An Approach to Reference Architecture Design for Different Domains of Embedded Systems

Liliana Dobrica¹ and Eila Niemelä²
¹Faculty of Automation Control and Computers, University Politehnica of Bucharest, Bucharest, Romania
²VTT Technical Research Center of Finland, Oulu, Finland

Abstract - The content of this paper addresses the issues regarding the reference architecture design for different domains in the context of a system of systems that is specific to today's embedded systems. The reference architecture contains core services of the domains included in abstract features package. The appropriate architectural style is provided by a knowledge base through service taxonomy. Services, commonality, variability management and rules for product derivation and configuration are the main issues considered in the architectural design process. Our contribution is the integrated vision based on our experiences and studies of the recent publications.

Keywords: cross domain product line, software architecture, service, quality, embedded systems.

1 Introduction

One of the most challenging engineering areas is embedded systems (ES). Among the requirements and constraints that have to be satisfied we can mention a higher diversity and complexity of systems and components, increased quality, productivity and reuse content, standardization, stricter requirements for time-to-market, fault tolerance and robustness. For a long perspective the design process for ES requires introduction of the higher level abstractions that are blurring the boundaries between hardware and software design. Also the domain technology causes exponential growth of the designed systems. Systems of yesterday become components of today. The fundamental principle stating that “any system consists of components” is common for any technical system as well as for a mature engineering discipline and it is sometimes called “a law of nature” [19]. Embedded systems (ES) are used as subsystems in a variety of domains, e.g. automotive, avionics, health care, industrial control, and consumer electronics. These domains include a variety of functions; however they are composed of a limited number of common software/hardware components. Nowadays in the embedded systems industries it has been recognized a significant duplication of development effort for hardware, software and services [1]. Due to the escalating complexity level of embedded systems, the technology trends and the bigger competition in the world market, a coherent and integrated development strategy for embedded systems is required. It becomes a research priority to create a generic platform and a suite of abstract components with which new developments in different application domains can be engineered with minimal effort [2]. Generic platforms, or reference designs, will be based on a common architectural style that supports the composition of systems out of pre-validated independently developed subsystems that meet the requirements of the different application domains. Given a core architectural style, different components can be created for different specific application domains, while retaining the capability of component reuse across these application domains. In this paper we propose a coherent and integrated development strategy for embedded systems that considers the architecture the main driver. We argue with our experiences in the software architectures design and analysis for embedded systems domains [11, 12] and other researchers’ recent studies that will be revealed during the paper. Our contribution is the synthesis of the most important issues of product line architectures (PLAs) in our development strategy for cross domain architecture design of embedded systems. We propose a service based approach for reference architecture design. The reference architecture is built based on structured existing domains knowledge.

2 Background

2.1 The Software Product Line development

In general the software product line development consists of two stages which are domain engineering and application engineering [4,5]. The main processes and results for each stage are described in Figure 1. Domain engineering can be divided in three main processes: Domain Analysis, Domain Design and Domain Implementation. The domain analysis consists in capturing information and organizing it as a model. Some methods, such as FODA (Feature-Oriented Domain Analysis) [3] propose a set of notations for the domain modeling using the notion of "features" to refer to products properties. The input represents domain knowledge and outputs are domain requirements. The domain design consists of establishing the PLA. The domain implementation realizes the PLA defined during the domain design as software components. The results represent core assets such as, domain requirements, PLA and components. Application
engineering builds products based on the results of domain engineering and users needs. During application analysis of a new system, the requirements from the existing domain model, which matches the user’s needs, are selected. Applications are assembled from the existing reusable components.

![Diagram of Software Product Line development processes.](image)

Variability management is a key issue in the success of product line development. Approaches based on model driven architecture (MDA) [13] or ontology based support [14] have been identified in our researches. Deriving individual products from shared core software assets is still rather time consuming and expensive for a large number of organizations. A method for building product populations with software components has been applied with success in industries [8]. However various studies and experience reports of the current practice in industry [6,7,18] have identified several problems associated with products derivation. Knowledge externalization, variability management and scoping and evolution are three important areas of concerns. Under development are automated approaches [15] or specific tools [17] that are still immature for industrial processes.

