INTEGRATING REUSABLE CONCEPTS INTO A REFERENCE ARCHITECTURE DESIGN OF COMPLEX EMBEDDED SYSTEMS

Liliana Dobrica
University Politehnica of Bucharest, Faculty of Automation and Computers
Spl. Independentei 313, Bucharest, Romania

Keywords: Embedded systems, Software architecture, Reference architecture, Patterns, Quality.

Abstract: The content of this paper addresses the issues regarding integrating reusable concepts for a quality-based design of reference architecture in the context of complexity that is specific to today's embedded control systems. The reference architecture consists of core services and is designed based on considering taxonomy of requirements and constraints, reusable control patterns and a quality-based measurement instrument.

1 INTRODUCTION

Nowadays an embedded system (ES) application represents one of the most challenging development domains. Among the requirements and constraints that have to be satisfied we can mention a higher diversity and complexity of systems and components, increased quality, standardization, fault tolerance and robustness. In the design process an ES requires introduction of the higher level abstractions that are blurring the boundaries between hardware and software design. Due to the escalating complexity level of ESs a coherent and integrated development strategy is required. It becomes a priority the creation of reference architecture (RA) and a suite of abstract components with which new developments in various application domains can be engineered with minimal effort. RA is based on a common architectural style that provides the composition of independently subsystems that meet the requirements of the various application domains. Thus different components can be created for various specific domains, while retaining the capability of component reuse across these domains.

ES complexity resides in a multitude of interdependent elements which must be organized. To handle complexity, an architectural approach helps to consider separation of concerns realized through different levels of abstraction, dynamism and aggregation levels. In the field of control, the knowledge acquired in software engineering is not really exploited, although it helps to manage complexity. Patterns and quality based approach may be used to establish a direct link between the concepts from the field of control and the software architecture concepts. They guide the analysis and synthesis of software components and they can be used to develop complex control architecture. The architecture is comprehensible as it shows the elements necessary for doing a functionality and the manner in which they interact, and it is flexible because it can be adapted to other systems of the same type in the application domain. In the context of control systems the problem is modelling and documenting software architectures reusable knowledge dedicated to control.

In this paper we propose an approach to manage complexity of complex ES based on defining sources of knowledge for RA. Building the RA is based on well known and reusable concepts from software engineering. Our contribution is in the synthesis of the most important issues that can be applied.

2 BACKGROUND

At this moment there is no general consensus about the definition of embedded terms. ESs are subject to limited memory and processing power and many ESs are also real-time systems that have strict performance constraints. Even for non-real time

ESs, developers have to take into account the timeliness, robustness, and safety of the systems. The fact that ESs are embedded, that is they cannot easily be taken out of their environment to be maintained or evolved, poses reliability requirements. Nevertheless it includes subcategories such as embedded domain, reactive domain, control/command domain, intensive data flow computation domain, best-effort services domain (Marte, 2008). Traditionally an ES represents a computer system which is integrated into another system, the embedding system. The requirements for an ES must be derived from the embedding system. There are two different areas. One is when the embedding system is a product and the other is when it is a production system. The fist one includes automotive electronics, avionics, and health care systems and the second one includes manufacturing control, chemical process control, and logistics.

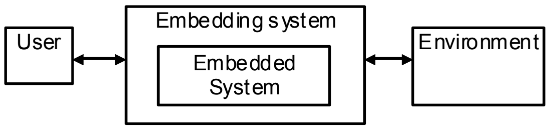

Figure 1: Traditional embedded system model.

ESs are doing control such as measuring physical data (sensing), storing data, processing sensors signals and data, influencing physical variables (actuating), monitoring, supervising, enable manual and automatic operation, etc..

In the embedded world a model driven approach is used to express the requirements in a modeling environment that automatically generates the application code. The well-known example of such an environment is the Matlab tool suite. The increase in efficiency arises from the fact that the software design and implementation phases are automated and the control engineer has not care about the implementation issues as in software engineering processes.

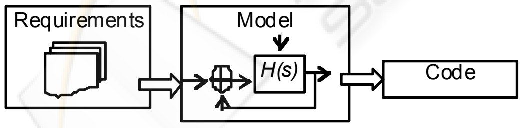

Figure2: Typical model-driven approach.

The problem for control engineering domain is that these applications tend to be multi-domain. A complete control application does not simply cover implementation of control laws. In most cases, the implementation of control laws, the specific domain of Matlab, is only a small fraction of the total control software. Most of the software normally is concerned with various functionalities and Matlab-like tools are inappropriate to cover these functionalities. A new approach is required to deal with the new requirements.

