

Variable architecture strategies and component models for applications on consumer devices

(DRAFT)

Lech Krzanik^{1,2} and Heikki Koivikko¹

¹University of Oulu, Department of Information Processing Science, Linnanmaa, P.O. Box 3000, FIN-90014 Oulu, Finland; Lech.Krzanik@oulu.fi

²CCC Software Professionals Oy, Oulunsalo, Finland

Abstract. Strategies of component-based software development for a number of cases of variable architecture are demonstrated and discussed in the context of applications on consumer devices. Possible implications for the component models are reviewed. Special cases of COTS component systems, architecture evolution, and systems with variable structure, e.g., for limited resource consumer devices, are presented in more detail. Possible tool support and development workflows are discussed and illustrated with examples.

Keywords: Component-based software, development platforms, limited resource consumer devices, software architecture, component integration, cooperative development, non-functional property analysis.

Extended abstract

Conventionally we treat architecture of a software system as something stable. The assumption simplifies, for example, the methodology of component-based development projects. There are however many practical cases when this stability is commonly violated. Examples include COTS systems, architecture evolution, and systems with variable structure. A typical case is a limited-resource-aware consumer device application of variable structure, where selected components should remain valid for a broad range of different architectures matching the distinct resource availability modes. Other typical cases result from a conventional approach to component system development that starts with producing a fixed architectural specification. Such specification often lacks flexibility to match the evolving architecture as user feedback and developer's understanding grow throughout the project. Frequently we deal with COTS components whose architectural and component specifications are incomplete or lacking. The architectural assumptions taken for analysis and evaluation purposes will inevitably change – e.g., in terms of components' architectural models – to match the changing level of understanding of commercial components and communication within the development team.

This work originated in connection with the PAR initiative that runs at the University of Oulu that aims at investigating the issue of variable architecture along with its practical implications on component-based projects. The initiative has been motivated by the growing importance of the issue in the current software development practice and lack of adequate tools suitable for dealing with it. The work proceeds in cooperation with a number of technology development projects including NOMSA-V, MyUI (preparatory stage), BEYOND, AMBIENCE, etc. The initiative addresses specifically applications on consumer devices. NOMSA-V is developing methods and tools for evaluations of architectures and component suites, oriented towards selected domains, such as the domain of opto-electronic measurement systems. MyUI (preparatory stage) investigates component suites for user interfaces in ubiquitous consumer applications, their modeling and simulation. BEYOND provided, among other results, methods and tools for supporting development of advanced

user interfaces taking into account various architectural and component models, and special tasks such as online application measurement. AMBIENCE includes investigation of reusable architectures and components, and re-configurable texture for mobile applications with location awareness. The initiative intentionally cooperates with many relevant projects in order to produce a representative sample of cases in the domain of consumer and mobile devices, and to create a context for development of an adequate solution.

Frequently changing architecture may lead to a state of permanent component architecture mismatch. The initial solutions we propose and discuss in this paper include modifications to the component models, and the emphasis on architecture and component analysis and evaluation. Other investigated methods include modified component-based development workflows, architectural monitoring of components and applications, simulation, architecture re-factoring, cost-effective approaches to architecture and component assessments, etc. A software component is a reusable unit of independent deployment, third-party composition, and versioning, with contractually specified interfaces and explicit context dependencies only. In PAR a component is a packaging of a set of models. The models may refer to any component development or operation stage, and abstraction level, and they may be human or computer readable. In general, there are no mandatory models, however several types are particularly important: (a) a component executable that realizes component functionality, (b) the general model defining an interaction and composition standard such as EJB, CORBA, or COM+, (c) the specific architectural model addressed by the component, (d) component development and runtime architectural evaluations, and (e) references to local or external system implementations including the component in question. Other common types of models address component behavior, nonfunctional properties, quality, tests, simulations, user support, trading, etc. Multiple variants are possible to address various platforms etc. Models (b)-(c) are in practice often specified separately. Models (d)-(e) generate associations between related component uses. The decomposition of the component representation into models allows us to deal with different aspects, also architectural, of components one by one, to facilitate partial architectural evaluations and corresponding partial development steps. The models of a component are interrelated. To keep the structure clear and understandable we use a number of architectural standards and guidelines such as IEEE 1471 or SARA adapted to components. Models (b)-(c) determine the target architecture for which a component has been prepared. In practice these models often remain insufficiently specified. For instance, for systems based on internal components it may be the result of a separate platform specification (usually initiated at the outset of the project) not properly synchronized with component models. For systems based on external components it may be the result of the commercial effects of marketplace. We propose including explicit models (b)-(c), along with a representation of architectural variability. We also propose including models (d)-(e) to facilitate architectural evaluations. The proposals create opportunity for a number of revised component-based development strategies. Cost-effectiveness of such revised strategies is currently being investigated.

The next step in variable architecture component-based strategy synthesis goes deeper into component integration and assumes shifting the responsibilities for matching architecture variability to components so that simplified target reference architecture could be used in application development for consumer devices. Such simplified architecture is referred to as 'parsimonious'. In these setting also dedicated components performing architectural functions should address a range of architectures rather than be linked to particular platforms. This opens way to optimizing the architecture for cost-effectiveness in the context of consumer devices, provided there exist a feasible solution supplied by available component registries. Here architecture becomes independent variable like, on a smaller scale, in some limited-resource consumer device solutions resource consumption becomes independent variable.

We demonstrate a number of cases of component architectural mismatch caused by architecture changes, with tools provided by one of associated projects. Examples of solutions

for nonfunctional property inconsistencies are shown with a component property editor. The corresponding development workflows can be implemented with a workflow manager supporting cooperative component-based development. A property editor may operate as part of a component's visual virtual model or (where virtual representation is not adequate) as part of a decision manager that supports property definition, architectural design, evaluation, alternative consumer product comparison, and delivery planning. The workflows we propose build on previous results and introduce new elements, such as low-cost approximate architectural assessments of components.

Acknowledgments

Projects NOMSA-V and MyUI (preparatory stage) are supported by University of Oulu, Department of Information Processing Science. Projects BEYOND and AMBIENCE are supported by TEKES in the ITEA program.