ATTRIBUTE-BASED PRODUCT-LINE ARCHITECTURE DEVELOPMENT FOR EMBEDDED SYSTEMS

LILIANA DOBRICA*, EILA NIEMELÄ**

*Politehnica University of Bucharest, Faculty of Control and Computers,
Spl. Independentei 313, sect. 6, 77206, BUCHAREST, Romania
e-mail: *liliana@ciid.pub.ro*

**VTT Electronics,
Kaytoväylä 1, P.O. Box 1100, 90571 OULU, Finland
e-mail: *Eila.Niemela@vtt.fi*

ABSTRACT

This paper suggests a practical solution for product-line architecture (PLA) development in embedded systems, founded on analysis of the software quality attributes at the architecture level. PLA is developed based on a measurement instrument, which determines whether a product should be added to a product line or not, which parts should be added to the domain architecture framework, and which parts should be product specific.

1. Analyzing quality attributes in a domain

Product-line (PL) and reusable software components are suitable approaches for embedded systems, which are often re-engineered from existing systems. Important issues in the development and maintenance of these software systems are functionality and quality. Although there are some similarities between embedded systems regarding quality attributes, there are also differences. If a quality attribute is important to one product-line domain, it does not necessarily mean it is important to another one. For example, availability is very important to switching systems, but generally it does not have the maximum level of priority. Power consumption and weight are features that may cause restrictions in the case of wireless products. In addition, the weight changes when different types of software that may exist in a product are considered. Performance requirements, such as latency and throughput, have the highest value for digital signal processing software. An important quality is cost, which is caused by the code size and data storage capacity. In order to be able to define a list of quality attribute priorities the embedded systems domain should be very well delimited.

Domain analysis is also important for the evolution of a PL. From this point of view the development quality attributes such as modifiability, reusability, maintainability etc. should be considered and another list with their priorities could also be organized.

2. Definitions of the main concepts related to software PLA

In this section we present an overview of the main concepts that are frequently relevant in the context of software PLA. The first part is concerned with the terminology related to the common denominator in software engineering development, which is considered to be software architecture. The definition, description and design elements of software architecture are introduced here. The second part focuses on the particular concerns related to PL software development. The definition, initiation and evolution processes of software PL and also PLA specific description represent the main headings of this second part.

Definition of software architecture. A definition of software architecture is given in [3]. Here the software architecture of a program or computer system is defined as "the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them". This definition focuses only on the internal aspects of a system and most of the analysis methods are based on this definition.

Description of Software Architecture. The research in the architecture description is designed to address the different perspectives one could have of the architecture. Each perspective is described as a view. Although there is still no general agreement about which views are the most useful, the reason behind multiple views is always the same: *Separating different aspects into separate views help people manage complexity.* The

information relevant to one view is different from that of others and should be described using the most appropriate technique for each view. Several models have been proposed that include a number of views that should be described in the software architecture. The view models share the fact that they address the static structure, the dynamic aspect, the physical layout and the development of the system. Bass et al. [3] introduce the concept of architecture structures as being synonymous to view. In addition to the four structures that are identical to the views in the 4+1 View model [18], the authors mention the *uses structure*, the *calls structure*, the *data flow*, the *control flow* or the *class structure*. In general, it is the responsibility of the architect to decide which view to use to describe the architecture.

