen. SYSGO ist dabei auch auf Cyber-Sicherheitsthemen in anderen Avionik-Standards eingegangen, z.B. DO-356 (2014) und DO-356A (2018) und diese Ergebnisse wurden beim IDEA-Workshop vorgestellt. Unter *Konzepten und Methoden* hat SYSGO MILS (Multiple Independent Levels of Safety / Security) aus dem Projekt EURO-MILS vorgestellt. Wir haben EURO-MILS-Erfahrungen zur Erstellung flugtauglicher gemischtkritischer Systeme unter Verwendung von Trennungskernen und Partitionierung dargestellt. Ebenso hat SYSGO in gemeinsamen Diskussionen mit Partnern andere Anforderungen und Definitionen diskutiert und verbessert.

SYSGO hat vom 22.-24.10.2019 mit zwei Projektpartnern (Avionik und Weltraum) einen PikeOS-Workshop durchgeführt, in dem die Konzept und Benutzung von PikeOS erklärt wurden (z.B. die Verwendung von virtualisiertem Linux auf dem PikeOS-Hypervisor, Volume-Providers).

2.2 TP2 Domänenschnittstellen

Wir haben mit Partnern gemeinsam den Weg von grafischer Benutzerschnittstelle in eine XML-Schnittstelle aus Sicht eines Systemintegrators diskutiert, und ob hier weitere Modellierung und / oder Verifikation ansetzen kann. Z. B. ist unter Gesichtspunkten der Produktentwicklung bei SYSGO eine Trennung von PSP und ASP geboten; aus Nutzersicht hingegen auf einem Board-Support-Package aus funktionaler Sicht der HW-SW-Schnittstelle PSP und ASP eng verbunden sind, und in Schnittstellenmodellen z. T. einheitlich betrachtet werden können.

Eine der Hauptfunktionen eines Betriebssystems ist Hardwareabstraktion für Software. Auch ein minimiertes Betriebssystem wie ein Trennungskern spannt sich in natürlicher Form über die Domänenschnittstelle zwischen sowie Software und Hardware. Das heißt, bei einem Trennungskern (Separation Kernel) wird die genaue Hard-/Softwareschnittstelle zur Laufzeit durch die Konfiguration zur Integrationszeit bestimmt, und der hierbei gegenüber einem herkömmlichen Betriebssystem größere Determinismus ist eines der Hauptfunktion eines Trennungskerns. Aus diesem Grund haben wir uns entschlossen, die geplanten modellbasierten Entwicklungen mit dem Konfigurationswerkzeug eines Echtzeittrennungskerns von SYSGO zu verbinden. Um spätere Verwertungsmöglichkeiten zu unterstützen, ist die von uns gewählte Herangehensweise, dass wir dabei auch untersuchen, wie wir eine gute Produktintegration schaffen, essentiell. Diesbezüglich haben wir insbesondere zunächst einen bereits existierenden Ansatz untersucht, der auf dem W3C Document Object Model (DOM) beruht.

2.2.1 Verwendung von Eclipse Modeling Framework

Die Avionik braucht zuverlässige Betriebssysteme, die ein vorhersehbares Laufzeitverhalten haben und daher statisch konfiguriert werden. SYSGO‘s PikeOS ist ein Betriebssystem, welches für konkrete Anwendungen statisch von einem Systemintegrator konfiguriert wird, z.B. Allokation von Ressourcen und Kommunikationsprimitiven wir Ports, Shared Memory, usw. Dieses wird derzeit mit einer graphischen Benutzeroberfläche und einem selbstdefinierten XML-Format realisiert. Wir haben untersucht, wie dieser Konfigurationsansatz mit der Eclipse-Modeling-Framework realisiert werden kann.

Motiviert durch den Erfahrungsaustausch u.a. mit Partnern haben wir uns danach mit der Sirius-Plattform im Speziellen und dem generischen Eclipse-Modeling-Framework auseinandergesetzt, und eine Beispielskonfigurationsgenerator unter Verwendung eines Eclipse-Modeling-Framework -Modells geschaffen. Wir haben dann Viewpoint Specification Models („odesign“-Format) dazu verwendet, bestimmte Ansichten zu erstellen, beispielsweise, falls der Nutzer diese Sicht will, um eine Partition zu verbergen kann er somit eine bessere Sichtbarkeit realisieren. In einem zweiten Demonstrationsprojekt haben wir diese Viewpoint Specification Models dazu verwendet, um Überlappungen von Speicherallokationen zu entdecken, dabei wurden Queries in AQL (Acceleo Query Language) formuliert.

In Hinblick auf Ansichten, haben wir untersucht, inwieweit Darstellungen von EMF-Modellen mit Sirius OBEO dynamisch dargestellt werden können (z.B. das optionale Verbergen einer für den Nutzer nicht unbedingt notwendigen Systempartition). Des Weiteren wollten wir verstehen, wie logische Anforderungen mit EMF geprüft werden können und haben hierzu die Mächtigkeit der Acceleo Query Language (AQL) analysiert.

Wir haben einen Export als UML XMI erzeugt, um zu validieren, dass Papyrus als weiteres Entwicklungstool genutzt werden kann.

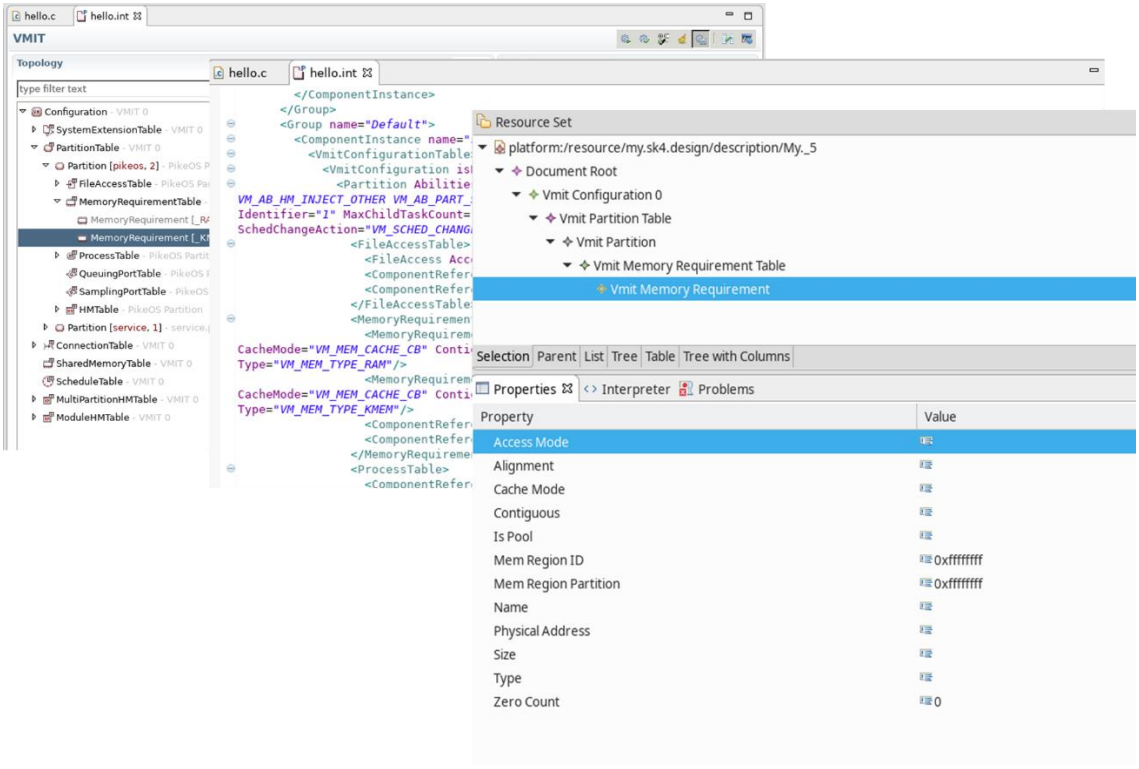


Abbildung 1: CODEO-Detailsichten

Wie in Abbildung 1 gezeigt, bietet CODEO eine Detailsichten an, die Aufgabenstellung: verschiedene Detaillierungsgrade darstellen, z.B. Boardeinstellungen, Cache-Partitionen usw. Im linken Teil der Abbildung ist die traditionelle Konfiguration gezeigt (GUI), im mittleren Teil eine XML-Darstellung und im rechten Teil, eine Darstellung in Eclipse Modeling Framework-Modell.

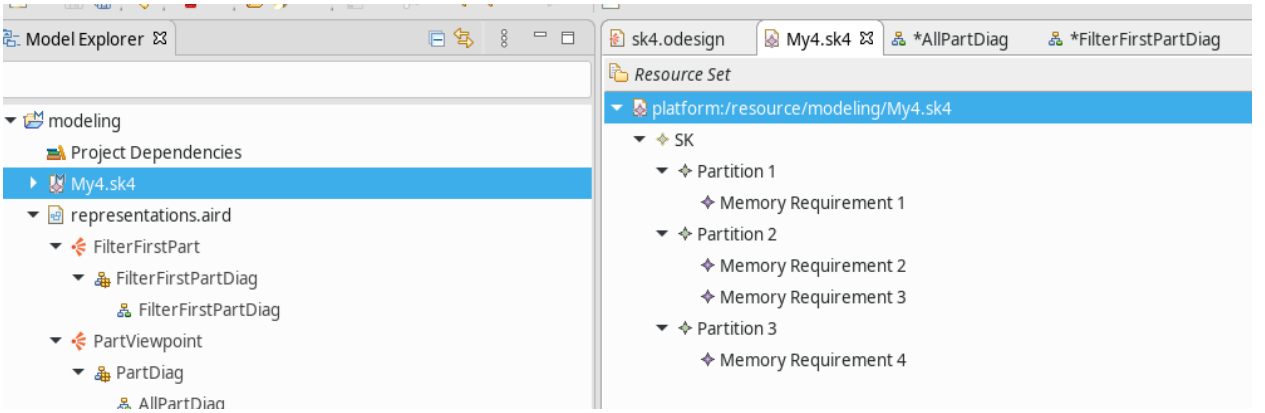


Abbildung 2: Modell: SK -> System partition: MVP

Abbildung 2 zeigt eine Zuordnung von Memory Anforderungen zu Partitionen.

