Genomial Co-Design: An MDA-compliant approach for embedded architectures

Janis Silins

Riga Technical University, Institute of Applied Computer Systems, Meza str. 1/3, LV-1048 Riga, Latvia
janis.silins@lep.lv

Abstract. This paper proposes a modified, genomial, version of function - architecture co-design method so that wider evaluation of architectures from both functional and structural point of view can be performed. The topological modelling technique is used to create the Computation Independent Model that is absent in the original method. A case study has been performed that clarifies the key ideas of this approach.

Keywords: Embedded, architecture, co-design, formal

1 Introduction

The rising complexity and quality requirements for modern embedded systems calls for development and use of highly integrated and formal design approaches. Many researchers agree that a novel perception of embedded architecture is required - one that does not draw strict boundaries between hardware and software components of a single system. The problem of unified view on embedded systems is addressed by various formal techniques, such as function - architecture co-design [4]. Unfortunately, this approach has certain drawbacks that have been discussed in [5]. A number of modifications and improvements have been proposed therein, having been consolidated into a new, genomial, approach.

2 The architectural genome

The most significant shortcomings of function - architecture co-design approach can be corrected by inclusion of evolving architectures in its meta-model. In general, the process of architecting should based on both functional and structural aspects of the system. Although the functional specification still remains a cornerstone for development of any architecture, the implementation details must not be left out of the architecting scope. The method should provide means of traceability; in the context of Model Driven Architecture (MDA), the trail is used to prove correctness of transformations. Finally, multiple iterations of the development process must be properly supported. The architecture, although being modified, should keep its original integrity without any *ad hoc* solutions.

Such evolving architectures are materialised in the modified co-design method as the "architectural genome" - a collection of formal descriptions and transformations. It acts as a repository that holds architectural components, expressed in pre-defined formal notation, supports join, merge and other operations on them, and enables the developer to revise and extend its contents during the development process. The description of embedded architectures has been accordingly extended and consists of:

- set of functional requirements or functional specification ($X = \{X_0 \dots X_i\}$);
- set of architectural constructs ($A = \{A_0 \dots A_i\}$) as a combination of primitives: elements of the utilized programming language (L), services provided by operating system (O) and services provided by hardware components (H);
- set of requirements enforced on the model of communications (Cr).

It must be noted that the set of architectural constructs (set A) includes members of various degrees of complexity and levels of abstraction, as listed below (in order of increasing complexity):

- Architectural primitives that are perceived as "black boxes". Their descriptions are included in the genome before development process starts, are treated as read-only information and can be freely re-used. The primitive set $P = \{O, H, L\}$ contains both abstract and real-world elements.
- The set of architectural constructs contains various combinations of primitives, therefore significantly more complex parts of the system can be developed. They still are not self-sufficient but nevertheless can be included in the models of either current or future systems. The specifications of members of this set $K = \{\{O_1, H_1, L_1\}, \{O_2, H_2, L_2\} \dots \{O_n, H_n, L_n\}\}$ are created by using merge operations on members of specific subsets of P. The set of architecture constructs is also suitable for re-use in other projects.
- The set of architectures consists of implementation candidates that are fully functional and self-sufficient. Its members are created from the elements of set of architecture constructs and primitives ($A = \{\{K_1, P_1\}, \{K_2, P_2\} \dots \{K_n, P_n\}\}$), and differ from them by being non-universal and valid only within boundaries of a particular project.

Genomial architectures are created as abstract entities (if the original requirements do not imply inclusion of particular COTS components), and after verification the abstract parts are gradually replaced with their real-world equivalents.

Although the genomial approach seems to bear a strong resemblance to the well-known component-based architecture design and style-based development methods, it shows significant differences from them. Genomial architecture constructs, unlike those of the component-based architecture, can be extended at will. If some of them are perceived as black boxes, it is an exclusion, not a rule. Also, architectural designs derived from a common genomial base, do not have the same vocabulary. In most cases, their constructions are only distantly related: they may be derivatives of common abstract objects.

Extension of function - architecture co-design approach also deals with selection of suitable formalisms with emphasis on aspects significant to architecture. One of such methods is topological modelling of functioning [3]. Its role in the