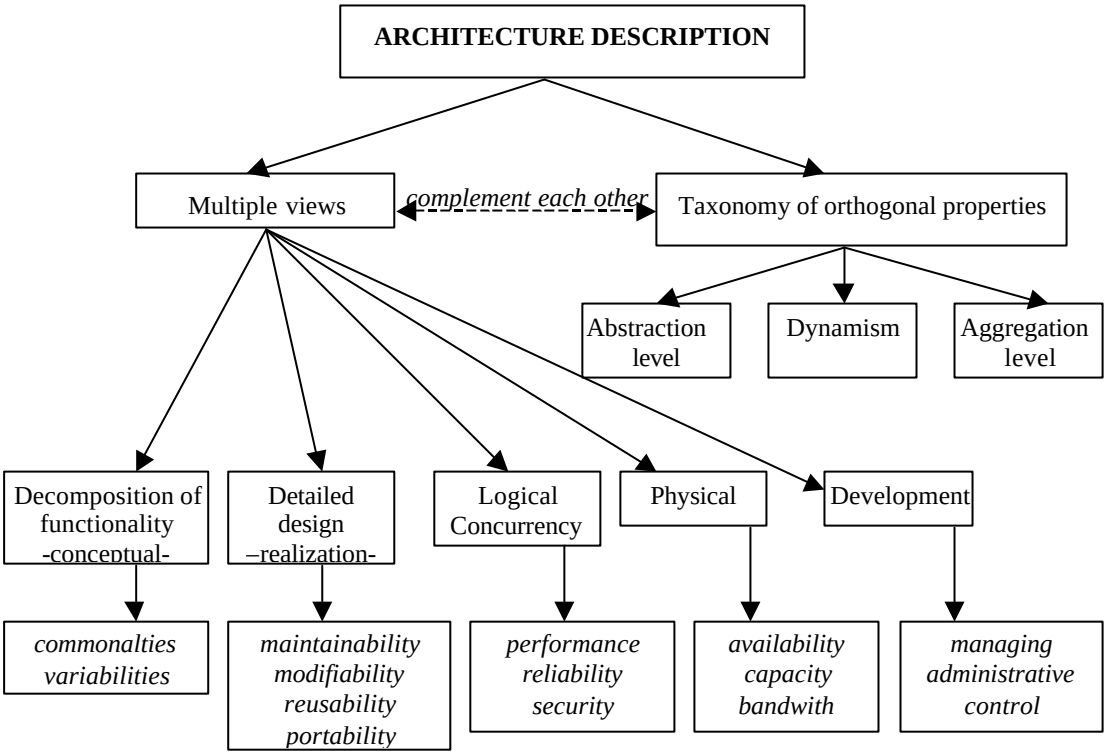

Figure 1. Architecture description and the relevance to analysis of quality attributes.

From the point of view of quality analysis at the architectural level, the possible representations could be very relevant in quality prediction and effort estimation. An evaluation method may need structures, which are concerned with (Figure 1):

- The decomposition of the functionality that the products need to support. Components are functional (domain) entities, and the connectors are 'uses' or 'passes-data-to' relations.
- The realization in a detailed design of the conceptual abstractions from which the system is built. Components can be packages, classes, objects, procedures, functions, methods, etc., all of which are vehicles for packaging functionality at various levels of abstraction. Relations include passes-control-to, passes-data-to, shares-data-with, calls, uses, is-an-instance of, etc. This structure could be crucial for understanding the maintainability, modifiability, reusability and portability of the system.
- Logical concurrency. The components of this structure are units of concurrency that are ultimately refined to processes and threads. Relations include synchronizes-with, is-higher-priority-than, sends-data-to, can't-run-without, etc. Properties relevant to this structure include priority, preemptability and execution time. This structure is a key to understanding performance and is also important in reliability and security.
- Hardware, including central processing units, memory, buses, networks and input/output devices. Properties relevant to this structure include availability, capacity and bandwidth.
- Files and directories. This structure is important for managing and ensuring administrative control of the system as is fleshed out, including the division of work into teams (i.e. modularity) and configuration management.

A taxonomy of formally-defined orthogonal properties of software architectures (TOPSA) that extends an architecture definition is given in [8]. The TOPSA space has three dimensions: *abstraction level* (conceptual, realization), *dynamism* (static, dynamic) and *aggregation level*. The TOPSA can facilitate discussions regarding software architecture during development and evolution. This model makes a clear distinction between

conceptual architecture and realization architecture, suggesting the creation of architecture models suited for different purposes and different stakeholders.

Syntactic architectural notations should be well understood by the parties involved in the analysis. Architectural descriptions need to indicate the system's computation and data components as well as all the relationships between components. The result of an architecture evaluation process depends on how well the description is made. The TOPSA model and the architecture representation based on multiple views complement each other. Different views offer valuable examples for abstraction, dynamism and aggregation dimensions in a TOPSA space. Table 1 exemplifies the mappings of two quality attributes, the performance and reusability, on the architecture views associated in a TOPSA space. For example, performance could be analyzed from the logical concurrency viewpoint in a realizational plane of TOPSA space.

Table 1. The mapping of quality attributes on architecture description.

Quality attribute	***Architecture View***	***TOPSA space (Abstraction, Dynamism, Aggregation)***
Performance	Logical Concurrency	Realizational
Reusability	Functional, Detailed design, Development	Conceptual/Realizational, Aggregation

An analysis method could exploit these relationships in the form of a defined set of rules, which states which view in the TOPSA space is the most appropriate for the analysis of a given quality attribute.

