A Transformation-Based Model of Evolutionary Architecting for Embedded System Product Lines

Jakob Axelsson

School of Innovation, Design and Engineering, Mälardalen University,
SE-721 23 Västerås, Sweden

Abstract. In many industries, embedded software plays an increasingly important role in defining the characteristics of the products. Often, a product line approach is used, and the system architecture is developed through evolution rather than being redone from scratch for each product. In this paper, we present a model of such an evolutionary process based on architecture transformations. The model attempts to give an accurate description of how real architects actually work. Key elements of the approach are how the transformations interact with consistency constraints and with feasibility in terms of resource limitations. The work is based on findings from previous case studies in the automotive industry. The model can be used to enhance our understanding of the architecting process, and to find ways to improve it.

Keywords: Architecture, embedded systems, evolution, transformations.

1 Introduction

The increasing complexity of embedded systems leads to soaring development costs, and many companies strive to curb this trend by reusing software and hardware between products through a product line approach. This makes architecture very important, and we have previously done in-depth studies of the architecting practices at a some companies (see e.g. [6]), showing that instead of following a well-defined process and method, the architects base their work on experience and gut feeling. Academic literature on architecting is mostly concerned with developing a new system from scratch, something that rarely occurs in the organizations mentioned above. We term this traditional approach *revolutionary* architecting, and we have previously argued based on another case study that the focus should instead be on the *evolutionary* architecting where a new version of an existing product is developed [1]. To systematically attack the problem of lacking processes and methods for architecting, there is a need to provide a description of how architects work today. The research question of this paper is therefore: What is a suitable model for capturing how evolutionary architecting is performed in organizations developing complex embedded system? The contribution of the paper is to propose such a model, which is based on transformations of an architectural description and related analyses. Using this model, it becomes possible to reason about aspects of the architect's work and to describe phenomena encountered during empirical research on architecting.

2 Evolutionary architecting and architecture descriptions

In the evolutionary process, architecting is triggered by a product change request. The architects get input in terms of requirements primarily from the function developers. The architects then try to design a high-level technical solution, focusing on the distribution of functionality onto different systems, and on the interfaces between systems. When designing the high-level solution and evaluating alternatives, they take into account not only the requirements, but also architectural quality attributes, which are properties of the architecture itself which they strive to maintain. Throughout the work, the architects create descriptions of the architecture. The descriptions are used to define pre-requisites for the system developers.

The architect primarily focuses on resolving issues that go across several subsystems, and this entails dealing with the following concerns:

- *Feasibility*, i.e. possibility to implement the functionality by the available computational resources.
- *Consistency*, i.e. that all interfaces between parts are well defined.
- *Optimality*, in terms of important quality attributes (including cost).
- *Modifiability,* to enable future evolution.

The model presented here attempts to describe what information the architects deal with in their work. That information might appear in many forms: formal models, sketches, texts, or just as mental models inside the architect's head. Our model tries to capture the essence of that information, and disregard its representation.

For modeling the architecture descriptions for embedded systems, it suffices with a metamodel (M2 level) that is essentially an annotated graph, containing *elements* of different kinds; *relations* between pairs of elements or between pairs of relations; and *attributes* describing properties of elements and of relations.

For distributed, embedded systems, a model (M1 level) for describing the architecture can be grouped into several levels of abstraction. In this paper, we will use four different views, whose elements and relations are shown in Table 1. (The description is similar to that provided in [2], except that the cluster level is implicitly captured through the allocation relations. Also, the physical packaging level is added in this paper, and the task level is excluded since it is internal to an ECU.) There are also relations between entities in different views, indicating which modules *realize* each function, how modules are *allocated* to ECU:s, where hardware elements and external entities are *positioned*, and how communication is *routed*.