3 PROPOSED APROACH

The design of RA for complex ES is realized with core services which are abstract architectural models and depends on the quality attributes, styles and patterns and others that are shown in Figure 3.

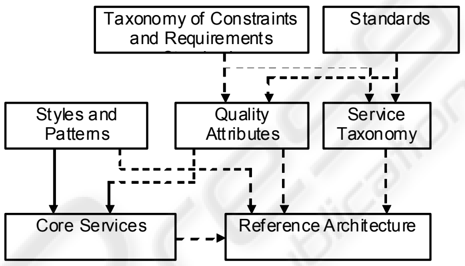

Figure 3: Reference architecture realization.

Quality attributes clarify their meaning and importance for core services. The interest of the quality attributes for the RA is how they interact and constrain each other (i.e., trade-offs) and what the user's view of quality is. 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. A style defines a class of architectures and is an abstraction for a set of architectures that meet it. Design patterns are on a detailed level. They refine single components and their relationships in a particular context.

RA creates the framework from which the architecture of new ESs is developed. It provides generic core 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 RA level. RA is build based on a service taxonomy. A reusable knowledge base is integrated and adapted to service engineering for ESs. The standards related to each ES domain, applicable architectural styles and patterns and existing concepts of services and components are the driving forces of ESs development. A service taxonomy defines the main categories called domains. Typical features that have been abstracted from requirements and constraints characterize services. The service taxonomy guides the developers on a certain domain and getting assistance in identifying the required supporting services and features of services.

4 DISCUSSIONS

A *taxonomy of constraints and requirements* that delimit the design space for a RA for ES is presented in figure 4. Composability refers to the way that larger systems can be composed of smaller subsystems. A system is composable with respect to a certain property if this property is not invalidated by integration. Integration of subsystems that are realized in different technologies are subject heterogeneity. Growth and scalability require if the available resources permits the integration of more subsystems, then the new ones must not disturb the correct operation of the already integrated subsystems. Integration of distributed services must adhere to well established standards.

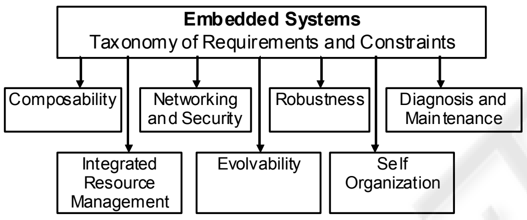
Embedded Systems
Taxonomy of Requirements and Constraints
Composability
Networking and Security
Robustness
Diagnosis and Maintenance
Integrated Resource Management
Evolvability
Self Organization

Figure 4: ESs requirements and constraints.

Networking refers to control loops to be supported at network level. Communication service reliability depends on the application parameters, and protocol standards (Ethernet, USB, CAN, Bluetooth, etc). Integrity mechanisms are required to prevent undetected modification of hardware and software by unauthorized persons or systems, meaning defence against message injections, message replay or message delay on the network. By robustness an ES must handle the increasing failure rate. Fault tolerant mechanisms are used to adapt to reliability changes of subsystems during the ES's life time. Services should be provided for error containment, membership, error detection and error masking. A generic fault-tolerance layer, design for verifiability, formal methods and specification support, software management methods for time, space, and I/O allocations should be considered, too. Diagnosis and maintenance requires a system health monitoring service and a diagnostic service to identify faulty subsystems. The diagnostic service must not interfere with the operation of the subsystems that are to be diagnosed. Predictive maintenance at the architecture level supports the identification of components that are likely to fail in the near future. Design for testability with respect to unit testing, system integration testing, manufacturing testing and assembly testing. Integrated resource management needs dynamic reconfiguration to support changing of the configurations of applications while they are executed. Evolvability is based on uncertainty with respect to application characteristics and technological capabilities. Development of products delivered in multiple variants should be considered. Implementation independence, virtual machines, legacy integration, auto-integration, and test reuse for reusable design core are included. Verification reuse defines verification patterns and environments for the subsystems at different abstraction levels. Self organizations support ubiquitous secure connectivity, mobile ad-hoc networks, and ability to adapt to user-specific behaviour.