Software Product Line. A *software product line* is defined as a group of products sharing a common, managed set of features that satisfies specific needs of a selected market [3]. The products should be suitable to a market strategy and application domain. They share an architecture and are built from components. The PL definition includes three main concepts which are feature, architecture and component. The common set of features represents the source of information for the shared architecture design, which is called PLA. From the viewpoint of decomposition and composition approaches of the architecture design methods, we can identify a bi-directional relation between PLA and the set of components [2]. In one direction of this relation, PLA influences component development considering the decomposition. In the opposite direction of this relation the composition aspect acts and the set of components influences PLA development. Some of the issues with PL are related to the process of initiation and how to deal with its evolution process.

- *Initiation.* Software PL does not appear accidentally, but require a conscious and explicit effort from the organization interested in using the PL approach. Basically, one can identify two relevant dimensions with respect to the initiation process. The organization may take an evolutionary or a revolutionary approach to the initiation process. On each dimension the PL initiation can be applied to an existing line of products or to a new PL that the organization intends to use. Each case has an associated risk level and certain benefits. For instance, in general, the revolutionary approach involves more risk, but should bring higher returns compared to the evolutionary approach. [7]. The revolutionary approach to a new PL means that PLA and components are developed to match the requirements of all expected PL members, before developing the first product in a new domain.
- *Evolution.* Research into software PL shows that the evolution of a PL is driven by possible changes in the requirements of existing members or the addition of new product members [6]. In the case of changes in the requirements of existing members, a typical way of handling this is to create two independent evolution cycles for each product. One cycle incorporates new product-specific requirements, and the other is for PLA as a whole, incorporating new requirements that affect all or most of the products in the PL.

PLA specific description. PLA is a software architecture and a set of reusable components shared by a family of products [3]. The components are large pieces of software, and they are typically modeled as object-oriented frameworks that cover functionality common for the products in the PL, and support the variability required for the various products. An important aspect which should be considered is the component-based development of PLAs. A component-based architecture is a general solution for improving modifiability and reusability. At the conceptual level PLA description is modeled by an abstract framework with component interconnections and a detailed representation of each contained component. The abstract framework could be considered a domain-specific architecture which provides a global perspective of the group of products, while a component detailed representation is a micro-architecture which represents a cluster of features.

Similar structures as for single product software architecture could be used for the representation of PLA. However, due to the presence of additional terms like scope, commonality and variability, it is possible that new

views are to be introduced, such that facilitate the analysis of PLA. Variability could be incorporated in each view of the architecture. For example, the conceptual view describes the components and their interconnection from the static point of view. In an abstract framework the variability could be expressed by not showing what is variable or by using specific relations and notations (aggregation, specialization, etc.) of the modeling language (UML for example). The usage of packages is a solution to express the diversity of specific component models. For each product member a package with specific concrete components is organized.

3. A PLA analysis strategy

The advantages of adopting the PL approach are decreased development and maintenance cost and time to market and increased software quality. This section suggests a practical solution for PLA development founded on analysis of the software quality attributes. The first part is an introduction to the PLA analysis strategy. The remainder focuses on PLA analysis when evolution of the PL is considered. We propose a measurement instrument, which determines if a new product should be added to an existing PL, which components should be added to the domain architecture framework, and which parts should be product-specific.

Introduction to the PLA analysis strategy. There is a common opinion among researchers that the development process for PLA is different than for one product [19]. Practice areas essential to the development of a single product architecture include software architecture design, representation, analysis, implementation in conformance with the specification, and traceability with requirements. In the PL approach, additional areas like generic and flexible software architectures, reusable components design and development, and traceability with common and variable requirements in the group of products are considered.

