Application-Driven Embedded System Design

The majority of processors produced nowadays are targeted at dedicated computing systems that execute either a single application or a small set of previously known applications. In contrast to generic computing systems, these dedicated systems have very specific run-time support requirements, which are not properly fulfilled by general-purpose operating systems. The impossibility to anticipate which applications will be executed results in generic operating systems being forced to provide an extensive set of services targeted at making all resources available to all applications. The standardization of such generic system services locked general-purpose operating system inside a hard shell that prevents innovations from reaching applications. With regard to dedicated computing, these generic operating system provide uncountable services that are not used by individual applications, and yet fail to fulfill application demands.

Within this context, the *Application-Driven Embedded System Design* (ADESD) multiparadigm method proposes a novel strategy to systematically construct application-oriented operating systems as arrangements of adaptable software components. Instead of standard compliance and hardware properties, the features offered by such a system emanate directly from application requirements, thus enabling it to be customized according to the needs of particular applications. Such application-tailored system instances are produced by selecting, configuring, and composing proper components. Even if applications refrain from the new application-oriented services in benefit of standard interfaces, most dedicated applications require such a small subset of those interfaces that mapping them to new system services — instead of porting their traditional implementations — is usually possible.

The ADESD multiparadigm design method guides domain decomposition towards families of scenario-independent system abstractions that can be reused to build a variety of run-time support systems. Environmental dependencies observed during domain decomposition are separately modeled as scenario aspects, which can be transparently applied to system abstractions with the aid of scenario adapters. The assembling of such software components to produce a functioning system is assisted by component frameworks, which capture elements of reusable software architecture identified in the course of domain engineering. Usability is improved by inflated interfaces, which export whole families of abstractions to users as if they were single macrocomponents, passing the responsibility of selecting appropriate family members to the system.

For more information on the ADESD method, please refer to its book.