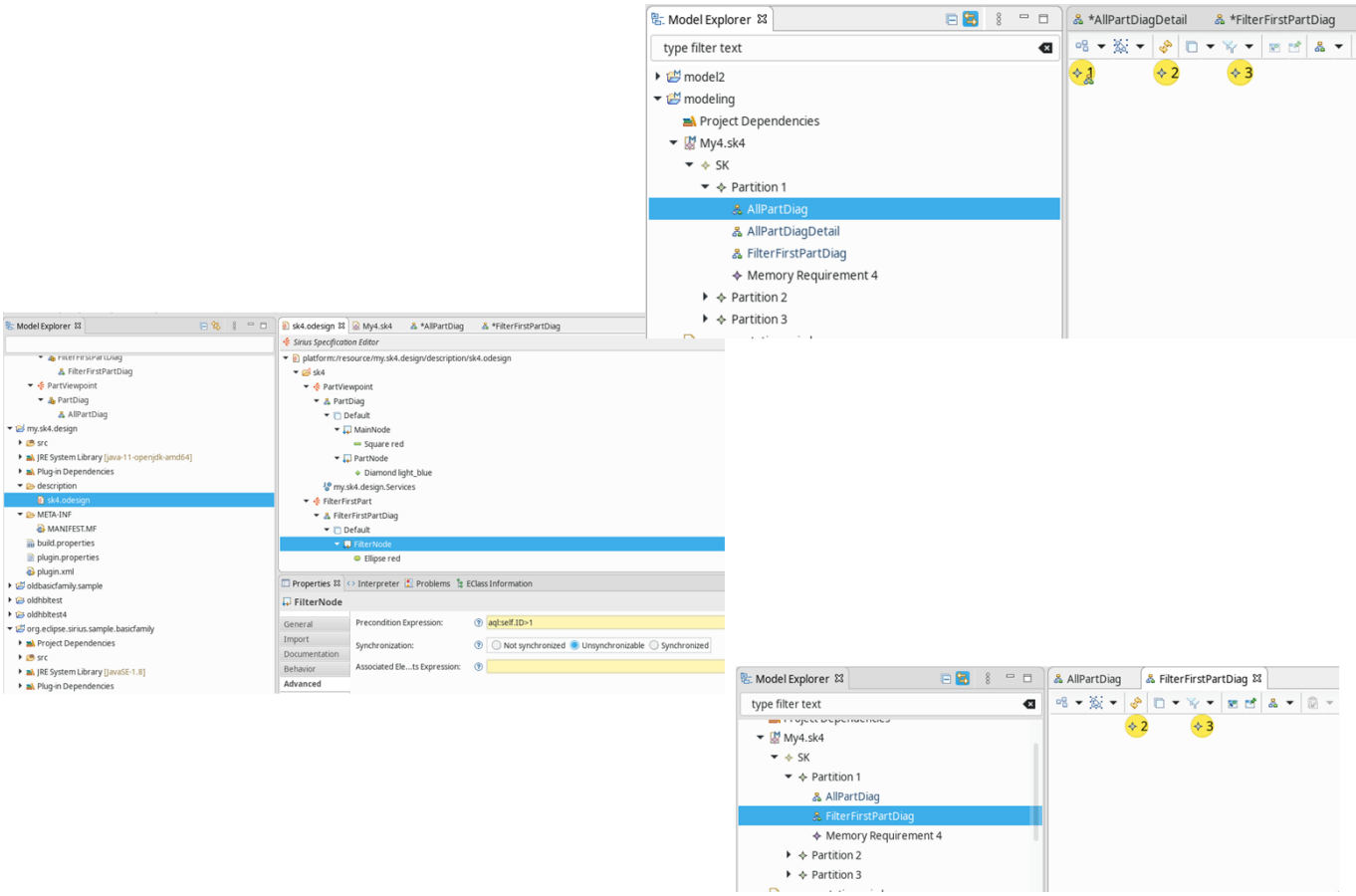


Abbildung 3: Verbergen einer Partition

Wir haben untersucht, inwieweit Darstellungen von EMF-Modellen mit Sirius OBEO dynamisch dargestellt werden können. Abbildung 3 zeigt die Verwendung der Acceleo Query Language (AQL), um eine bestimmte Partitionen aus dem Modell nicht anzuzeigen. Diese Funktionalität ist relevant, um abzuschätzen, ob in EMF-Modell Sichten dieser Art realisieren werden können.

2.2.1.1 UMI-Interface

Ein gängiges Austauschformal für Modelle ist UML XMI (XML Metadata Interchange). Wir haben einen Export des zuvor hergestellten Eclipse EMF Modells als UML XMI erzeugt, um zu validieren, dass Papyrus als weiteres Entwicklungstool genutzt werden kann.