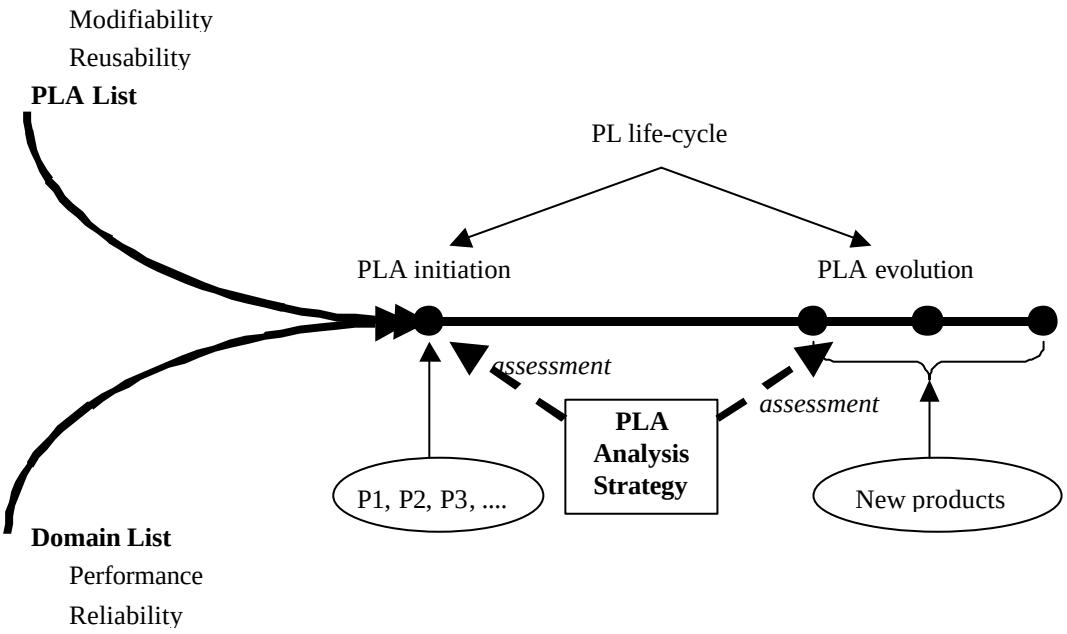

Figure 2. The context of PLA analysis strategy.

PLA is mainly design whilst the PL is initiated based on existing products and it evolves as long as new products are added to the line. Figure 2 describes the context of PLA analysis strategy, which considers the process of initiation and evolution of PLA. PLA analysis strategy takes into account the assessment process of the two lists of quality attributes.

The measurement instrument. The measurement instrument is defined by a taxonomy for quality attributes, which is organized with respect to three main elements:

1. The priority in a domain or PLA 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.

The priorities of the quality attributes in a domain are established based on the experts' knowledge and the stakeholders' objectives (Figure 3). Quality function deployment (QFD)[11] is a suitable technique for showing the relational strengths from stakeholders' objectives and architectural objectives to quality attributes. The list of priorities is important for the evaluation process, which considers an appropriate analysis method for each quality attribute. At this moment various architecture analysis methods, such as scenario-based architecture analysis (SAAM)[16], architecture tradeoff analysis (ATAM)[15], architecture level prediction of software maintenance (ALPSM)[5], or scenario-based architecture re-engineering (SBAR) are available [4]. Our study of the existing state-of-art research into the domain reveals that the methods are distinguished by taking into account the included evaluation techniques (qualitative or questioning, such as scenarios; quantitative or measuring, like metrics, etc.), the number of considered quality attributes and their interaction for tradeoff decisions, the stakeholders' involvement, and how detailed the architecture design is at the moment the method is applied to the architecture-based development process [12]. Figure 3 exemplifies the most appropriate methods for analyzing several quality attributes.

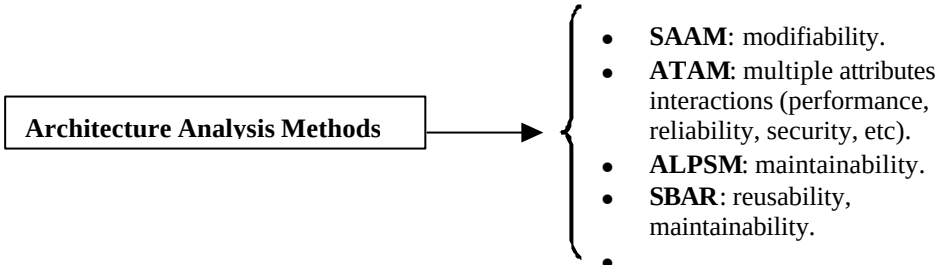

Figure 3. An example of quality attributes and the appropriate analysis methods.

