DDC-I

TECH NOTE

Minimizing Change Impact:

Deos Reusable Software Components

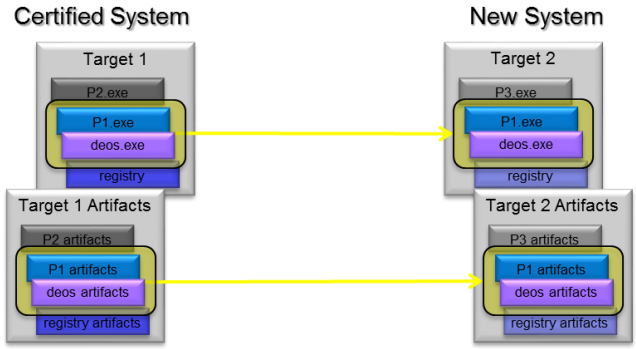
Developing DO-178 certifiable avionics software is an expensive and time consuming process. Software changes then typically require reverification and retesting for the executable being recompiled. Recognizing this, Deos architects designed Deos over twenty years ago such that change impact could be isolated to a small module being modified rather than a complete monolithic executable. To implement this modularity, Deos contains a DAL A linking loader; and the operating system is developed using reusable software components. Deos customers can then save on certification and minimize the time required to obtain a certification by employing the same reusable software component strategy that DDC-I uses to build Deos.

Key Features Overview

- **Reusable Software Components**
  - **Greatly Reduces Certification Costs and Schedule**
- **Binary Modularity**
  - **Enables Reuse of Software & Certification Artifacts**
- **Deos Certification Baselines**
  - **A Single Line of On-Going DAL-A Development for Over 20 Years**

Reusable Software Components

Deos is designed to enable application software component portability, where binary (not just source code) reuse is the ultimate form of portability. Deos was originally developed for use in the aerospace industry where verification and certification costs are notoriously high. The ability to reuse executables and shared libraries without modification on new (compatible) target systems significantly reduces costs. Reuse of software is not a new concept but in the avionics market it is can provide even more value, especially if software modules (executables and shared libraries) do not change from product to product or from one aircraft type to another.



With Deos, binary modularity and reuse is accomplished with the following key technologies:

- DO-178C, Level A, dynamic loader/linker
- Well controlled software component interfaces
- Component based documentation and XML configuration files
- No artificial constraints on resources
  - No magic numbers in code
  - API parameters in **XML configuration files, not in the code**
- Data abstraction mechanism, provided via a certifiable publish/subscribe Data Distribution Service, Deos I/O infrastructure (IOI).

Binary Re-use and Dynamic Linking at Run-Time

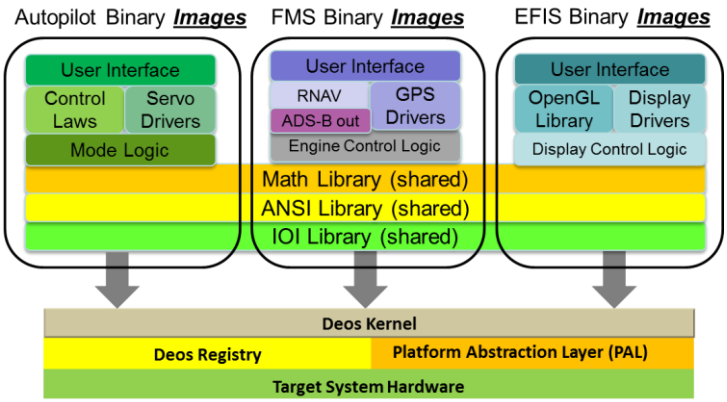
A key differentiator is the Deos DAL-A dynamic loader/linker. This is a critical element in enabling binary modularity and the reuse of DO-178 verification evidence. Deos software applications as well as the Deos operating system itself are made up of individual executables and run-time linkable libraries. Each is separately compiled and linked into stand-alone executable files. At boot time,

the Deos DAL-A kernel loads executables and shared libraries into a protected virtual address space and dynamically links them as part of application start-up (dynamic linking). This process enables a software system to be decomposed into a collection of individual software components with the following characteristics:

- Each executable and run-time library has its own individual digital signature (for integrity checking).
- Dynamic linking the executables & libraries does not change the digital signature of the executable or the run-time library.

The value of this methodology is that the executables on the system are not dependent on an unqualified static linker which injects the library object code into the executable image, breaking its digital signature, and necessitating re-verification. Or worse, a qualified linker that leaves you tied to a specific version of tools.