Design and architectural patterns are important concepts in the field of software architectures to design applications by reusing generic design schemas established from successful and effective solutions. A great number of software applications are based on the same principles and their knowledge allow design efforts to be reduced considerably. Today, the main patterns are described in catalogues (Gamma et. al., 1994), (Buschmann et al., 1996). These catalogues describe the styles of organization and interaction at a higher level of abstraction, by presenting layered architectures, for example. A basic pattern for control is Strategy pattern. This separates the control from function to protect a client from various strategy services that it requires. Composite pattern is used in situations where it is necessary to treat components uniformly, regardless of whether they are primitive or composite. From behaviour perspective we can mention Chain of Responsibility pattern. Recursive control pattern (Selic, 1998) explains how to specify, then create hierarchic control architectures that are more flexible and more robust. This separates control aspects and the service providing aspects of a real time system allowing each to be defined and modified separately. The applicability of this pattern is across a wide range of levels and scopes starting from the highest system architectural level to individual components. This is useful in situations, typical in event driven real time applications where a complex software based server needs to be controlled dynamically in a non-trivial manner, where control policies may change over time. The Recursive Control is structurally related to the

Composite. It simplifies the implementation of complex systems by applying hierarchically a single structural pattern. Also it simplifies the development (and understanding) of both functional and control aspects by decoupling them from each other. It allows control or diagnosis services policies to be changed without affecting the basic functionality.

A *quality* based design requires a measurement instrument that must be defined by a taxonomy for quality attributes, which is organized with respect to three main elements: (1) The priority in a quality attributes list. The presence of this element in the taxonomy is necessary, due to the costs required by an analysis method at the architectural level. (2) Architecture views which are relevant for that quality attribute; (3) Appropriate methods to be applied for quality attribute analysis.

Quality attributes may be classified in essential, very desirable, desirable, don't care and forbidden. The priorities are established based on the experts' knowledge and the stakeholders' objectives. Quality function deployment (Reed, 1993) is a suitable technique for showing the relational strengths from objectives of stakeholders and architectural to quality attributes. These priorities are important for the evaluation process, which considers an analysis method for each quality attribute. At this moment various architecture analysis methods, such as scenario-based architecture analysis (SAAM) (Kazman et al. 1994), architecture tradeoff analysis (ATAM) (Kazman et al, 1998), architecture level prediction of software maintenance (ALPSM) (Bengston, 2004), or reliability analysis using failure scenario (SARAH) exist. Methods are distinguished by the evaluation techniques, the number of quality attributes and their interaction for tradeoff decisions, the stakeholders' involvement, and how detailed the architecture design is at the moment the analysis (Dobrica and Niemela, 2002).

The measurement instrument is applied to the RA during analysis. The quality attribute with the first priority in a list is first analyzed with respect to the appropriate architecture view and the appropriate method. Then the next quality attribute from the list is analyzed in isolation and then considering the interaction with the first one for finding sensitivity points and tradeoffs on the services included in the RA. The process is repeated for all the attributes in the list. In order to decide on RA core services, this procedure could also be improved and refined. In this case special attention should be paid to the collections of services in the architecture which are critical for achieving a particular quality attribute, or architectural elements to which multiple quality attributes are sensitive. A deeper level of analysis could influence the decision on the addition of new services to the RA.

5 CONCLUSIONS

This paper has proposed an approach for a RA development for complex ES application domains based on a knowledge of reusable concepts from software engineering at architectural level. The approach has an immense potential to improve embedded control systems development as well as reduce time and costs in stages such as architecture design and analysis. However, for this approach's success it is necessary to create a cooperation culture among embedded control system developers. Future research work is needed to develop systematic ways of bridging these reusable concepts to a RA, reducing in this way the cognitive complexity.

ACKNOWLEDGEMENTS

This work is supported by the Romanian research grant CNCSIS IDEI no. 1238/2008.

REFERENCES

MARTE, 2008, Modeling and Analysis of Real Time and Embedded, www.omg.org.

Buschmann F., R. Meunier, and H. Rohnert, 1996, *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley and Sons.

Gamma E., R. Helm, R. Johnson, and J. Vlissides, 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley.

Selic B., 1998, Recursive control, in: R. Martin, et al. (Eds.), *Patterns Languages of Program Design*, Addison-Wesley, pp. 147–162.

Kazman R., L. Bass, G. Abowd, M. Webb,1994, SAAM: A method for analyzing the properties of Software Architectures, *Procs of the ICSE*, 81-90.

Kazman R., M. Klein, M. Barbacci, H. Lipson, T. Longstaff, S. J. Carrière, 1998, The Architecture Tradeoff Analysis Method, *Procs. of the ICECCS.*

Reed B.M., D.A. Jacobs, 1993, *Quality Function Deployment For Large Space Systems*, National Aeronautics and Space Administration.

Bengston PO, 2004, Architecture Level Prediction of Software Maintenence, *Procs of the ICSR5*.

Dobrica L. and Niemelä E., 2002, A survey on software architecture analysis methods, *IEEE Transactions on Software Engineering*, 28(7), 638-653.