Ein Ausschnitt des Export-UMLs mit XMI-Header ist in Abbildung 4 dargestellt und zeigt die Verwendung des Schemas von EMF / Ecore als Namensraum.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI  xmi:version="20131001"  xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:Ecore="http://www.eclipse.org/uml2/schemas/Ecore/5"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"                    xsi:schemaLoca-
tion="http://www.eclipse.org/uml2/schemas/Ecore/5                 pathmap://UML_PRO-
FILES/Ecore.profile.uml#_z1OFcHjqEdy8S4Cr8Rc_NA">
  <uml:Model          xmi:id="_s5tFgPI4Eeuqydw3KOg_xw"          name="MILSSystem"
URI="htttp://sysgo.idea">
[…]
```

Abbildung 4: XMI header

Eine Sicht in Eclipse als UML-Diagramm ist in Abbildung 1 wiedergegeben.

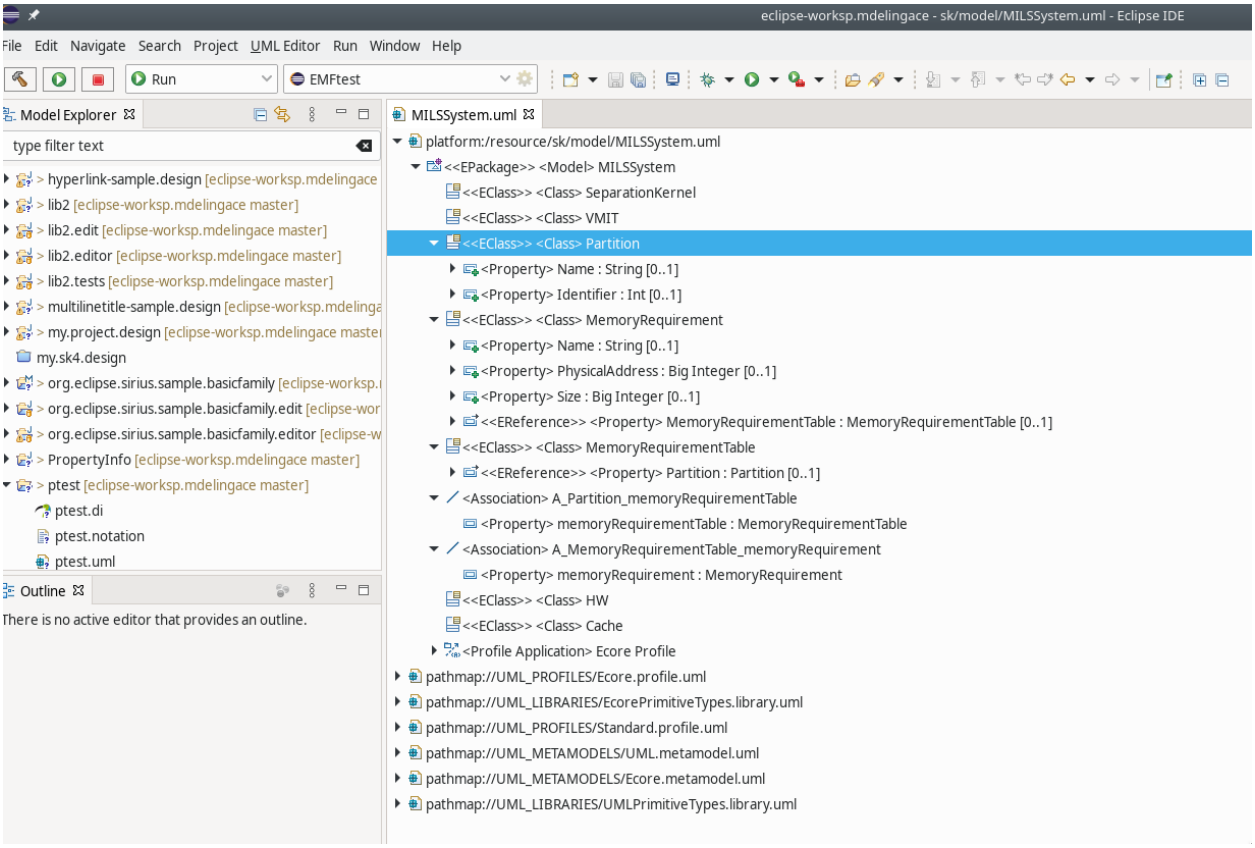


Abbildung 5: Eclipse-Sicht von UML XMI

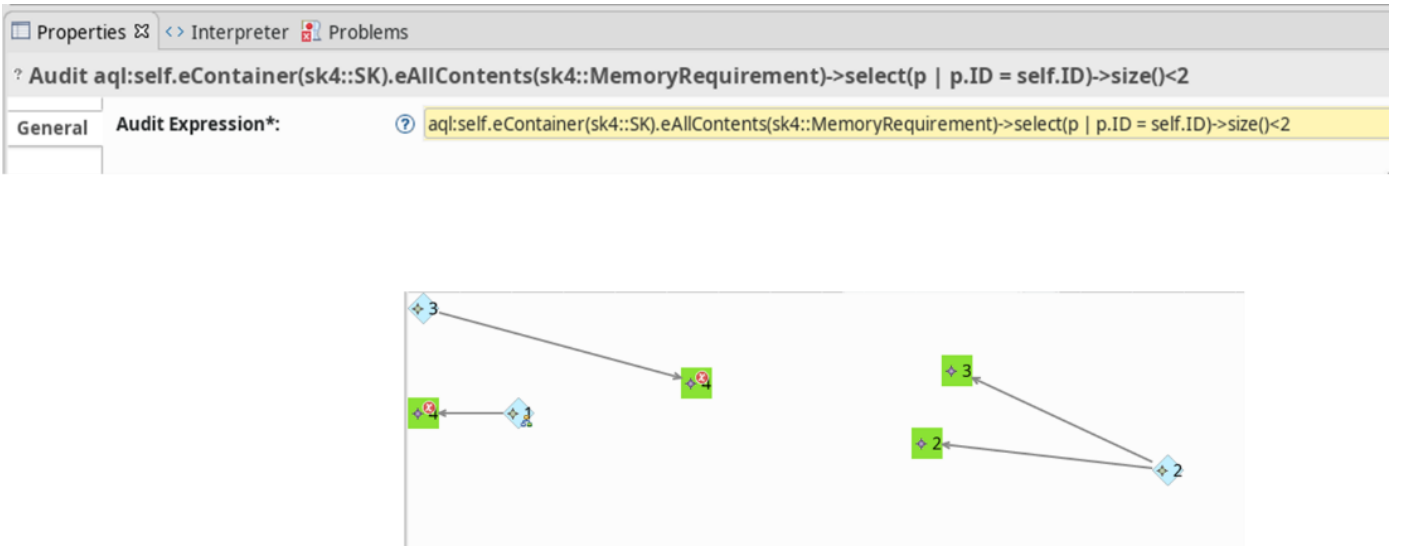


Abbildung 6: Prüfung von Eindeutigkeit der Memory-Zuweisung

Abbildung 6 zeigt die Verwendung von Acceleo Query Language (AQL) um die Eindeutigkeit von Memory-Zuordnungen zu analysieren.

2.2.2 Unterstützung von Speicherpartitionierung

Als hardwarenahe Maßnahme zur Leistungssteigerung und Erhöhung des Determinismus in MILS Separation Kernels haben wir Arbeiten bei einem Partner zur Speicherpartitionierung dokumentiert und entsprechend unterstütz (z.B. mit Hinweisen zur Konfiguration und Performanceevaluierung). Unsere Speicherpartitionierung erlaubt nun eine in PikeOS-Integrationsprojekte eingefasste Konfiguration auf Partitionsebene in genau definierte zusammenhängende Speicherbereiche zu fassen. Hier waren lange Zeit nur direkte Manipulationen über das PSP möglich,

2.3 TP3 Anwendungsfälle

Im Moment sind in der Luft- und Raumfahrt verschiedene CPU-Prozessorarchitekturen (z.B. PowerPC, ARM R52, ARM A Cores, RISC-V) auf dem Markt. Portabilität durch Hardwareabstraktion ist daher wünschenswert. Wir haben Architekturen (z.B. für die Anwendungsfälle von Mehrpartitionssystemen mit Multiprozessor Hardware Abstraction Layer (HAL), portabler Signalkomponente unter Verwendung von HAL, Sicherheitsinfrastruktur, Sicherheitsgateway mit „Ready-to-Run-Konfiguration“) in Avionik entworfen und verfeinert.

2.3.1 Gemischt kritischer Demonstrator für Weltraumanwendung

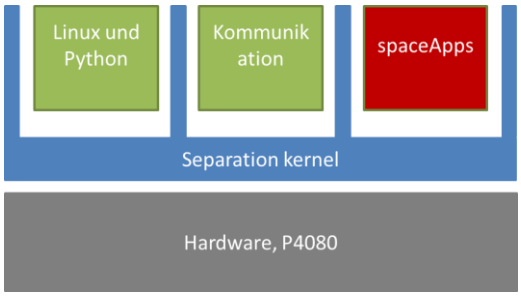


Abbildung 1: Partitionslayout

Im gemischt kritischen Demonstrator für Weltraumapplikationen wird der Chipsatz von NXP PowerPC P4080 verwendet, einschließlich Netzwerkschnittstelle DPAA (data path acceleration architecture). Im Demonstrator kommuniziert ELinOS Embedded Linux mit anderen PikeOS-Partitionen über PikeOS-Queuingports, die innerhalb von ELinOS als Linux-Chardevice abgebildet werden. Eine Kommunikationspartition verbindet beide Anwendungen. Wir wählen eine Realisierung als drei Entwicklungsprojekte, davon eine Applikation zu Kommunikationspartitionen, eine Integrationsprojekt für ELinOS Embedded Linux, und ein Integrationsprojekt für das Gesamtsystem, das auch andere Binäranwendungen des Satellitensystems integriert. ELinOS erlaubt es, das Embedded-Linux-System mit einem zentralen grafischen Benutzeroberfläche zu konfigurieren, z.B. für die Konfiguration von Netzwerk und SSH-Server. Python wurde über ein externes Dateisystem eingebunden, das bei der Kompilation in das übersetzte Programm „Binary“ eingefügt wurde. Die Kommunikationspartition enthält eine

