

Whitepaper – Cross-Language DLL-Architectuur voor Rocket COBOL, Intel C, Fortran en ML64

Whitepaper - Cross-Language DLL-Architectuur

Een schaalbaar, onderhoudsarm en high-performance ontwerp voor COBOL-, C-, Fortran- en ML64-ecosystemen

Executive Summary

Deze whitepaper beschrijft een moderne, duurzame en high-performance architectuur voor multi-language softwareontwikkeling waarin **Rocket COBOL 11.0, Intel C, IFX Fortran en ML64-assembly** samenwerken via één centrale **DLL-kern**. De aanpak elimineert code-duplicatie, voorkomt inconsistenties, verhoogt performance en maakt het systeem toekomstvast. De kern van de architectuur is een **strikt C-ABI-gebaseerde DLL-laag** die fungeert als stabiel contract tussen alle talen. COBOL roept de DLL aan via eenvoudige CALL-interfaces, terwijl C, Fortran en ML64 de interne implementatie verzorgen.

1. Inleiding

Moderne systemen bestaan vaak uit meerdere programmeertalen, elk met hun eigen sterke punten. COBOL blinkt uit in bedrijfslogica en stabiliteit, C in systeeminteractie, Fortran in numerieke performance en ML64 in pure snelheid. Zonder duidelijke architectuur leidt dit tot versnippering, duplicatie en onderhoudsproblemen.

Deze whitepaper presenteert een **gestandaardiseerde interoperabiliteitsarchitectuur** die deze talen samenbrengt in één coherent systeem.

2. Architectuurprincipes

2.1 Single Binary Source of Truth

Alle kernlogica wordt gecentraliseerd in één DLL. Dit voorkomt duplicatie en inconsistent gedrag tussen talen.

2.2 Strict C-ABI Contract

De DLL-interface gebruikt uitsluitend C-compatibele types en calling conventions. Hierdoor kunnen alle talen de functies betrouwbaar aanroepen.

2.3 Language Isolation

Elke taal heeft een dunne wrapper. De logica zit uitsluitend in de DLL-kern.

2.4 Replaceable Core

De DLL kan worden vervangen zonder wijzigingen in COBOL-, C- of Fortran-code, zolang de exports stabiel blijven.

3. High-Level Architectuur

```
COBOL Program
|
| CALL USING
v
C Wrapper (extern "C")
|
| normalized parameters
v
DLL Core (C)
|
| dispatch
v
Fortran Routine ----> ML64 Fastpath (optioneel)
|
v
COBOL Array (COMP-2)
```

4. Interface-Contract

4.1 Exportregels

- Geen name-mangling
- Geen C++-decoratie
- Exports gedefinieerd in .def-file

Voorbeeld:

```
EXPORTS
FILL_VALUES
SCALE
core_get_version
```

4.2 Type-Mapping

Concept	COBOL	C	Fortran	ML64
Int32	COMP-5 S9(9)	int32_t	integer(c_int)	eax
Int64	COMP-5 S9(18)	int64_t	integer(c_long_long)	rax
Double	COMP-2	double	real(c_double)	xmm0
Pointer	COMP-5 64-bit	void*	type(c_ptr)	rax

5. Memory-Architectuur

5.1 Ownership-model

- DLL beheert eigen geheugen
- Callers beheren inputbuffers

5.2 Vrijgavefunctie

```
void core_free(void* p);
```

6. Error-Architectuur

6.1 Error-codes

Code	Betekenis
0	OK
1	Invalid argument
2	Null pointer
3	Out of range
10	Memory allocation failure
100-199	Internal errors

6.2 COBOL-mapping

COBOL vertaalt error-codes naar statusvelden en logging.

7. Logging-Architectuur

```
void core_set_logger(void (*logger)(int level, const char* msg));
```

Log-levels: Fatal, Error, Warning, Info, Debug.

8. Build-Pipeline

8.1 Build-stappen

1. ML64 → OBJ
2. Fortran → OBJ
3. C → OBJ
4. Link naar DLL
5. ABI-check
6. Unit tests
7. Publicatie van artefacten

8.2 Artefacten

- DLL
 - LIB
 - Header
 - Versie-info
 - Testset
-

9. Teststrategie

9.1 Unit tests

- C-laag: ABI-validatie
- Fortran-laag: numerieke stabiliteit
- COBOL-laag: interface-validatie

9.2 Integration tests

- Multi-language scenario's
 - Memory leak detection
 - Stress-tests
-

10. Use-Case: COBOL roept Fortran via DLL

10.1 Fortran-routine

```
subroutine FILL_VALUES(array, length, factor, err) bind(C)
```

10.2 COBOL-CALL

```
CALL "FILL_VALUES"  
USING WS-VALUE WS-LENGTH WS-FACTOR WS-ERR-CODE
```

11. Uitbreidingsmogelijkheden

- AVX2/AVX-512 fastpaths
 - Multi-DLL architectuur
 - Async-interfaces
 - Vectorized numerieke kernels
-

12. Conclusie

Deze whitepaper biedt een volledig, toekomstvast en professioneel raamwerk voor multi-language interoperabiliteit. De DLL-kern vormt een robuuste basis waarop COBOL, C, Fortran en ML64 efficiënt kunnen samenwerken zonder duplicatie van logica.

Het resultaat is een systeem dat:

- sneller is
- eenvoudiger te onderhouden
- schaalbaar blijft
- en klaar is voor toekomstige uitbreidingen.