- **The APIs are the connections**
- **Cyclic Redundancy Check (CRC) is the protective casing**



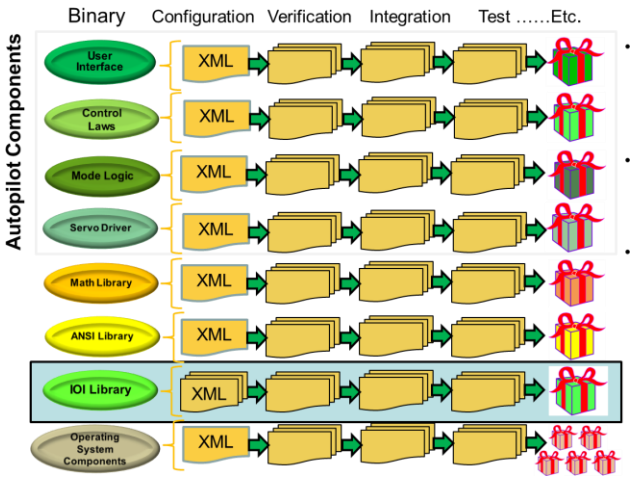
- **The Solution: Binary Partitioning Isolates Change Impact & Maximizes Reuse**
- **Binary Reusable Software Components is the Software Equivalent of a Black Box**

In short, the Deos kernel is a DO-178C Level A verified linker that can be trusted. It connects together the interfaces leaving the binary partitions integrity and pedigree intact. It does this at run-time, instead of establishing the coupling at build time and performs this service independent of linker version, switches, or language constructs used. All of this reduces the impact to change that occurs throughout a program, easing software integration & field updates.

By decomposing a software system into discrete components, one can limit component couplings to well defined interfaces. This provides the advantage of limiting interaction between software components, and thus minimizes the change impact to one when another is altered. A well-controlled inter-component coupling helps create a much smaller control and data coupling issue than if all the source code were in one large executable binary.

An Operating System Built for Portability

Deos itself is composed of a set of individually verified building blocks, each with their own artifacts, executable and CRC. Each operating system component then travels with its own artifacts (design documents and test code), executable, CRC and XML. DDC-I can reuse the executable components from previous certifications as a previously verified component in the next verification. This applies to the file system, the kernel, math libraries, ANSI libraries, Deos' Data Distribution Service (its I/O infrastructure (IOI)), etc. Deos' IOI capability provides data abstraction and is a key feature of reuse (detailed paper available from DDC-I). The overall system is then verified using a combination of previously verified components and modified components (typically the BSP will be a new component to the verification). DDC-I can verify Deos systems faster and with significantly less expense because the operating system was designed to minimize change impact.



Isolation from Processor Specifics – Platform Abstraction Layer

Deos is a true microkernel. As such, the amount of code running in kernel space is at a minimum. This in turn restricts the number of potential safety and security vulnerabilities.

The Deos kernel which executes on a NXP LS1043A card for example is the exact same binary with the exact same DAL A artifacts that executes on an i.MX8 since they are both Arm architectures. The change between the architectures is isolated to the Platform Abstraction Layer (which is a different component than the kernel). The component architecture of Deos is also why it is easy for DDC-I Deos customers to build their own BSPs and drivers if desired. Those BSPs and drivers are not linked into the kernel architecture and don't affect the kernel. Other RTOS vendors sometimes require they be contracted for BSPs and drivers as their RTOS has hardware and system configuration dependencies – which can raise costs and limits their market scalability. Whereas with Deos, DDC-I openly offers training and support for customer to create their own drivers and BSPs as, like everything else, they can be isolated from the rest of a Deos-based system. This helps reduce costs, enables parallel development, allows for faster development phases, and empowers customers to have more control over their system engineering.

Deos includes a platform abstraction library (PAL), which is a dynamically linked library which provides the interfaces to hardware devices needed by the kernel. These devices include memory as well as interrupt controllers and timers that the kernel utilizes for basic tasks such as scheduling. The interface between the Deos kernel and the PAL is kept strictly functional. That is, the kernel and the PAL each export a set of functions for the other's use. This permits the Deos kernel binary to remain unchanged from one platform to the next as long as both platforms are based on the same processor family.

Deos's reuse strategy is based on the idea that the Deos kernel dynamically links the libraries required for itself and the applications at startup. The PAL may also have other libraries that link with it to add access in the kernel address space for devices drivers. This type of library is called a platform resource library (PRL). It is loaded during Deos cold start into the kernel address space just like the PAL. PRLs are typically used for device drivers for devices that require supervisor access to setup or drivers that use special hardware interfaces such as PCIe or DMA since these drivers need to be handled with utmost care. This approach enables the splitting of the driver into separate parts. Only the part of the driver that needs supervisor access is done in the PRL, the rest of the driver is in user space written to the DAL level required by the applications using this interface. This allows one to minimize the amount of code in the PRL that has to be verified to at the highest system DAL level, lowering cost, risk, etc.

Conclusion

Using Deos technology to isolate software components greatly reduces certification costs. By using this technology a change to the defect prone or feature deficient can be made, while leaving other software components identical to the day they were tested and accepted. Deos certification reuse and its ability for users to minimize software impacts is a key differentiator Deos holds relative to other partitioned COTS RTOSs. These features allow mission and safety critical systems to evolve and innovate at a faster rate and lower cost.