Custom-App zum Auslesen von Queuingports. Der Anwendungsfall wurde in einem Laborsystem aufgesetzt, mit den Features: SSH-Zugang, Python (zunächst Python2, dann Python3), und Kommunikation von ELinOS nach PikeOS.

Im Demonstrator kommuniziert ELinOS Embedded Linux mit anderen PikeOS-Partitionen über PikeOS-Queuingports, die innerhalb von ELinOS als Linux-Chardevice abgebildet und angesprochen werden. Eine Kommunikationspartition verbindet beide Anwendungen. Die Kommunikationsarchitektur mit ELinOS und nativen-Applikationen wurde auch auf den PowerPC-qemu-Emulator portiert und ebenfalls Airbus als Projekte zur Verfügung gestellt. Für die Erweiterung der ELinOS-Partition wurde die zugehörige Infrastruktur erweitert, d.h diese wurde auf der passenden Entwicklungsumgebung neu gebaut. Dazu wurde ein Custom-RPM generiert (Schreiben von Specfile und Bau von RPM mit dem Kreuzkompiler für die PowerPC-Architektur), was einen vorhergehenden Ansatz, Python3 über die CODEO-app.rootfs-Schnittstelle manuell anzubinden, ersetzt. Zudem wurde das venv-Modul als virtuelle Pythonumgebung bereitgestellt, um für verschiedene Python-Anwendungen verschiedene Abhängigkeitsbäume realisieren zu können.

Wir haben die Cybersecurity-Demonstrator-Architektur validiert in Hinblick auf Gast-Pythonanwendungen, mögliche Integrationsszenarien für die pip-3-Installation (Filesystem, RAMFS/scriptbasiert/Netzwerkmount) und auch die mögliche Portabilität auf MPU diskutiert.

Auf dem Anwendungsfall haben wir einen speziellen Treiber für die DPAA (Data Path Acceleration Architecture) eingesetzt, der einen minimalen Code-Footprint hat (weitere Details zu dem Treiberdesign siehe Bericht vom Februar 2022 über 2. HJ 2021). Insbesondere haben wir im Berichtszeitraum ASI bei der Konfiguration von DPAA2 für die Linux-Partition (siehe Abbildung 2) und Allokation der Linux-RAMDisk unterstützt.

Fehler! Verweisquelle konnte nicht gefunden werden. zeigt das Satellitendemo: ELinOS kommuniziert mit der anderen PikeOS-Partitionen über PikeOS-Queuingports, die innerhalb von ELinOS als Chardevice abgebildet werden. Eine Kommunikationspartition kann verwendet werden, um Linux mit anderen Anwendungen kommunizieren zu lassen..

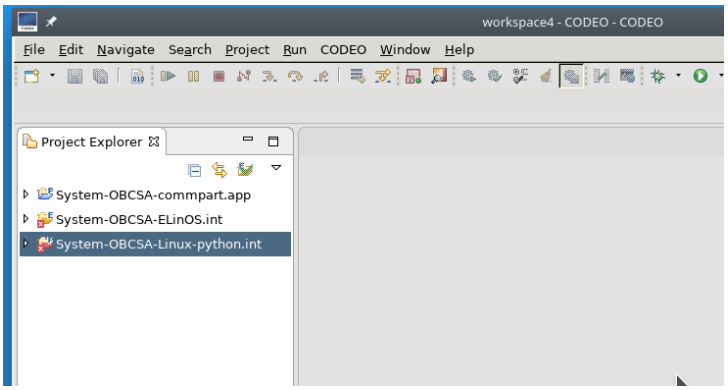


Abbildung 7: Anlegen der Projekte

Abbildung 7 zeigt eine Realisierung als drei Entwicklungsprojekte, davon eine Applikation zu Kommunikationspartitionen, ein Integrationsprojekt für ELinOS Embedded Linux, und ein Integrationsprojekt für das Gesamtsystem, das auch andere Binäranwendungen des Satellitensystems integriert.

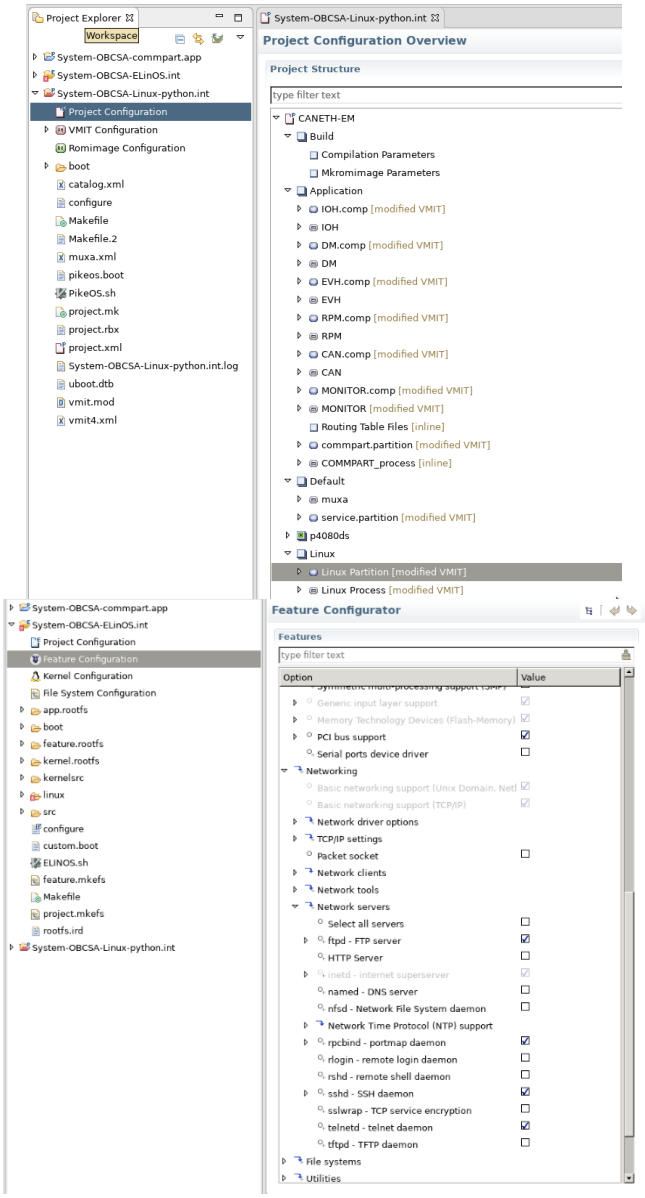


Abbildung 8: Integration in Baseline, Konfiguration Netzwerk und SSH

Das Integrationsprojekt bindet insbesondere Satellitenanwendungen ein (Abbildung 8 links). ELinOS erlaubt es, das Embedded-Linux-System mit einem zentralen GUI zu konfigurieren. Abbildung 8 rechts zeigt die Konfiguration von Netzwerk und SSH-Server.