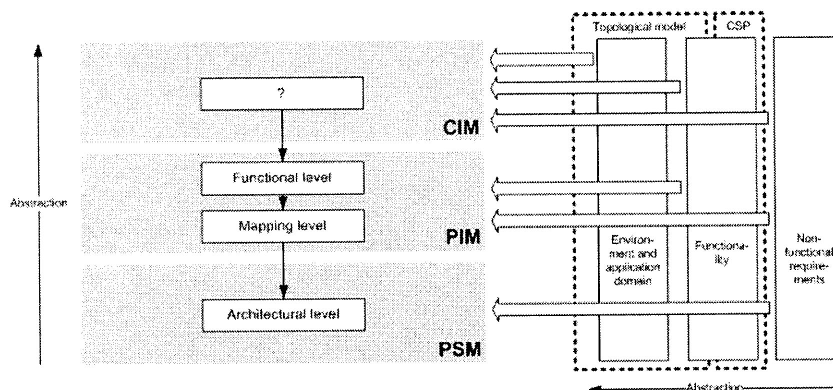

Fig. 1. Correlation between function - architecture co-design and MDA

genomial co-design environment is depicted in Figure 1. It covers all of the problem and application domain specification needs as well as a part of functional description - the portion of it that deals with high-level abstractions. By using the topological model as basis, the white areas of functional specification can be filled in with descriptions in functional algebraic notation (e.g., CSP). This specialized part of functional specification can be created simultaneously with the macro-architecture. Unlike the set of high-level functional properties, CSP-expressed requirements do not have to exist *a priori.*

3 Architecture description language requirements

Since the genomial co-design process is intended to cover all levels of MDA, including transition between CIM and Platform Independent Model (PIM), all features of it must be applicable to functional specifications and architectural entities alike. To keep the framework of genome as universal as possible, operations on architecture should be performed only on such essential properties that are provided by the most "minimalist" architecture description languages (ADL). As stated in [2], an ADL must provide a means of depicting at least (or semantic equivalents of) components and their interfaces, connectors and architectural configurations. For purposes of generality, a component and its interfaces can be expressed as structure
$C = \{B, I_1 \dots I_n\}; n > 0$
$I = \{P_1 \dots P_m\}; m > 0$

$B\ Implements(X_i \ldots X_j)$, where
B - behaviour expressed in formal way (e.g., using CSP) as implementation of one or more functional properties $\{X\}$ taken from the topological model,
I - set of interfaces,
P - set of ports that accommodate connectors.
Ports represent precise points of interaction between component and the outer world via connectors; at least one port per interface is required. Likewise, architectural configurations define topology and can be described as collections of components and connectors arranged in such a way that no component remains isolated and each connector has exactly two ports (P_i and P_j) connected.
$A = \{C_1 \ldots C_n, Cn_1 \ldots Cn_m\};$
$\forall l \in [Cn_1 \ldots Cn_m](l\ Joins(P_i, P_j))$, where
Cn - set of connectors.
Architectural configurations represent either full or sub-architectures. A configuration depicts full architecture if and only if it includes all functional properties X of a particular topological model; otherwise, it represents a sub-architecture. With these definitions made, general-purpose set operations on architectural configurations can be devised:

- Inclusion of components is expressed as union of sets of their interfaces and merger of behaviour descriptions:
  $C_1 \cup C_2 = B_1 \cup B_2, I_1 \cup I_2$
  Likewise, union of architectural configurations is expressed as:
  $A_1 \cup A_2 = C_1 \cup C_2, Cn_1 \cup Cn_2$
- Split of a component is a relative complement operation on its interface set; a new behaviour is also derived:
  $C_1 \backslash C_2 = B_1 \backslash B_2, I_1 \backslash I_2$
  Split of architectural configuration:
  $A_1 \backslash A_2 = C_1 \backslash C_2, Cn_1 \backslash Cn_2$

The actual implementation of these operations is language-dependent, and is generally performed by automated means.

4 Example

In order to demonstrate the nature of evolving architectures, the case study involves development of a consumer-grade car navigation system. The $xADL$ v2.0 architecture description language [1] has been chosen as the formal carrier for architectural design elements. It is assumed that developer has no ready-made component libraries and/or frameworks available, thus allowing the example to show how architectural libraries emerge from ground up. The main function of car navigation system is to display the current position of vehicle on an electronic "moving" map. Maps should be user-upgradeable from an external USB flash-memory module.

Step 1: analysis of description and creation of CIM

The initial high-level design, as devised from the non-formal natural-language description and represented by means of topological model of functioning, has been given in Fig. 2a. Acting as CIM if speaking in terms of MDA, this model does not address implementation-related problems and remains highly abstract, but nevertheless provides useful information that clarifies how the architecture should be built. The high-level model depicts main functional properties that