The evaluation procedure. The measurement instrument is applied to the architecture of the new product. The quality attribute with the first priority in a list is first analyzed with respect to the appropriate architecture view and the appropriate method. If the results of the analysis are not acceptable, then the new components of the product should be not added to PLA nor the new product to the PL. Otherwise the next quality attribute from the list is analyzed in isolation and is then considered in interaction with the first one. The process is repeated for all the attributes in the list. If the obtained results are acceptable, the new components of the new product should be added to PLA and the product to the PL. The steps of the evaluation procedure are described below:

Initialization
counter := the number of quality attributes in the domain list;
1. *The quality attribute with the first priority is analyzed with respect to the appropriate architecture view and the appropriate method.*
2. *If the results of the analysis are not acceptable the new components of product should be not added to the PLA.*
3. *Else*
counter:=counter-1;
repeat // repeat for all the attributes in the list
the next quality attribute from the list is analyzed as in the first step
in isolation;
considering the interaction with the previous ones;
if the results are poor
break ;
else
counter:=counter-1;
until (counter=0) // end repeat
4. *If (counter=0)* // all attributes of the list and the obtained results are good
the new product is added to the PL.

In the case of quality attributes with the same priority, they should be analyzed in isolation, and the interactions between each other should also be considered. In order to decide on PLA core development and maintenance, this procedure could also be improved and refined. From this point of view, the other list of priorities should also be considered. In this case special attention should be paid to the collections of components in the architecture of a new product which is critical for achieving a particular quality attribute, or architectural

elements to which multiple quality attributes are sensitive. A deeper level of architecture analysis could influence the decision on the addition of new components to PLA.

4. The case study

Until now we have discussed the theoretical aspects related to software PLs and architecture assessment. An analysis strategy for PLA is very hard to discuss on an abstract level. Instead, one needs a concrete example of a PL initiated in a revolutionary approach in software development. In this section we present a case study of PLA. To start with, we introduce the scope of the concrete software PL. The products that are planned to be part of the PL need to exhibit sufficient commonality, and it should be possible to handle the variability in a structured fashion. The last part exemplifies the available views of the PLA representation.

Scope of the Product Line. Our case study is a PLA representation of scientific on-board silicon X-ray array (SIXA) spectrometer control software [14]. 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 in observations of time-resolved X-ray spectra for a variety of astronomical objects. The instrument is programmed and operated using a set of commands sent from the ground station to the satellite. The role of a software 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 in use.

A striking similarity among the requirements of each measurement controller makes the initiation and development of a PL possible. The PL domain is structured in packages of features based on the requirements specification. The common and variable aspects of features are mapped onto common or specific packages. In order to initiate the PLA we consider the first software product members to be the following:

- EGYController, which includes specific features of a standalone control of EGY measurement mode;
- SECController, which includes specific features of a standalone control of SEC measurement mode;
- SECwithEGYController, which includes specific features of coordinated control.

PLA representation. In order to predict any quality attribute based on the architecture model it is important to define a clear and precise terminology of these elements.

- *Components or elements.* Each element has a name, an interface, ports and a behavior. The name serves the purpose of identification, the interface is the specification of what methods or services the architecture element provides for external use and subsequently must provide an implementation for.
- *Static relations.* The elements in the architecture have relations to other elements. These relations make up the form or the structure of the architecture. The relations must be documented by specifying, either in text or graphically, the related components (2 or more) and the relation type (association, aggregation, etc.).
- *Dynamic relations.* The dynamic aspect of the software architecture is important, in order to determine its functions and qualities. The plausible way to document the dynamic aspect is to use *message sequence charts* (MSC) for the method usage between architecture elements. MSC can offer a great deal of help in understanding the software without going too deep into the details, which are left open for the detailed design.

PLA design is represented by different views, which have to be considered in the analysis of its main structural qualities. Among these the conceptual view and detailed functional decomposition view could be enumerated. These are described in the following sections. We also introduce a diversity view, due to the necessity to represent the common and variable elements of different product members in a unitary structure.

Conceptual View of the PLA. Figure 4 presents the conceptual view of the spectrometer-controller PLA. This view is the result of a functional-based decomposition, and it includes relations between different functional modules of the PL. The architectural components are large functional (domain) entities, and the connectors are *"uses", "command" or "passes-data-to"* relations. This structure is useful for understanding the interactions between entities in the problem space, for understanding the domain perspective, and hence thereafter, the possibilities for creating a PL.