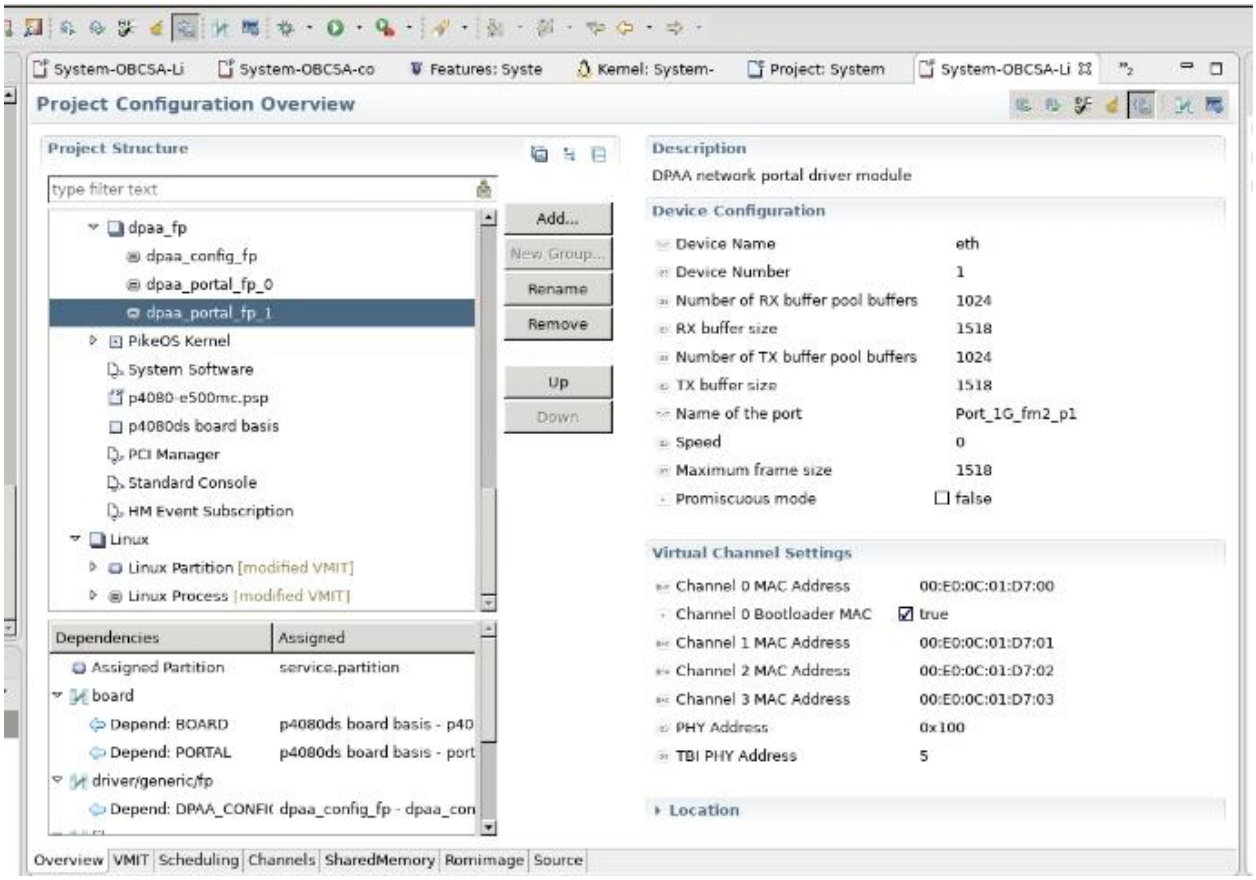


Abbildung 9: DPAA-Nutzung

Als Netzwerkschnittstelle wird NXP PowerPC P4080 DPAA (data path acceleration architecture) verwendet (Abbildung 9).

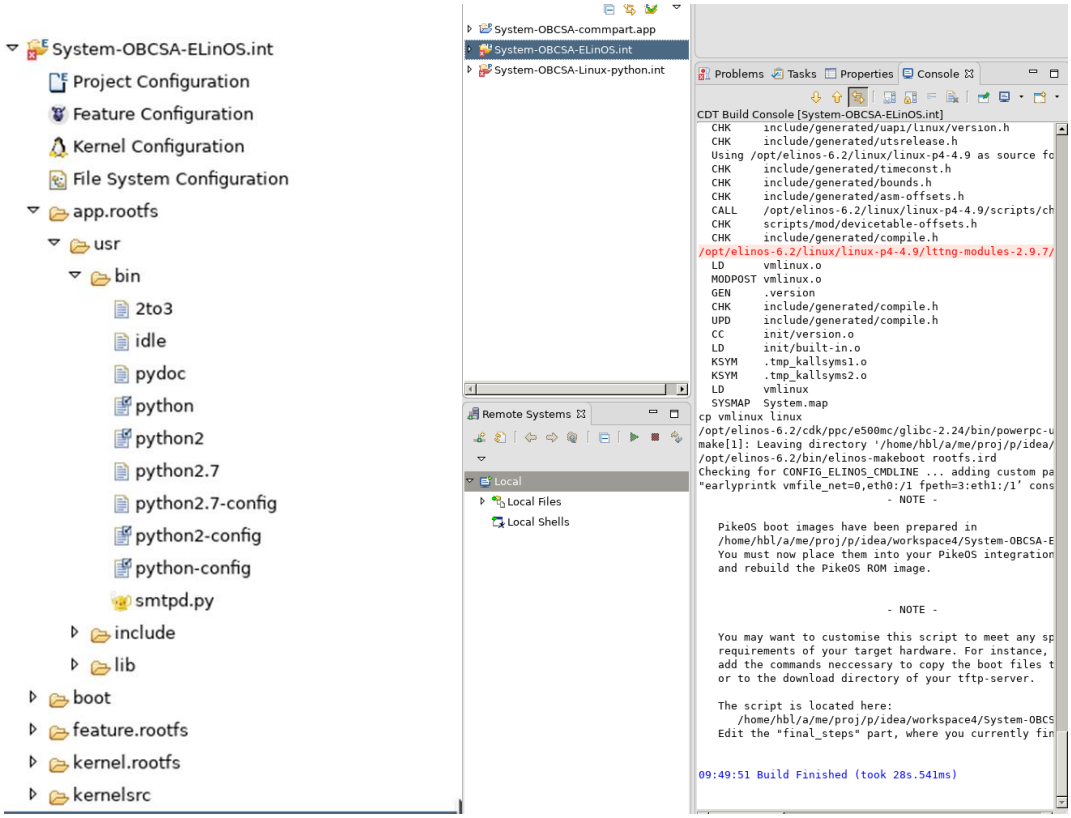


Abbildung 10: Einbinden von Python in ELinOS und Kompilation.

Abbildung 10 zeigt, dass wir Python über ein externes Dateisystem eingebunden haben, das bei der Kompilation in das Binary eingefügt wird.

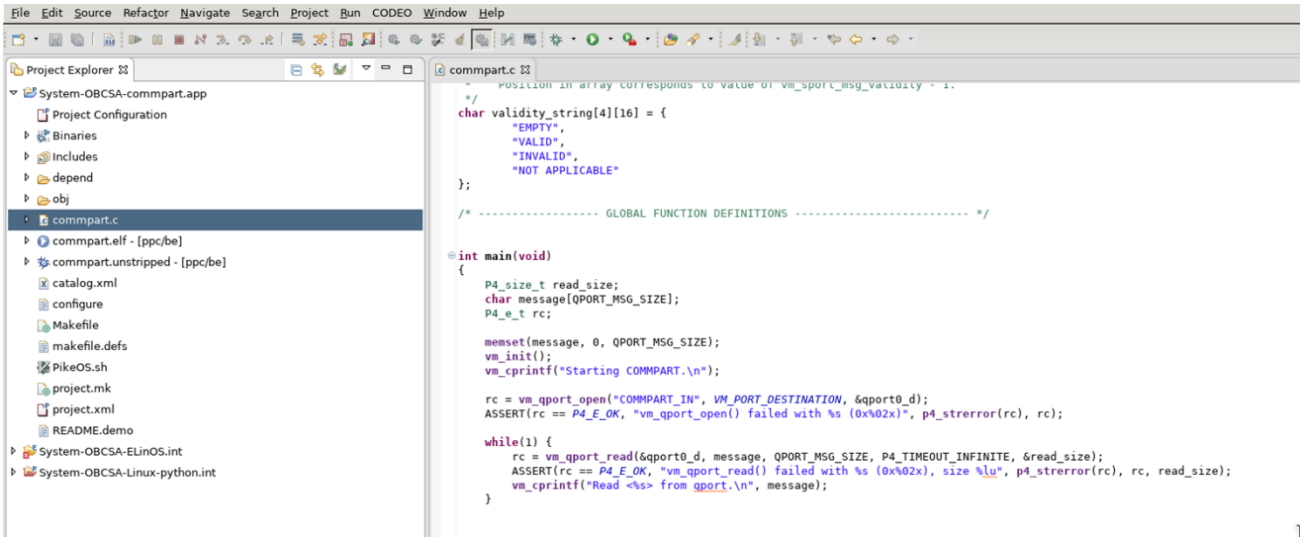


Abbildung 11: Kommunikationspartition über Queuing Port

Abbildung 11 zeigt das Auslesen eines Queuingports in der Kommunikationspartition.

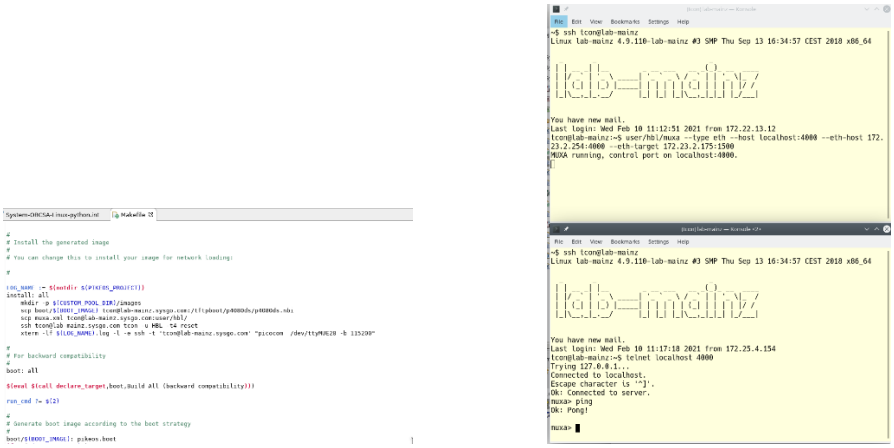


Abbildung 12: Ausführung im Lab

Abbildung 12 zeigt den Zugang zum Demonstrator über die Muxa-Schnittstelle.

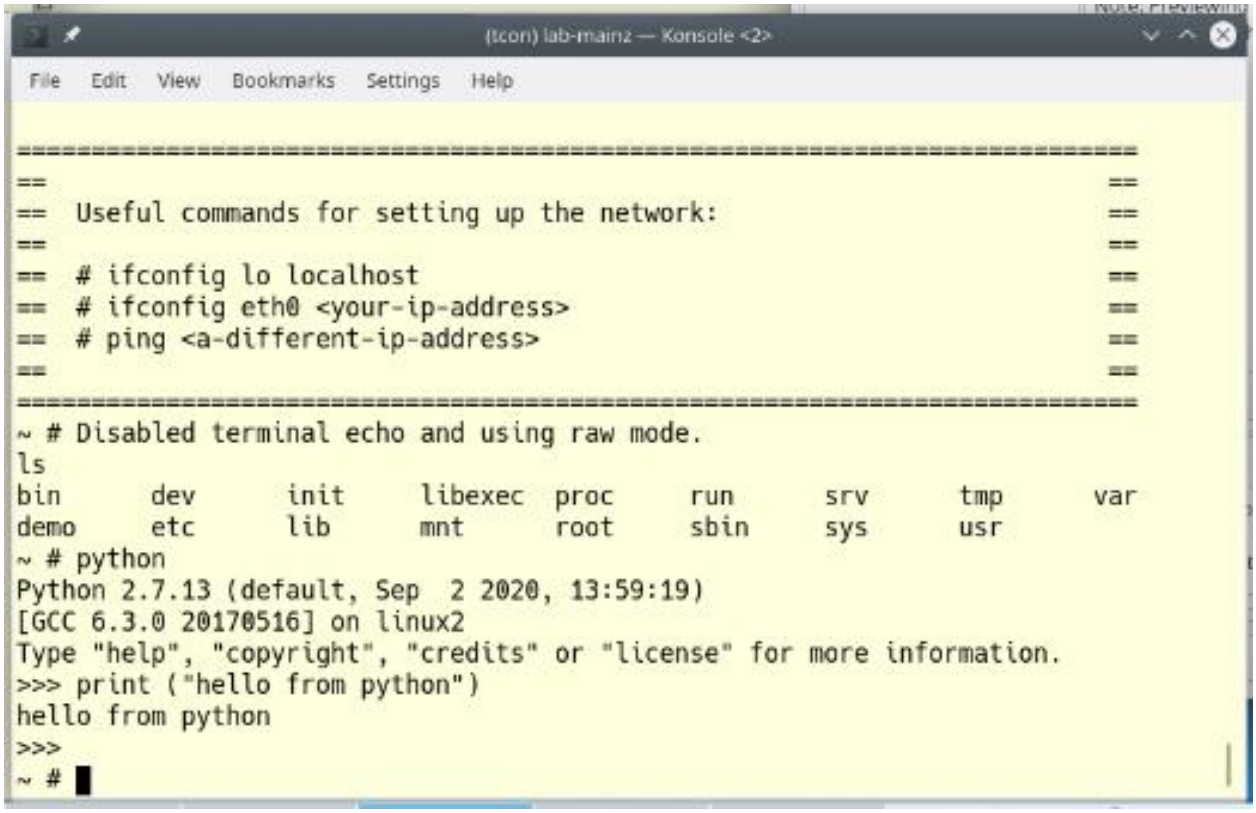


Abbildung 13: Python in ELinOS

Abbildung 13 zeigt die Ausführung von Python auf dem Satellitendemonstrator.

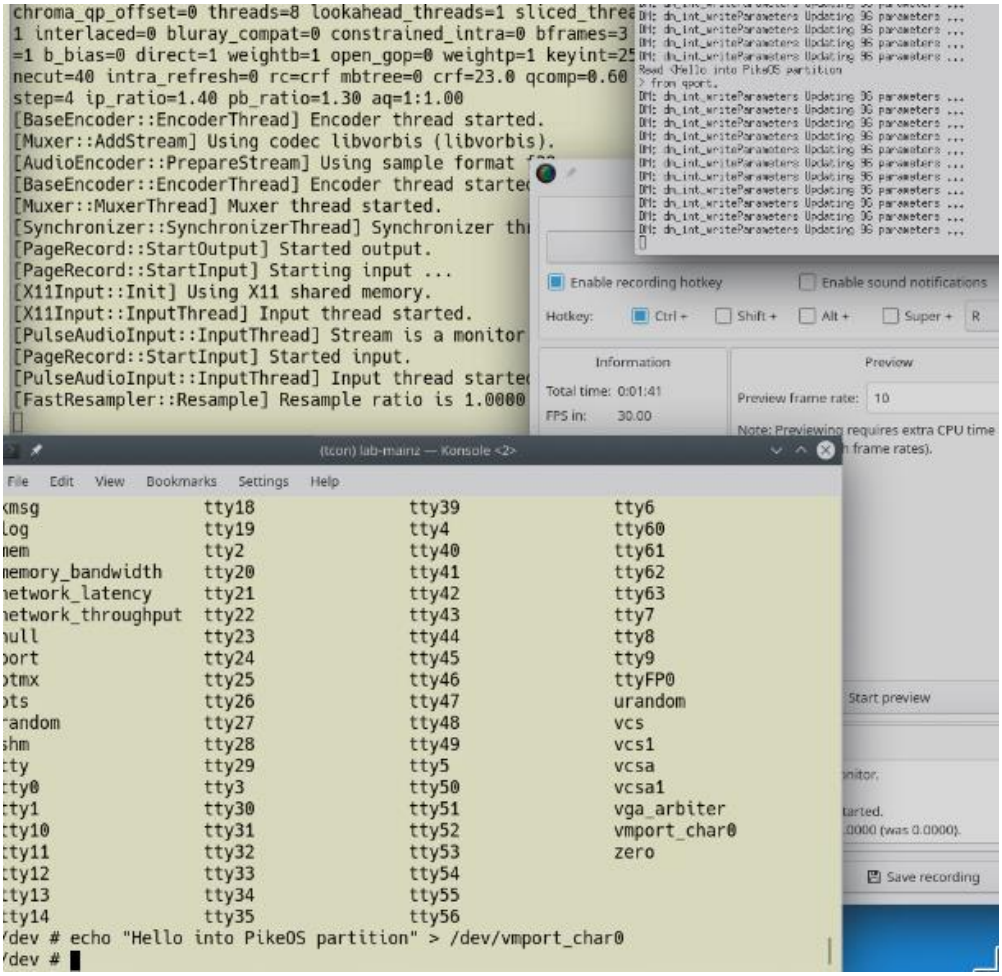


Abbildung 14: Schreiben von ELinOS auf Kommunikationspartition

Abbildung 14 zeigt die Kommunikation von Linux nach PikeOS: ELinOS virtualisiert Queuing-Ports als Char-Devices. Die Abbildung zeigt links ein Bild von einer Linux-Partition und rechts oben über den Muxa-Output das Ankommen der Nachricht auf dem Demonstrator („Read <Hello into PikeOS partition > from qport“).

Die ELinOS-Partition im Satellitendemo wurde um Python3 und die zugehörige Infrastruktur erweitert, dh diese wurde auf der passenden Buildumgebung neu gebaut. Dazu wurde ein Custom-RPM generiert (Schreiben von Specfile und Bau von RPM mit dem Kreuzkompiler für die PowerPC-Architektur), was einen vorhergehenden Ansatz, Python3 über die CODEO-app.rootfs-Schnittstelle manuell anzubinden, ersetzt. Zudem wurde das venv-Modul als virtuelle Pythonumgebung bereitgestellt, um für verschiedene Python-Anwendungen verschiedene Abhängigkeitsbäume realisieren zu können.

Abbildung 15 zeigt den Inhalt von Python3 für ELinOS (geöffnet mit rpm2cpio).

Abbildung 15: Inhalt von Custom-RPM Python 3 für ELinOS 6.2

Zunächst muss das Custom-RPM auf dem Host installiert werden mit „/opt/elinos-6.2/bin/elinos-rpm -i elinos-python3-ppc_e500mc-glibc-2.24-3.7.3-testversion.noarch.rpm“. Danach wird im ELinOS-Konfigurator der RPM-Inhalt vom Local File Systems des Hosts in dem Ziel-Filesystem hinzugefügt. Der ELinOS-Konfigurator bietet eine Vorschau des Ziel-Filesystems. Beim Übersetzen des Projektes wird das Ziel-Filesystem nach den Vorgaben der Konfiguration erstellt und in die Boot Dateien integriert.

Abbildung 16 zeigt die Installation von Python3, wobei die Dateien des installierte RPMs vom „Local File System“ ins Target File System kopiert werden.

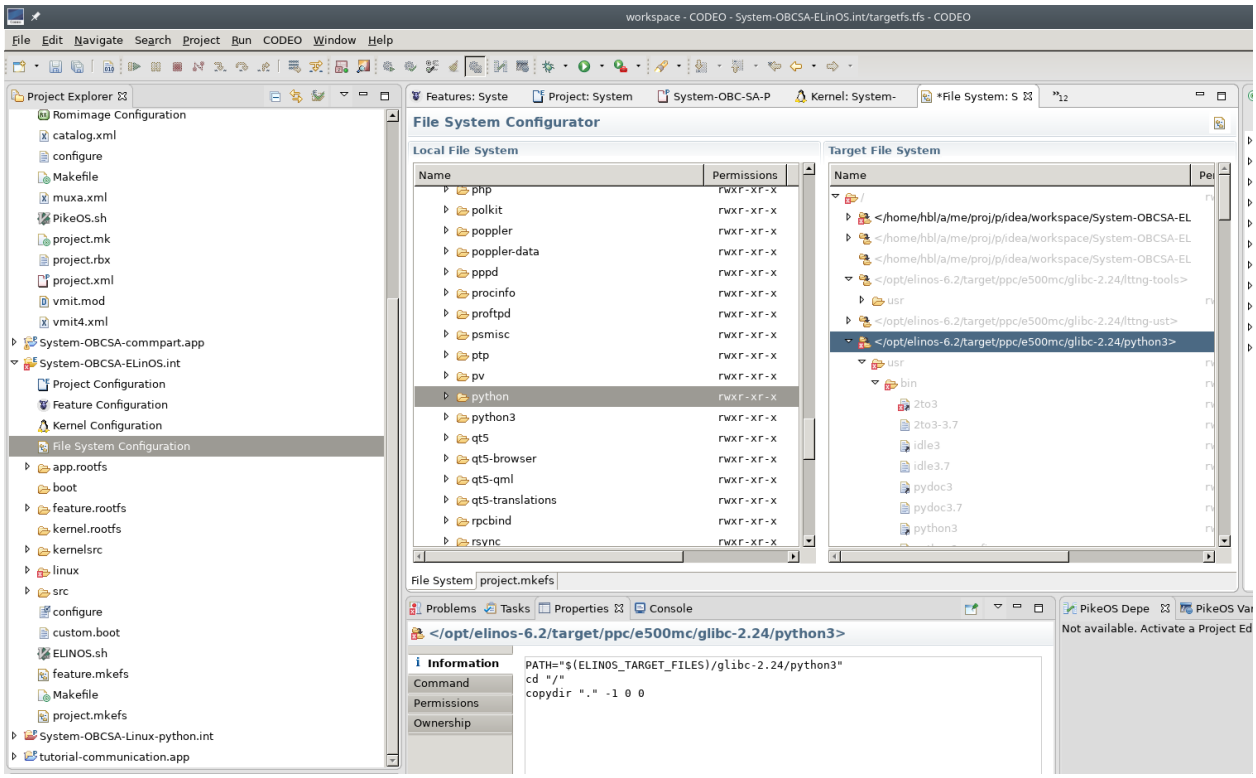


Abbildung 16: Installation von Python 3 für ELinOS 6.2 über File System Configuration

Abbildung 17 und Abbildung 18 zeigen die erfolgreiche Ausführung von Python 3 auf einem Embedded-Target P4080DS auf ELinOS und PikeOS (Labor Mainz).

Abbildung 17: Ausführung von Python 3 auf Avionik-PowerPC P4080DS (Lab Mainz): Python-Konsole