### 2.2 Service architecture description and quality

Service architecture is a set of concepts and principles for specification, design, implementation and management of software services [20]. This definition is similar to software architecture that also includes the principles for guiding its design and evolution and has a strong influence over the lifecycle of a system [22]. Service architecture refers mostly to the software architecture of applications and middleware which is the software that is located between applications and the network layer. A middleware layer hides the underlying network environment complexity and masks heterogeneity of platform technologies from applications [21]. A service based approach supports customization and adaptation of services for limited resources, and configurations.

Under a software perspective architectural styles are recurring patterns of system organization whose application results in systems with known, desirable properties. As such, styles are key software design idioms. Examples of well known styles are layered, pipe-and-filter, client-server, push-based, peer-to-peer, and event-based. In practice an architectural style consists of rules and guidelines for the partitioning of a system into subsystems and for the design of the interactions among subsystems. The subsystems must comply with the architectural style to avoid a property mismatch at the interfaces between subsystems. Important contributions have been also identified regarding patterns used in embedded systems development, e.g. object analysis [16], software architecture [36], design of distributed real-time [34,35,37], fault-tolerant telecommunication system [33], real-time design [38], security [39], etc.

A design approach of services at architectural level has to consider quality attributes specification and standards. In IEEE 1061 [27], software quality is defined as a degree of software to process a desired combination of quality attributes. The software quality model [28] defines six categories of characteristics (functionality, reliability, usability, efficiency, maintainability, and portability) that are divided into subcharacteristics, which are externally or internally observable properties of software systems. The quality attributes can be classified into three categories. The first category defines a set of external quality attributes that is observable at runtime, such as performance, functionality, and usability. The second one is the set of internal quality attributes that cannot be discerned at the runtime, such as reusability and integrability. The third one includes quality in use attributes that are dependent on achieving the necessary external and internal quality. All three levels of qualities have to be measured. A Non-functional Framework (NFR) [29] provides a method for managing quality attributes during development by cataloging the types of nonfunctional requirements and their associated concepts and terminology, the development methods that aid in meeting quality requirements, and trade-offs and correlation among quality attributes.

### 3 Our Approach

#### 3.1 Design Description

Our approach is applied to design architecture for various application domains of embedded systems (e.g. automotive, avionics, consumer electronics, control, etc.). Embedded systems applications are doing control. Control activities could be measuring physical variables (sensing), storing data, processing sensors signals and data, influencing physical variables (actuating), monitoring, supervising, enabling manual and automatic operation, etc.

We consider an embedded system domain a collection of cooperating services that deliver required functionality. These
services may be executed in a networked environment and may be recomposed dynamically. We propose an approach that extends to three levels the architecture development method for embedded systems application domains. In Figure 2 the reference architecture includes core services and it focuses on commonality. Domain architecture includes domain specific services and it requires variability management concerns. Finally, the last level is dedicated to the set of products architectures. On this level rules for product derivation and configuration should be included.

**Reference Architecture**
- Core Services
- Commonality

**Domain Architecture**
- Domain Specific Services
- Variability management

**Product Architecture**
- Concrete Services
- Rules for product derivation and configuration

![Figure 2. Cross domain architecture design.](image)

A feature model is a prerequisite of this approach. We propose a feature model as described in Figure 3. This model is essential for both variability management and product derivation, because it describes the requirements in terms of commonality and variability, as well as defines the product line dependencies. Feature types may be mandatory, optional, optional or optional alternative. Dependencies specified by the model are called composition rules between domain features. The requires rule expresses the presence implication of two features and the mutually exclusive rule captures the mutual exclusion constraint on feature combinations.

Reference architecture defines quality attributes, architectural styles and patterns and abstract architectural models (Figure 4). Quality attributes clarify their meaning and importance for core service components. The interest of the quality attributes for the reference architecture is how the quality attribute interacts with and constrains the achievement of other quality attributes (i.e., trade-offs) and what the user’s view of quality (i.e., quality in use) is. Embedded systems services have to meet many quality attributes, such as modifiability and integrability. Modifiability of a service can be divided into the ability to support new features, simplify the functionality of an existing system, adapt to new operating environments, or restructure system services. Integrability measures the ability of the parts of a system to work together and it depends on the external complexity of the components, their interaction mechanisms and protocols, and the degree to which responsibilities have been cleanly partitioned.