The architectural elements are:

1. The Measurement Controller Subsystem (MCS), which has the main role in controlling acquisition and dumping science data. It requires services provided by other components of the system.
2. Housekeeping (HK) forms the housekeeping reports and sends them to the command interface when requested by command from the command interface subsystem. It uses services provided by PMS.
3. The Command Interface Subsystem (CIS) hides the hardware buses' interfaces from the rest of the software. It delivers a command from the interfaces to the appropriate subsystem and routes the response back to the command interface where the command originated from.
4. The On-board clock (OBC) maintains an on-board clock used for time-stamping spectra in data files. It includes services for timing the start/stop of spectra and targets. It also provides other timing-related services.
5. The Memory Management Subsystem (MMS) provides services for handling the storage RAM, program and parameter EEPROM areas, memory refreshment and memory error exception.
6. The Parameter Management Subsystem (PMS) provides services for initiating, changing and reading the on-board parameters in EEPROM.
7. The Start-up program (StartUp) implements the power up and watchdog timer start-up functions.
8. Communication buffer management (BUFMAN) provides services for allocating/de-allocating buffers.
9. CPU specific services (CPU) provide highly optimized high-speed assembly language services (high-speed word copy, interrupt enable/disable).
10. Hardware encapsulation modules provide low-level services for controlling specific hardware.

The operating system (OS) forms its own subsystem, which has not been included in the list because it is totally application independent. The OS has been encapsulated to provide an easy change of operating system if necessary.

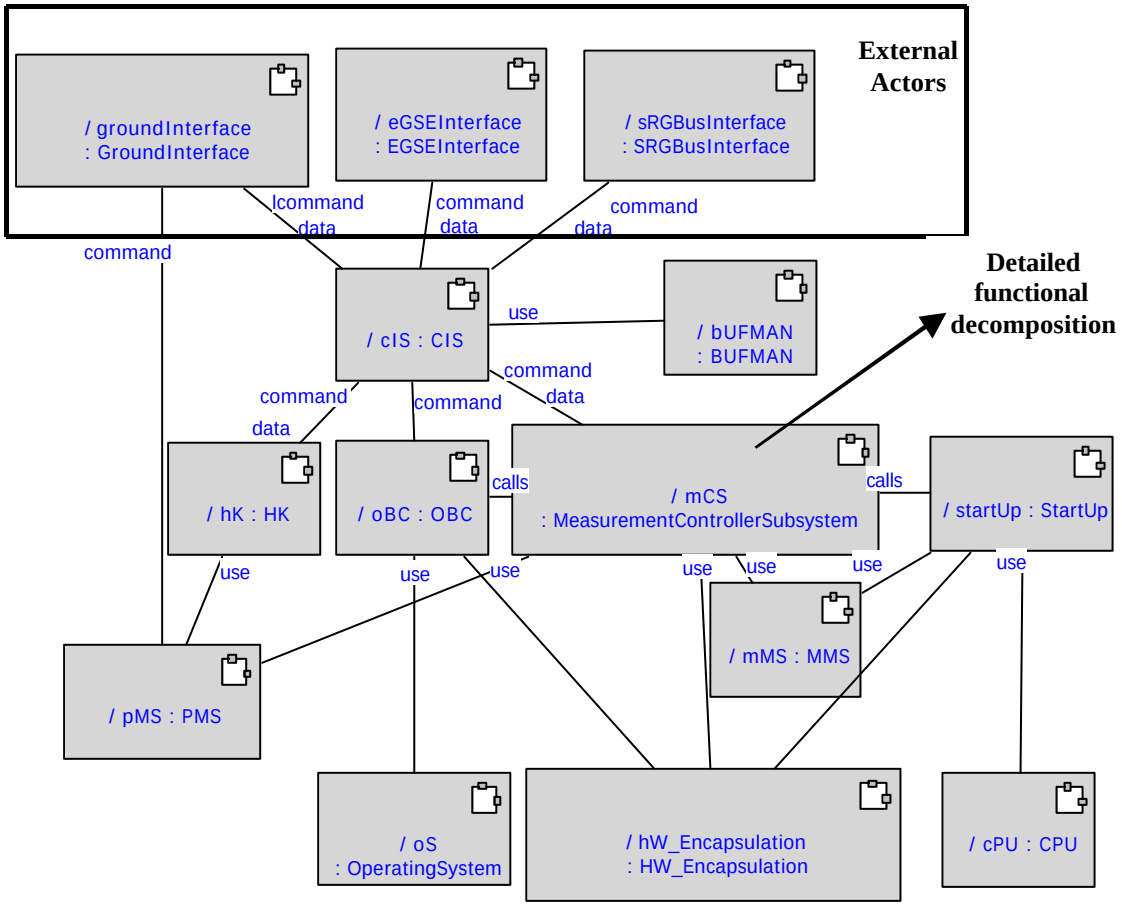

Figure 4. Conceptual view.