```
Linux version 4.9.231-ELinOS-1825-rt149 (hbl@hbl-laptop) (gcc version 6
.3.0 20170516 (GCC) ) #3 Mon Aug 2 10:33:43 CEST 2021
bootconsole [sswcon0] enabled
dtb= parameter not provided, the kernel may fail to boot
On node 0 totalpages: 77856
free_area_init_node: node 0, pgdat 00491374, node_mem_map 12db5000
  DMA zone: 609 pages used for memmap
  DMA zone: 0 pages reserved
  DMA zone: 77856 pages, LIFO batch:15
pcpu-alloc: s0 r0 d32768 u32768 alloc=1*32768
pcpu-alloc: [0] 0
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 77
247
Kernel command line: params=rfs:linux.params initrd=auto earlyprintk vm
file_net=0,eth0:/1 fpeth=3:eth1:/1’ console=tty0 console=ttyFP0,muxa:
```

```
/linux
loading initial ramdisk from file rfs:/linux.initrd, size 18551786 byte
s
PID hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Sorting __ex_table...
Memory: 285404K/311424K available (3420K kernel code, 253K rwdata, 692K
 rodata, 188K init, 278K bss, 26020K reserved, 0K cma-reserved)
vmalloc area: 0x13320000 - 0x80000000
PikeOS memory usage: 2035732k total, 443720k used, 1592012k free
SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
NR_IRQS:640
Info: DTB does not contain sysgo,p4int definition
clocksource: p4_timer: mask: 0xffffffffffffffff max_cycles: 0x1cd42e4d
ffb, max_idle_ns: 881590591483 ns
Timer IRQ 512 for CPU#0 started
Console: colour dummy device 80x25
standard input
```

Abbildung 18: Ausführung auf Avionik-PowerPC P4080DS (Lab Mainz): DMESG zeigt den Kernelringspreicher

Abbildung 19 zeigt Initialisierung und Test von pip (Package Installer for Python) auf dem Target.

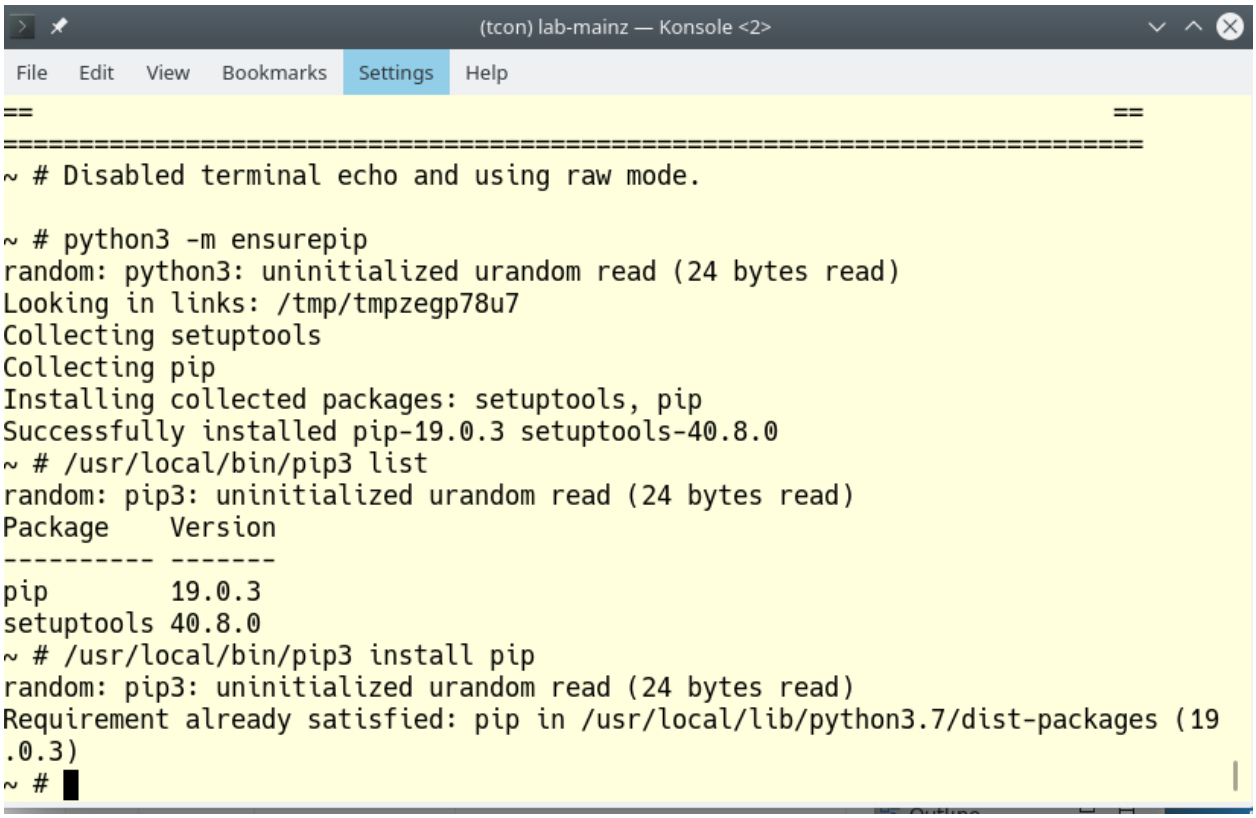


Abbildung 19: Aufsetzen von pip (Package Installer for Python)

2.3.2 Zertifizierbare POSIX API

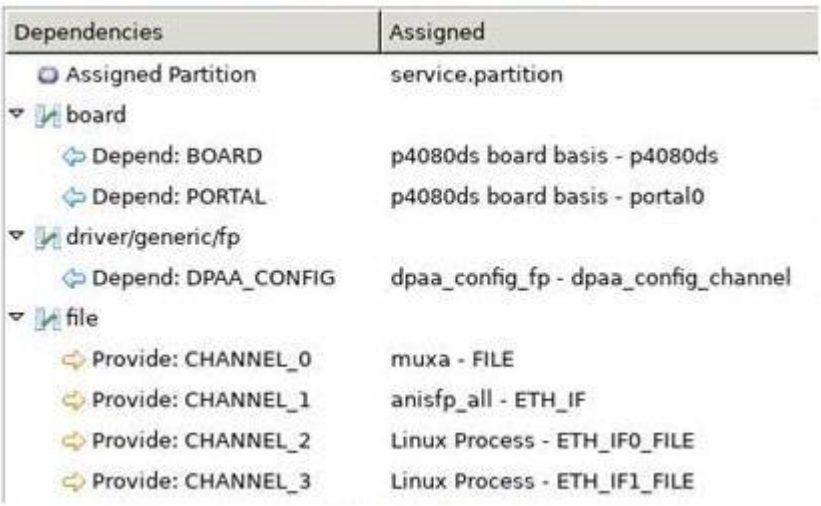


Abbildung 2: Partitionierung DPAA2-Treiber auf ELinOS

Mit dem Ziel einer kleinen zertifizieren Codebasis haben wir zudem Untersuchungen gemacht zu einer minimalen POSIX-API. Bei den POSIX real-time Profilen bieten sich für kleine Safety-kritische Systeme in der Avionik und im Verkehrswesen PSE51 und PSE52 an (Abbildung 3).

PSE51 real-time profiles IEEE 1003.13

https://www.opengroup.org/austin/papers/wp-apis.txt
Only one process but multiple threads:
4.3.1 Minimal Real-time System Profile IEEE Std 1003.13 PSE51
- with a single multi-threaded process
- no file system, no user and group support.

4.3.2 Real-time Controller Profile IEEE Std 1003.13 PSE52
- single multi-threaded process.
- Files and directories are supported, there is no user and group support.

4.3.3 Dedicated Real-time Profile IEEE Std 1003.13 PSE53
- This profile is an extension of the minimal real-time system profile with the addition of support for multiple processes.
- There is no file system since 2003 version
- no user and group support.

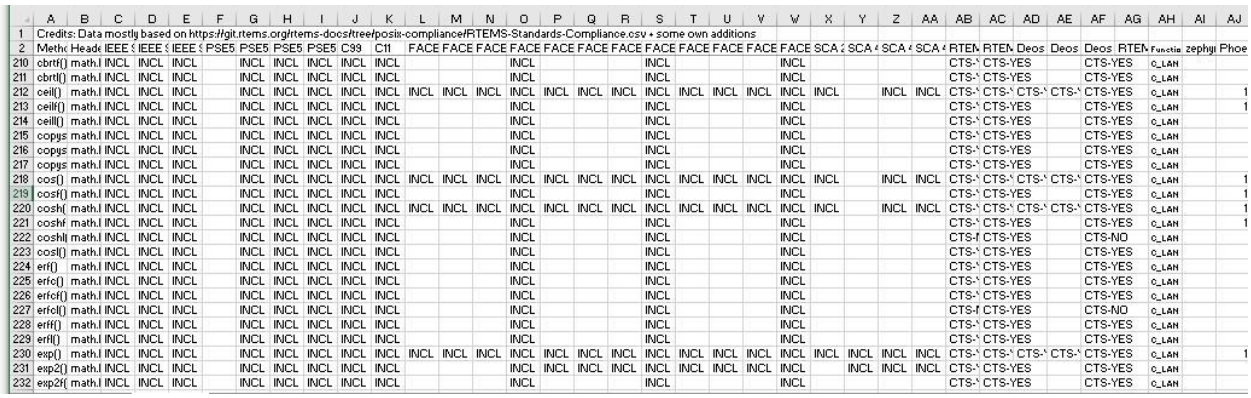
4.3.4 Multi-Purpose Real-time Profile IEEE Std 1003.13 PSE54
- Interactive users and groups

Abbildung 3: POSIX Subprofile

Zusätzlich haben wir mehrere Standards und Implementierungen von Echtzeit-/Betriebssystemen wie

RTEMS, zephyr, Phoenix Systems, freertos sowie SCA application profile[1] durchgesehen. Als

Ausgangspunkt haben wir zunächst Daten von RTEMS[2] verwendet und diese dann durch öffentliche Daten zu den APIs von zephyr, Phoenix Systems, freertos sowie SCA application profile ergänzt (Abbildung 4). Das Gesamtbild zeigt, dass neben der POSIX-Profilunterstützung die meisten RTOS die Funktionsprofile recht selektiv unterstützen.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ
1	Credits: Data mostly based on https://git.rtems.org/rtems-docs/tree/posix-compliance/RTEMS-Standards-Compliance.csv + some own additions																																			
2	Meth	Heade	IEEE	IEEE	IEEE	PSE5	PSE5	PSE5	PSE5	C99	C11	FACE	FACE	FACE	FACE	FACE	FACE	FACE	FACE	FACE	FACE	FACE	FACE	SCA	SCA	SCA	SCA	RTEM	RTEM	Deos	Deos	Deos	RTEM	Functio	zephyr	Phoe
210	cbrtf()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
211	cbrtl()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
212	ceil()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL		INCL	INCL	CTS-Y	CTS-Y	CTS-Y	CTS-Y	CTS-Y	CTS-YES		c_LAN		1
213	ceilf()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		1
214	ceill()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
215	copys	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
216	copys	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
217	copys	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
218	cos()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL		INCL	INCL	CTS-Y	CTS-Y	CTS-Y	CTS-Y	CTS-Y	CTS-YES		c_LAN		1
219	cosf()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		1
220	cosh(	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL		INCL	INCL	CTS-Y	CTS-Y	CTS-Y	CTS-Y	CTS-Y	CTS-YES		c_LAN		1
221	coshf	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		1
222	coshl	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-I	CTS-YES			CTS-NO		c_LAN		
223	cosl()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
224	erf()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
225	erfc()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
226	erfcf()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
227	erfcl()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-I	CTS-YES			CTS-NO		c_LAN		
228	erff()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
229	erfl()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		
230	exp()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	CTS-Y	CTS-Y	CTS-Y	CTS-Y	CTS-Y	CTS-YES		c_LAN		1
231	exp2()	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL	INCL		INCL	INCL	INCL	CTS-Y	CTS-YES			CTS-YES		c_LAN		
232	exp2f(	math.h	INCL	INCL	INCL		INCL	INCL	INCL	INCL	INCL				INCL				INCL				INCL					CTS-Y	CTS-YES			CTS-YES		c_LAN		

Abbildung 4: Untersuchung API (Ausschnitt)[3]

Um die Arbeit zu vereinfachen, haben wir die ASI-Lizenz auf Floating-Lizenz umgestellt, so dass sie innerhalb der ASI-Virtualisierungsumgebung genutzt werden kann.

Weiterhin haben wir ASI unterstützt beim Aufsetzen virtueller Netzwerke, dem Aufsetzen von Chardevices zwischen ELinOS und PikeOS und bei der Integration der Erzeugung der Boot Dateien für die Targets P4080DS und PowerPC-qemu in der ASI eigenen Bauinfrastruktur.