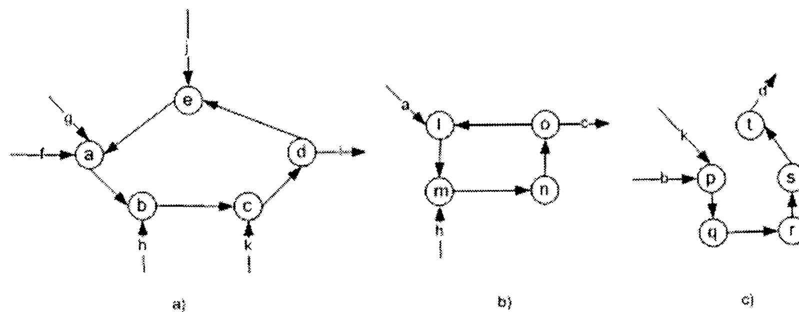

Fig. 2. Topological model of navigation system

are linked together by causal relationships, as well as ties with the surrounding environment that exist beyond the development scope but are sufficiently close-coupled with the system in question. The list of main functional properties (nodes) has been extracted from given description and forms the main cycle of functioning: a) manage power supply and initiate low-power mode; b) acquire GPS position data; c) prepare digital map data; d) display current map on screen; e) accept user's input commands from thumbwheel and react on them.

The system receives information and power from the environment, thus interacting with it: f) power from car battery (constant supply); g) power from ignition system (sporadically interrupted supply); h) signal from GPS satellite system; i) graphical display image; j) user's input commands; k) digital maps from external source (flash module).

In order to support proper GPS hot and cold start-up times, the model is refined further by expansion of node *b* (see Fig. 2b) and now contains provisions for support of proprietary GPS low-power modes. This extension allows for GPS module to be "woken up" for periodic ephemeris checks while the rest of system remains in inactive state.

The topological model now contains a number of functional sub-nodes that form a first-level sub cycle: l) manage GPS power; m) manage active external antenna; n) receive satellite signal; o) decode position data.

Likewise, node *c* has been extended for better understanding of map preparation phase (see Fig. 2c). Maps in digital form are pre-loaded from external media,

cached and sorted for faster access to them, and selected for subsequent display on screen. The following functional nodes have been added in linear manner: p) read maps from external media; q) cache geospatial outlines of maps; r) determine map visibility; s) pre-load visible maps; t) place visible maps on coordinate grid.

Although refinement of the topological model of functioning can proceed further, the basic functionality has been already included. With that, transition to PIM can commence.

Step 2: Transition to PIM
Each functional node present in the topological model is being mapped onto an unique component in ADL namespace, and each arc is being represented as ADL connector. In order to preserve arc directions, a default interface is created for each component with *In* or *Out* port defined for each respective arc (see Fig. 3). Arcs that connect the system with outer world are omitted, and only their associated ports (or collections thereof) remain. Figure 5 depicts a naive architecture (designated A_n) in box and arrow equivalent of *xADL* notation. A_n is abstract, for no behaviour has been defined for any of its components, and still platform-independent. All components and constructions resident therein have been entered into the repository of architectural genome after being sorted by origin and complexity: $A = \{A_n\}; K = \{a, b, c, d, e, l, m, n, o, p, q, r, s, t\}; L = \{\}; O = \{\}; H = \{\}$.

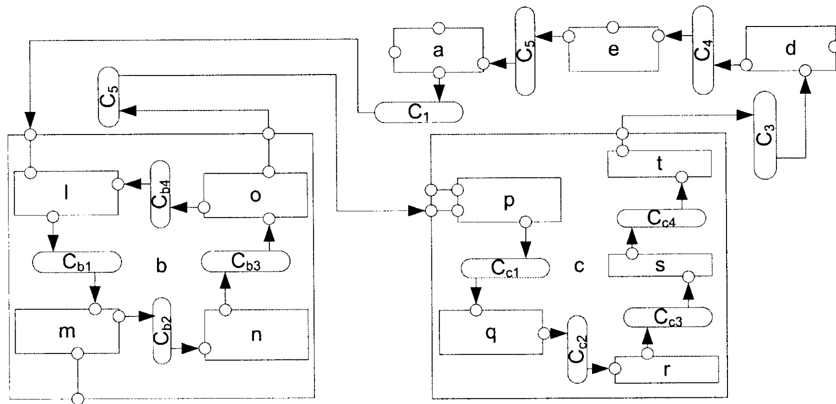

Fig. 3. Naive architecture of navigation system

As the next step, contents of A_n are being gradually replaced with their less abstract derivatives. This process is influenced by non-functional requirements that originate from both system's initial description and nature of problem domain. Its source topological model contains one first-order sub-cycle; therefore, the sub-cycle has to be implemented as a separate thread that runs in context

of the main process (main cycle of topological model). The following changes to architecture A_n have been made:

- Sub-architectures of process (u) and thread execution models (v) have been created and included in genome's repository; they both, although abstract, are operating system services: $O = \{u, v\}$;
- Component b has been merged with sub-architecture v that together form b_v. It now exposes interfaces of both b and v:
  $b_v = b \cup v; K = \{a, \{b, b_v\}, c, d, e, l, m, n, o, p, q, r, s, t\}$
- Components a, c, d, e are merged into new component z since they share a common execution context (reside in the same process):
  $z = a \cup c \cup d \cup e; K = \{\{b, b_v\}, \{a, c, d, e, z\}, l, m, n, o, p, q, r, s, t\}$
- Component z has been merged with u to form the main process z_u:
  $z_u = z \cup u; K = \{\{b, b_v\}, \{a, c, d, e, z\}, l, m, n, o, p, q, r, s, t\}$;
- Since component b forms a separate thread, its outer connectors can be implemented as shared variables and have been assigned type C_{sv}. Their behaviour can now be specified, thus they cease to be abstract:
  $L = \{\{C_1, C_2, C_{sv}\}\}$;
- All other connectors reside inside a common execution context and will form programming language function entry and exit points (C_a), depending on their direction: $L = \{\{C_1, C_2, C_{sv}\}, \{C_3, C_4, C_5, C_{b1}, C_{b2}, C_{b3}, C_{b4}, C_a\}\}$.

The changed architecture is designated A_{n1} and saved in repository:
$A = \{\{A_n, A_{n1}\}\}$

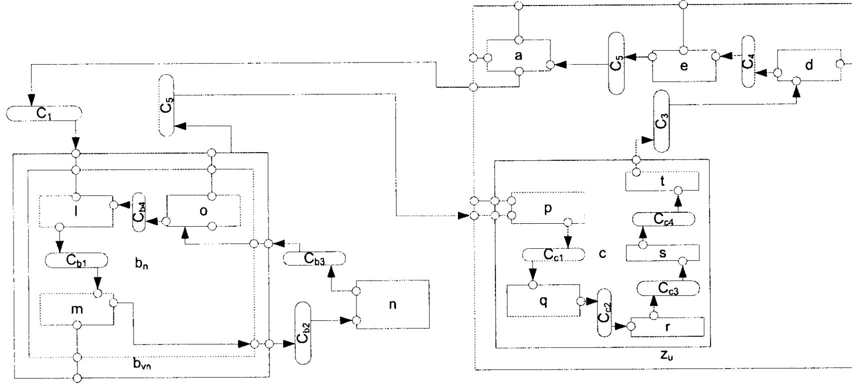

Fig. 4. Architecture refinement: second stage

By gradually supplying computation parts to components and extending them, the architecture becomes complete and ready for implementation. Components whose behaviour cannot be implemented in software or it is impractical to do so, will be excluded from their respective parent configurations. One of such components is n that, in reality, performs functions of a GPS receiver:

- Component n is excluded from b; b is replaced by derivative b_n and b_v with b_{vn}:
  $b_n = b \backslash n$; $K = \{\{\{b_v, b_{vn}\}, \{b, b_n\}\}, \{a, c, d, e, z\}, l, m, n, o, p, q, r, s, t\}$
- Connectors C_{b2} and C_{b3} will be implemented as serial lines and typed as C_s:
  $O = \{u, v, C_s\}$

The changed architecture is designated A_{n2} and saved in repository (see Fig. 4):
$A = \{\{A_n, A_{n1}, A_{n2}\}\}$
All of the changes made to architecture are reflected in the topological model via PIM-CIM transformation. The model of architecture A_{n2} can be considered functionally complete and ready for transition to PSM.

5 Conclusion

The example of practical application of the genomial co-design approach shows some of the possibilities offered by the concept of evolving architectures. With an established repository of ready-made architectures and their separate components, new systems can be created more effectively. Thus, a step is made towards reuse of models, specifications and code across single or multiple problem and application domains.

Further research has been planned concerning tool support and automated model checks. Its purpose is to improve merge and split operations on descriptions of behaviour so that their correctness and completeness can be guaranteed before and after each transformation of a model.

This work has been partly supported by the European Social Fund within the National Programme "Support for the carrying out doctoral study program's and post-doctoral researches" project "Support for the development of doctoral studies at Riga Technical University".

References

1. E. M. Dashofy, A. van der Hoek, and R. N. Taylor. An infrastructure for the rapid development of xml-based architecture description languages. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 266–276. ACM Press, 2002.
2. N. Medvidovic and R. N. Taylor. A framework for classifying and comparing architecture description languages. In M. Jazayeri and H. Schauer, editors, *Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97)*, pages 60–76. Springer–Verlag, 1997.
3. J. Osis. Formal computation independent model within the MDA life cycle. *International Transactions on Systems Science and Applications*, 1(2):159–166, 2006.
4. M. Sgroi, L. Lavagno, and A. L. Sangiovanni-Vincentelli. Formal models for embedded system design. *IEEE Design and Test of Computers*, 17(2):14–27, June 2000.
5. J. Silins. The genomial co-design approach for design of embedded systems. In *Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems PDeS 2006*, pages 150–155. Brno University of Technology, 2006.