The metamodel allows *attributes* on elements and relations describing their properties. For architects, the primary properties have to do with desired *qualities* and limited *resources* present. The desired qualities are those properties that the architect tries to optimize when selecting among alternative feasible solutions. One of the most important ones is *cost*, which can be further divided into *product cost* and *development cost*. The product cost is essentially the cost of hardware, so we add a product cost attribute to each element of the hardware view. Important resources are present in ECUs (*processing capacity*, *memory size*, *I/O pins*), communication channels (*bandwidth*), and spaces and routing channels (*volume*).

Table 1. Views, elements, and relations in architecture descriptions.

View	Element	Relations within view
Functional	Function External entity	Functional dependency
Logical	Module	Data flow
Hardware	ECU Sensor Actuator Comm. channel	Signal flow
Positioning	Space Routing channel	Connection

Architects do usually not make *complete* models of the entire architecture, but rather only describe those parts which are relevant to resolve a certain change request. Therefore, we should not assume that we are dealing with complete information. However, among those elements related to the change request, *consistency* must be reached so that for instance all necessary relations are present. As an example, if the change is to add a new function, which is realized by a certain set of modules, all those modules must be allocated to ECUs, and none can be left dangling.

3 Transformations and analyses

We believe that the essence of the architect's work can be captured as a sequence of transformations of the architectural description (on the instance, or M0, level), together with analyses to see that the solution is feasible, cost efficient, and future proof. Just as the architecture descriptions can take many forms, including mental models, the transformations can in reality be explicit or very implicit.

There are two basic transformations on the metamodel level: *add entity* and *remove entity*. Since the entities are either elements or relations, the possible transformations become *add element, remove element, add relation,* and *remove relation.* At the model level, these abstract transformations can be made concrete, resulting in, e.g., *add module*, *remove ECU.*

Sometimes architects also use composite transformations. A good example is *change relation*, which basically consists of *add relation*$_1$ followed by *remove relation*$_2$. A concrete example is when a module that used to be allocated to one ECU is moved to another ECU by a *change allocation* transformation. Other composites are *change element* (e.g. *change ECU* to, e.g., an upgraded processor); *split element* (e.g., *split module* when a software module is divided to allow distribution); and the reciprocal *merge element*. Through the composite relations, we end up with a formal language which is very close to the natural language used by architects during their daily work.

As described in Section 2, the architect's work is triggered by a change request to an existing architecture, which is consistent and feasible. This change request can be

described as an initial set of transformations. A typical change request is to integrate a new function, i.e., the transformation *add function*. At this point, the architecture description has become largely inconsistent.

The first step of the architect is usually to try to get more details about the functionality in the requirements analysis phase. This involves identifying external entities involved (using the transformations *add external entity*, *add functional dependency*). Also, it is important in this phase to identify placement limitations. After the requirements analysis is completed, there is usually a complete and consistent description of the functional view.

Next, the architect starts to generate possible solutions. This is done by filling in the details at the logical level through transformations such as *add module, add dataflow,* but also *change module* since a consequence of an added functional dependency may be that an existing module needs to be updated. Also, the hardware view is detailed, possibly by *add ECU, add sensor, add actuator* or *add network* transformations. The relations between the logical and hardware views also need to be figured out, by *add allocation* transformations. In this step, it is common that the logical view needs to be revisited to perform *split module* transformations in order to find a good allocation. Finally, the hardware and positioning views must be connected by *add positioning* and *add routing* transformations. According to our observations, there is usually not a clear step-by-step process through the views, but the architects appear to work with all views in parallel or iterate between them. Figure 1 illustrates the search process performed by architects when dealing with a change request.