[1] https://media.defense.gov/2020/Feb/13/2002249009/-1/-1/1/SCA_4.1_APP_B_SCAAPPLICATIONENVIRONMENTPROFILES.PDF

[2] https://git.rtems.org/rtems-docs/tree/posix-compliance/RTEMS-Standards-Compliance.csv

[3] https://svn.sysgo.com/viewvc/17132/trunk/tech/api-compliance/

2.4 TP4 Querschnittsthemen

Es wurden relevante Standards & Guidelines identifiziert und analysiert. Hierbei wurde der „state of the art“ aus der Avionik sowie Common Criteria im Detail analysiert.

SYSGO hat diesbezüglich die DO-356 und DO-356A analysiert. Diese wurden in einem angenommenen Vortrag auf CTIC (https://blog.sysgo.com/join-us-at-the-certification-together-in-toulouse) eingebracht. Wir haben uns vertieft mit relevanten Standards wie CAST-32A beschäftigt, sowohl auf Architekturebene und wie diese geeignet mit Tools analysiert werden können (z.B. Gespräche mit Hersteller von Coveragetools). Darüber hinaus haben wir die Standards der Multicore Association, MTAPI (Multicore Task API), MRAPI (Multicore Resource API, z.B. Shared Memory), MCAPI (Multicore Communication API), SHIM Software Hardware Interface for Multi-/Manycore angefordert und erhalten und diese in Hinblick auf Multicoreanforderungen analysiert. Des Weiteren wurde ein Webinar zu Verwendung von DO-356A erstellt und gegeben. -> https://www.youtube.com/watch?v=1uTs3Es6ixU Eine Gemeinsamkeit ist hier, dass Top-Down-Analysen gewählt werden. Auch war unser PikeOS-Security-Target durchaus hilfreich dabei, Assets zu identifizieren. Anders als bei der Analyse von Applikationssoftware können verbreitete Analysemethoden wie Kemmerers Shared-Ressource-Matrix oder einfache Taint-Analyse nicht abbilden, da es bei Systemsoftware viele Fälle von Ressourcenwiederverwertung gibt, die an sich auch gewollt sind, da sie abgesichert werden können wie z.B. Register beim Kontextswitch.

2.4.1 AC20-193/AMC20-193

Wir haben eine Analyse des Updates AC20-193/AMC20-193 des Avionikstandards CAST-32A für Multicore (z.B. in Hinblick auf Hardware-/Softwareressourcen) gemacht (Abbildung 20).

AC20-193 Objective	Text	Resources with explicit mention	Provider of platforms w. robust partitioning	User of platform w. robust partitioning	Notes	
	7. → Identify the methods and tools to be used to develop and verify all the individual software components hosted on the MCP so as to meet the objectives of this document and comply with the applicable software guidance, including any methods or tools needed due to the use of an MCP or the selected MCP architecture.					
MCP_Planning_2 (IDAL A, B, C)	The applicant's plans or other deliverable documents: ¶ 1. → Provide a high-level description of how MCP shared resources will be used and how the applicant intends to allocate and verify the use of shared resources (*) so as to avoid or mitigate the effects of contention for MCP resources and to prevent the resource capabilities of the MCP from being exceeded by the demands from the software applications and/or the hardware components of the MCP. ¶ 2. → Identify the MCP hardware resources to be used to support the objectives in this	Shared resources, hardware resources (shared caches, shared memory, interconnect), dynamic features (if any) such as energy saving, safety net (if any)	Provide guidance on how to configure shared resources, how to control dynamic features, setup safety net	Understand guidance on how to configure shared resources, dynamic features, safety net	Identification of shared resources, dynamic features, safety net	

Abbildung 20: Analyse AC20-193/AMC20-193: Beispiel MCP_Planning_2 (Auszug)

2.4.2 Analyse geteilter Ressourcen

Multicore-Architekturen werden in der Avionik zunehmend wichtig, und hierzu ist die CAST-32A und AC 20-193-Guidance anzuwenden, die wir bereits analysiert haben und ein Schwerpunkt liegt in der Analyse geteilter Ressourcen.

Hier haben wir uns nun darauf konzentriert, einen einfachen Ansatz zur Identifikation von geteilten Ressourcen zu machen. In IDEA hatte zuvor ein IDEA-Projektpartner eine Graphendatenbank (Neo4j[4]) zur Analyse von Anforderungen verwendet. Eine derartige Graphdatenbank ist zum einen auf Performanz bei Analyse komplexer Grafen optimiert und andererseits bietet sie eine Beschreibungssprache die für hierarchische Strukturen geeignet ist. Die Idee der Verwendung einer Graphdatenbank haben wir in dieser Arbeit aufgegriffen, um ein Framework herzustellen um geteilte Ressourcen identifizieren zu können. Hiermit können die mögliche Quellen von Interferenz in sicherheitskritischen Anwendungen ermittelt werden.

Dazu haben wir die Konfigurationsdateien von SYSGOs PikeOS-Integrationsprojekten (vmit4.xml) umgewandelt mit XSLT Stylesheets und dem Saxon-B XSLT-2.0-Prozessor zu Definitionsfiles für Daten in Neo4j, wie wir im Folgenden am Beispiel FileAccess zeigen (Abbildung 6). Eine Lehre dabei war, dass eine optimale Darstellung der Ressourcenzugriffe mehrere Iterationen benötigt, um Duplikate in der Graphdarstellung zu vermeiden.

[4] https://neo4j.com

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:p4="http://www.sysgo.com/xsd/p4/vmit-4.5.xsd"
>
<xsl:output method="text" indent="no" omit-xml-declaration="yes" />
<xsl:strip-space elements="*"/>
<xsl:key name="filenames" match="//p4:FileAccess" use="@FileName"/>

<xsl:template match="/">
CREATE
<xsl:for-each select="//p4:FileAccess[count(. | key('filenames', @FileName)[1]) =
1]">
(File<xsl:value-of select="translate(@FileName, ':.-/*', '____')"/>:File {FileName
:'<xsl:value-of select="translate(@FileName, ':.-/*', '____')"/>'}),
</xsl:for-each>
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="p4:Partition" priority="1">
(Partition<xsl:value-of select="@Identifier"/>:Partition {Identifier:'<xsl:value-o
f select="@Identifier"/>'}),
     <xsl:apply-templates/>
</xsl:template>

<xsl:template match="p4:FileAccess" priority="1">
(Partition<xsl:value-of select="../../@Identifier"/>)-[:R]->(File<xsl:value-of sel
ect="translate(@FileName,':.-/*','____')"/>),
     <xsl:apply-templates/>
</xsl:template>
</xsl:stylesheet>
```