Interdependencies and tradeoffs also exist between quality attributes.

![Figure 3. UML Features model.](image)

The styles and patterns are the starting point for architecture development. Architectural styles and patterns are utilized to achieve qualities. The style is determined by a set of component types, the topological layout of the components, a set of semantic constraints and a set of connectors [30]. A style represents a class of architectures and is an abstraction for a set of architectures that meet it. A small taxonomy identifies five style categories: independent components, data-flow-centered, data-centered, virtual machine, and call-and return style [30]. An architectural pattern is a documented description of a style or a set of styles that expresses a fundamental structural organization schema applied to high-level system subdivision, distribution, interaction, and adaptation [31]. Design patterns, on the other hand, are on a lower level. They refine single components and their relationships in a particular context, and idioms describe how to implement particular aspects of components using the given language [32].

In this way the reference architecture creates the framework from which the architecture for new embedded systems is developed. It provides generic architectural services and imposes an architectural style for constraining specific domain services in such a way that the final product is understandable, maintainable, extensible, and can be built cost-effectively. Potential reusability is highest on the reference architecture level. Core services and the architectural style of the reference architecture are completely reused on the domain architecture level.

Reference architecture is build based on a service taxonomy. We adopted the idea from WISA [26] of an existing knowledge on software engineering that is integrated and adapted to service engineering for embedded systems. The standards related to each embedded system domain, applicable architectural styles and patterns and existing concepts services and components are the driving forces of embedded systems development. A service taxonomy defines the main categories called domains. Typical features that have been abstracted...
from requirements characterize services. The purpose of the service taxonomy is to guide the developers on a certain domain and getting assistance in identifying the required supporting services and features of services.

Domain architecture describes ready made building blocks that assist application/products developers in using specific domains services. When the reference architecture has been defined, the existing components and services are considered as building blocks in the architecture of set of products. The domains services provides variable assets repository. Variability appears in functional and non-functional requirements (including quality attributes). A structured domain architecture repository may be provided. A schema for this repository has to be defined in a form of relationships between services. In this way we are mapping domain specific services to core abstract services. Specialization relation is a solution to be used for variability management. Modeling run-time quality attributes variability requires tool support. Approaches [10] are under development and refining. Monitoring mechanisms, measuring techniques and decision models for making tradeoffs should be better defined and validated.

A product architecture consists of concrete services derived and configured based on the rules. The goal of product derivation is to reach a configuration of the product line in which necessary variabilities have been bound. The decision model for binding specific services of a domain to a product may be in a tabular form or a more comprehensive tool based on the feature types and composition rules. By selecting a consistent set of features required for the individual product, the corresponding domain specific services that realize those features are selected from the domain repository to constitute the product.

3.2 Architectural Development

There are several architectural development approaches that can be adopted to service based embedded systems development. The Model-driven Architecture [23] is an approach that guides the specification of information systems. The idea is to separate descriptions of functionality from the implementation specifications and thus increase the integrability and evolvability of a system. Implementation independent descriptions of functionality last longer than implementation specifications that change as soon as a better technology is available. In MDA, a model means a specification of a part of the function, structure and/or behavior of a system. The specification expects either textual or graphical language with strictly defined syntax and semantics.

Another design approach concentrates on multiple views of architecture. An architectural view is a representation of a whole system from a perspective of a related set of concerns [22]. Among these approaches [24,25], there is no agreement on a common set of views or on the way to describe software architecture. The need for different architectural views depends on three issues: the size, the domain and the number of different stakeholders [26]. Although a multiple view approach helps in developing software products, it is easy to introduce errors and inconsistencies in a multiple view model. It is therefore necessary to provide support for consistency checking among the views.

4 Example

4.1 Description

In this section we describe a simple multiple domains example to illustrate our vision for product line architecture development. The cross-domain reference architecture considers measurement control, data acquisition control and data management domains. Our example is abstracted from our experiences with the architecture design of a scientific on-board silicon X-ray array (SIXA) spectrometer control software. SIXA is a multi-element X-ray photon counting spectrometer. It consists of 19 discrete hexagonally-arranged circular elements and specific domain hardware architecture. The SIXA measurement activity consists of observations of time-resolved X-ray spectra for a variety of astronomical objects. Figure 5 introduces the context view of SIXA considering it a measurement controller. External elements that interface with our measurement controller are a command interface and physical devices (detectors) representing sensors and actuators. The system is programmed and operates using a set of commands sent from a command interface.