Detailed Functional Decomposition Structure. In a detailed functional decomposition structure the main elements are packages, components, ports and protocols. The static relations between components are association, specialization, generalization, etc. Considering the dynamic relations, statechart diagrams and

message-sequence charts (MSC) are also part of this view. In this view abstract components of the PLA are included.

- The MeasurementControl component, which is responsible for 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.
- The DataAcquisitionControl component collects events (science data) to the spectra data file during the observation of a target. The component includes as well as hides data acquisition details.
- Data File Management provides interfaces for storing science data - opening/closing/writing the data files, hiding data storing details. The component also provides interfaces for controlling the transmission of stored data to the command interface.

The Diversity View of PLA. The diversity view of PLA is introduced from the necessity to represent in one view the common and variable elements of different product members. This view is the one which makes the difference between PLA and the architecture of one product. This view must describe the stability of the PL domain, because the architecture is one of the reusable assets of the PL. Generally, the boundary of the PLA must be defined before the beginning of the activity of its detailed design. We can identify an inclusion relation between domain architecture, PLA and a single product architecture. PLA is included in the domain architecture and includes single product architecture. From this point of view PLA must represent part of the abstract features of the domain, but it should also be easy to identify in its views the concrete features of architecture of one PL member. The diversity view of PLA for the design of an on-board X-ray spectrometer control software is presented in Figure 5. This view associates the generic with the concrete. Looking top-down, the Abstract Spectrometer Features encapsulated in the measurement component are decomposed into three abstract components: MeasurementControl, DataAcquisitionControl and DataManagement.