Abbildung 6: XSLT Stylesheet vmit4.xml nach Neo4j: Regeln für FileAccess

Nachdem Daten in Neo4j eingepflegt sind, kann man die durch eine spezifische Konfiguration erzeugten Abhängigkeiten als Graphen darstellen. Abbildung 7 zeigt wie 3 Partitionen eines FileProviderIntegrationsprojektes Ressourcenzugriff (hier: FileAccess-Zugriffe) teilen, dabei gemeinsam auf einen Konsolentreiber („con_“) zugreifen.

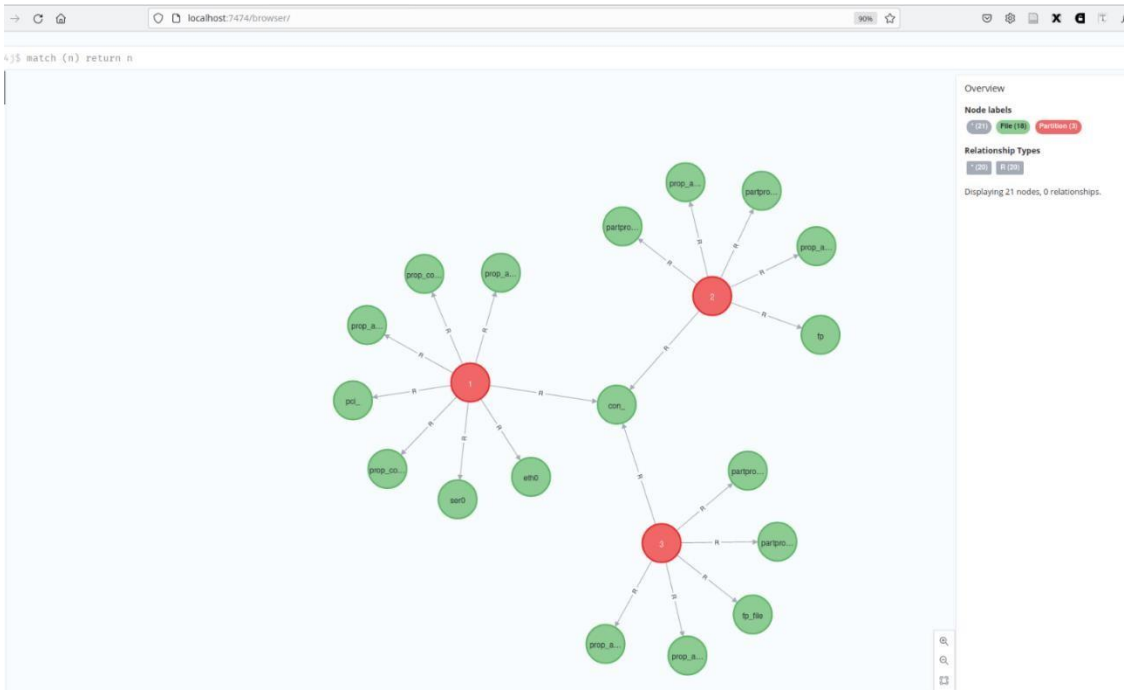


Abbildung 7: File-Nutzung durch Partitionen: Shared Konsole im Integrationsprojekt „File Provider“

In Neo4J können verbundene Partition mit der einer Match-Anfrage abgefragt werden, und es werden alle drei Partitionen zurückgegeben (Abbildung 8).

Abbildung 8: Verbundene Partitionen

Wenn wir beispielsweise im Datenbankinterface (oder durch eine entsprechend geänderte Konfiguration) diese Shared-Konsole von Partition 1 entfernen, dann sind erwartungsgemäß nur noch zwei der Partitionen verbunden (Abbildung 9). Das zeigt, dass die Graphdatenbank zur Darstellung von Interferenzen und Interferenzfreiheit durch Analyse der Konfigurationsdateien verwendet werden kann.

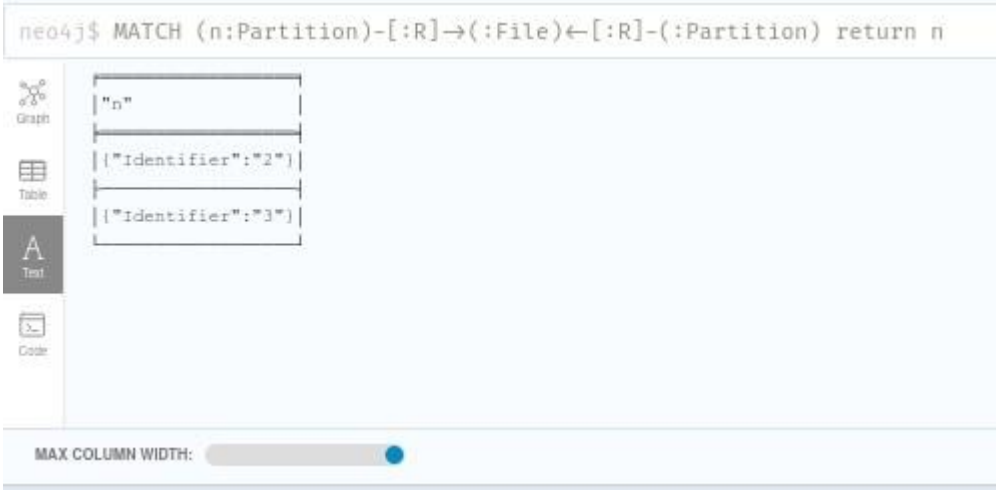


Abbildung 9: Entfernung der Verbindungen von Partition 1

3 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Die Arbeit war *notwendig*, weil sich bei bisherigen Avionikentwicklungen bei SYSGO zwar eine große Erfahrung gesammelt hat, diese aber bisher in kleinem Rahmen mit Partnern diskutiert wurden konnten (SYSGO agierte hier in der Regel als Auftragnehmer, der zwischen Anforderungen der Hardware und den Anforderungen eines jeweils sehr spezifischen Gesamtsystems vermitteln muss). MZB-IDEA erlaubte es SYSGO, in der (offeneren) Kommunikation eines Forschungsprojekts die Produktfertigungskette speziell unter Berücksichtigung von MILS an Partneranforderungen anzupassen.

Konkret wurden einerseits (1) Entwicklungen vorgenommen, die neuartige Fragestellungen in fortgeschrittenen Avionik- und Aerospacesystemen (Memory-Partitionierung und gemischt-kritische Systeme mit mehreren Partitionen sowie (2) Entwicklungs- und Modellierungsansätze zu verbinden, und hier verschiedenen formales Ausdrucksmöglichkeiten zu erforschen. Die Arbeiten zur Integration der Konfigurationsumgebung in formale Modelle wurden im Projekt Admorph aufgegriffen, um Partitionsanalyse für GraphML zu machen.

Die SYSGO ist durch Abschluss dieses Projektes besser vorbereitet, den Luft- und Raumfahrt Markt gezielt zu adressieren und sieht hier langfristiges Wachstumspotential. Dies wird die langfristige Steigerung von Umsatz und Ertrag sowie die Sicherung vorhandener Arbeitsplätze bei der SYSGO in Deutschland garantieren.

4 Fortschritt auf dem Gebiet des Vorhabens bei anderen Stellen

Es sind hinsichtlich des Teilvorhabens keine Fortschritte bei anderen Stellen bekannt.

5 Erfolgte oder geplante Veröffentlichungen

- Oliver Kühlert, Mark Tootell, 2020, Compliance of the Common Criteria Evaluation with the RTCA DO-356A Standard, Talk at Toulouse “Certification Together” https://www.sysgo.com/blog/article/join-us-at-the-certification-together-in-toulouse -> https://www.youtube.com/watch?v=1uTs3Es6ixU
- Bartakovic, D., Söding-Freiherr von Blomberg, A., Kühlert, O., Jukić, T., Fumaroli, G., Herpel, H.-J., Lopez Cueva, P., Carranza, J.-M., & Guy, Maxime. (2022). Use of Multiple Independent Levels of Security and Safety (MILS) architecture for space on memory protection unit (MPU)-based systems. DASIA.

Mario Brotz, Holger Blasum