The role of the spectrometer controller is to control the following measurement modes:
- Energy Spectrum (EGY), which consists of three energy-spectrum observing modes: Energy-Spectrum Mode (ESM), Window Counting Mode (WCM) and Time-Interval Mode (TIM).
- SEC, which consists of three single event characterization observing modes: SEC1, SEC2 and SEC3.

Each measurement mode could be controlled individually. A coordinated control of the analog electronics is required when both measurement modes are on.

The spectrometer controller cross domain architectural approach is described in Figure 7.

Reference architecture. Looking top-down, the Reference Architecture encapsulated in the Measurement <<MultipleDomain>> is composed of three core abstract <<Domain>>s: MeasurementControl, DataAcquisitionControl and DataManagement.

In each core <<Domain>> abstract features are collected. The MeasurementControl is responsible for services of starting and stopping the operating mode for data acquisition according to the commands received from the command interface and according to the events generated in other parts of the software. DataAcquisitionControl services collects events (science data) to the spectra data file during observation of a target. This abstract service includes as well as hides data acquisition details. DataManagement abstract services provides interfaces for storing science data - opening/closing/writing the data files, hiding data storing details and controlling transmission of the stored data to command interface.

Domain architecture. Domain architecture consists of domain specific services and variability management services. Each of the three core services is specialized in domain specific services. For example, MeasurementControl is specialized in Stand Alone Control (SAC) and Coordinated Control (CC), DataAcquisitionControl (DAC) is specialized in SEC_DataAcquisitionControl (SEC_DAC), and DataManagement (DM) is specialized in EGY_DataManagement (EGY_DM) and SEC_DataManagement (SEC_DM). Domain architecture includes services associated to variability management.

4.2 Cross-domain architecture of spectrometer controllers

The spectrometer controller cross domain architectural approach is described in Figure 7.
**Product architecture.** Product architecture for the sets of products includes rules for product derivation and configuration.

<table>
<thead>
<tr>
<th>Product Domains</th>
<th>Specific Service</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurement Control</td>
<td>SAC</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Data Acquisition Control</td>
<td>EGY_DAC</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Data Management</td>
<td>SEC_DAC</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

In Table 1 the set of products are horizontally distributed and the domain services are dispersed vertically. Each cell \( t_{ij} \) of the table is marked if product \( P_j \) uses component \( C_i \). For example, two products, P1 and P2, includes a SAC service of the measurement control domain.

The architecture have been documented around multiple views describing conceptual and concrete levels, for each view a static and dynamic perspective being offered. Architecture documentation addresses specific concerns for measurement control, data acquisition control and data management. The views were illustrated with diagrams expressed in UML-RT, a real-time extension of UML. The conceptual level considered a functional decomposition of the architecture into domains. The relationships between architectural elements are based on pass control and pass data or uses. The concrete level has considered a more detailed functional description, where the main architectural elements are packages, capsules, ports, protocols. The relationships are association, specialization, generalization, etc. Considering the dynamic aspects state-chart diagrams and message-sequence charts are also part of this description level.

### 5 Conclusions

We have proposed an approach to design a cross-domain reference architecture for embedded systems. The approach has been exemplified by a simple example. The problem dimension for the development of an embedded system reference architecture increases due to the larger number of requirements and constraints specified by a group of experts. Building the features model may require a tool in order to manage the analysis and structuring the abstract features in domains. The reference architecture contains core services of the domains included in abstract features package. The appropriate architectural style is provided by a knowledge base through a service taxonomy. A domain architecture repository is a solution for variability management of specific services. The decision support tool is proposed for product derivation. The role of this tool is to bound variabilities in order to get a right configuration for a product architecture. In our example, we used a tabular form for the decision model.

When the problem complexity increases, a more elaborated tool is required and is a subject for our future research. However, our incremental design and analysis approach is based on a systematic service oriented approach, which is more practical, easier to follow and benefits of advantages provided by service engineering.

Future research work is needed to develop systematic ways of bridging other requirements and constraints from a taxonomy to reference architecture. However, this paper presented the main concepts and justified why these concepts are required.

### 6 REFERENCES