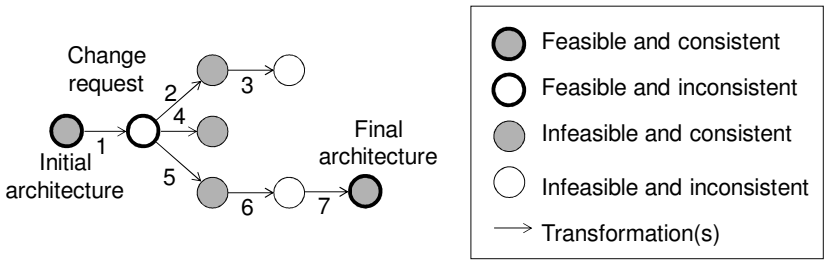

Fig. 1. Evolutionary architecting as a sequence of transformations.

If consistency is what drives the architecting forward, analysis of feasibility and quality is what guides it. The most important analyses correspond to the concerns of the architect described in Section 2 above. Usually, the analyses are qualitative rather than quantitative, and often relative rather than absolute. Difficult trade-offs between the concerns are often needed.

For each resource, a set of users can be derived to see that the solution is *feasible*, i.e. that the resources are not exhausted. Whenever a relation is added to the model, which entails that one element will use resources of another, the feasibility should be checked. An example is when a module is allocated to an ECU. Then the architect must evaluate if it will fit in terms of ECU memory and CPU footprint.

The *product cost* is simply the sum of the cost of all components, which can be calculated by adding the cost attributes of all entities in the hardware view. *Development cost* is more complex to assess. In [3], it is described how to reason

about the cost for software changes. The approach is to first identify which modules change, and then either simply count how many modules are touched, or try to perform a more refined analysis or initiated guess of the magnitude of change.

The architects also try to keep in mind that the architecture should be *modifiable*. However, as pointed out in [5], it is not meaningful to reason about modifiability as such, but only how modifiable the architecture is with respect to a certain class of changes. For embedded systems, a common barrier to modification is lack of hardware resources. The architects try to strike a balance between adding surplus resources to the hardware for future growth, and optimizing the resources in order to reduce product cost. This kind of reasoning can be thought of as a real options analysis [4]. In such an analysis, the main difficulty is to estimate the likelihood of certain types of changes. If architects keep track of how frequent certain transformations are, they can extrapolate more reliable figures. The transformation model thus gives the architect a language for capturing knowledge about changes.

4 Conclusions

In this paper, we have outlined a Transformation-based Evolutionary Architecting Model (TEAM), which attempts to describe essential knowledge about how real architects go about developing embedded system product lines. The basis is data collected from observing real architecting work, and we have attempted to construct a model with a high fidelity in the sense that the language the architects use to describe their own process should be possible to map to the model.

Although the model presented in the paper is largely based on experiences from the automotive domain, the fundamental ideas are captured in the metamodel which is much more general and allows many different views and elements to be included.

References

1. Axelsson, J. Evolutionary architecting of embedded automotive product lines: An industrial case study. In Proc. Joint 8th Working IEEE/IFIP Conf. on Software Architecture & 3rd European Conf. on Software Architecture, pp. 101-110. Cambridge, UK, Sept. 14-17, 2009.
2. Broy, M., Krüger, I., Pretschner, A., and Salzmann, C. Engineering Automotive Software. Proc. IEEE, Vol. 95, Issue 2, pp. 356-373, Feb. 2007.
3. Eden, A. H. and Mens, T. Measuring Software Flexibility. IEE Software, Vol. 153, Issue 3, pp. 113-126, June 2006.
4. Gustavsson, H. and Axelsson, J. Evaluating Flexibility in Embedded Automotive Product Lines Using Real Options. In Proc. 12th Intl. Software Product Line Conf., pp. 235-242. Limerick, Ireland, Sept. 8-12, 2008.
5. Parnas, D. L. Software aging. In Proc. Intl. Conf. Software Engineering, pp. 279-287. Sorrento, Italy, 1994.
6. Wallin, P. and Axelsson, J. A case study of issues related to automotive E/E system architecture development. In Proc. 15th IEEE Intl. Conf. on Engineering of Computer Based Systems, pp. 87-95. Belfast, Northern Ireland, March 31-April 4, 2008.