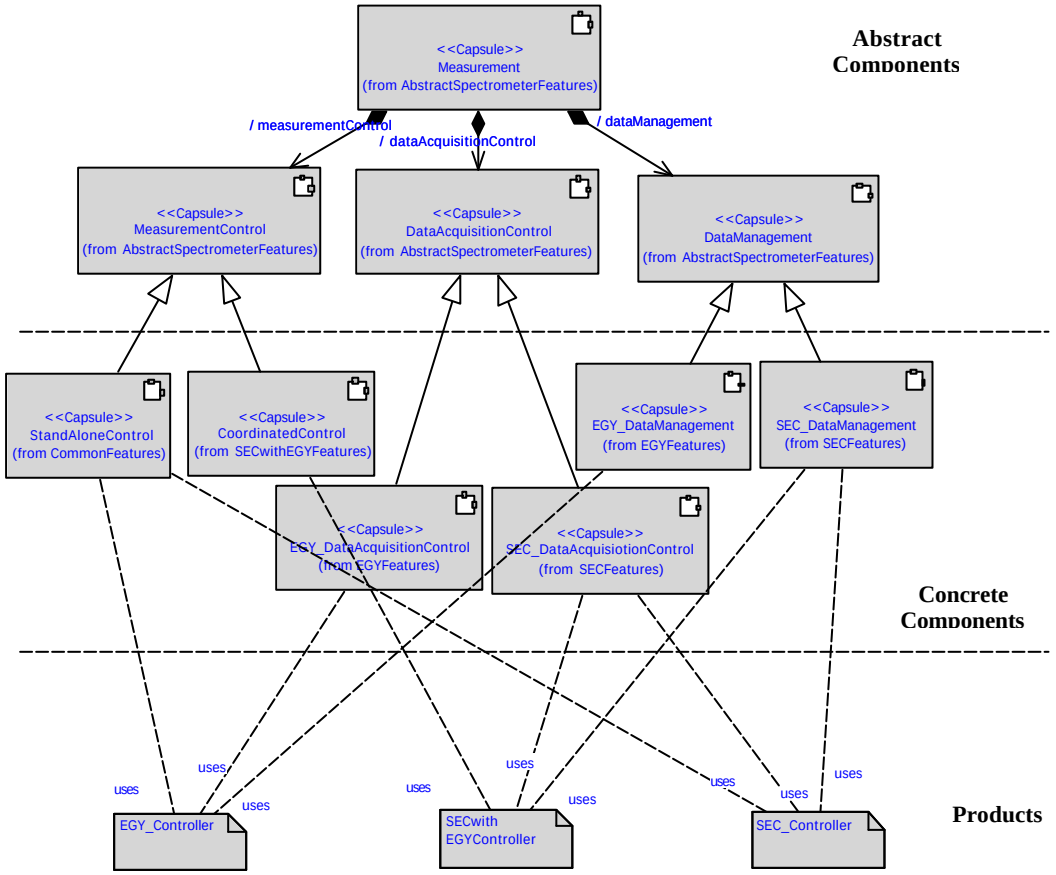

Figure 5. PLA Diversity view.

In each component, abstract features of the corresponding subdomains are collected, which are subsets of the Measurement domain abstract features. For each product which is member of the family, each of the three abstract components is specialized in a concrete component. For example, MeasurementControl is specialized in StandAloneControl and CoordinatedControl, DataAcquisitionControl is specialized in EGY_DataAcquisitionControl and SEC_DataAcquisitionControl, DataManagement is specialized in EGY_DataManagement and SEC_DataManagement.

The diagram includes the uses relation, which is directed from products to concrete components, providing in this way information on the reusability of each component. The same information could also be represented in tabular form, as in Table 2. The products of the group are vertically distributed and the components are dispersed horizontally. Each cell *(i, j)* of the table is marked if product P_i uses component C_j. For example, two products, EGYController and SECController, use the StandAloneControl component.

Table 2. Reusability of the concrete components in a PLA.

Component Product	StandAlone Control	Coordinated Control	EGY_Data Acquisition Control	SEC_Data Acquisition Control	EGY_Data Management	SEC_Data Management
EGY_ Controller	x		x		x	
SEC_ Controller	x			x		x
SECwithEGY Controller		x		x	x	x

5. Experiences with spectrometer controller PLA analysis

The potential risk of PL development is minimized if the interactions of quality attributes are considered in the analysis method [17]. Not only modifiability but also other structural or run-time quality attributes are important to PL software development. However, reusability and modifiability are the main quality drivers for PLA design. The analysis of these two quality attributes could be combined with other run-time quality requirements (performance, reliability, security, etc.) of the PL domain.

In the previous section we discussed PLA analysis strategy and we introduced in general terms a measurement instrument and a procedure with which to apply this instrument. The content of this section focuses on the specific reusability and modifiability properties of PLA and we shall present our experiences of theoretical thinking on the case study introduced previously.

Reusability. In [9] the reusability in software architecture development is divided into two categories. It is considered that reusability has two major aspects - software development *with* reuse and software development *for* reuse. The *with* reuse aspect requires the construction of software architecture that allows 'plug-in' prefabricated structures and code components. The system is composed of existing components by adapting them to the needs and implementing 'glue' components to connect them. If the *for* reuse aspect is considered, the produced components are potentially reusable in future projects as part of the current software development. Parts of the software system under development are taken and reused in other systems without modification.

Reusability Strategy. One important benefit from the PL approach is reusability. A reuse-based software development process requires specific methods adapted for this context. Reusability is a difficult property to assess. This quality attribute could be seen as a balance between generality and specifics and is the main driver for PL development. First the architecture and its components should be general because they should be applied in other similar situations. Secondly, the architecture should provide concrete functionality that supplies considerable advantage when reused. Considering the aspect of time we can identify two important parts in the life cycle of a PL. One is the initiation moment of the PLA and the other is its evolution. PLA is initiated from a set of existing product features and the variability in space is present. The other part is the evolution, where variability in space is multiplied by the variability in time.

The product architecture is analyzed by asking the typical question for reusability: *'How much of the software can be reused?'* in the context of each reuse scenario. The effect on the architecture is evaluated in a statistical manner. This means that every scenario is assigned a quote number of affected components in the scenario,

divided by the total number of components in the current architecture. The result should be as close as possible to one (as many of the components as possible should be reusable *as-is*). A PL needs a strategic method. The method must not consider simply the reusability of a single product and then re-engineer its architecture for each other products. It is also recognized that better results are obtained if one quality attribute is not analyzed in isolation, but its interactions with other quality attributes are considered. Furthermore, the PL concept brings about new aspects, such as commonality and variability, and their interaction should be considered, too.

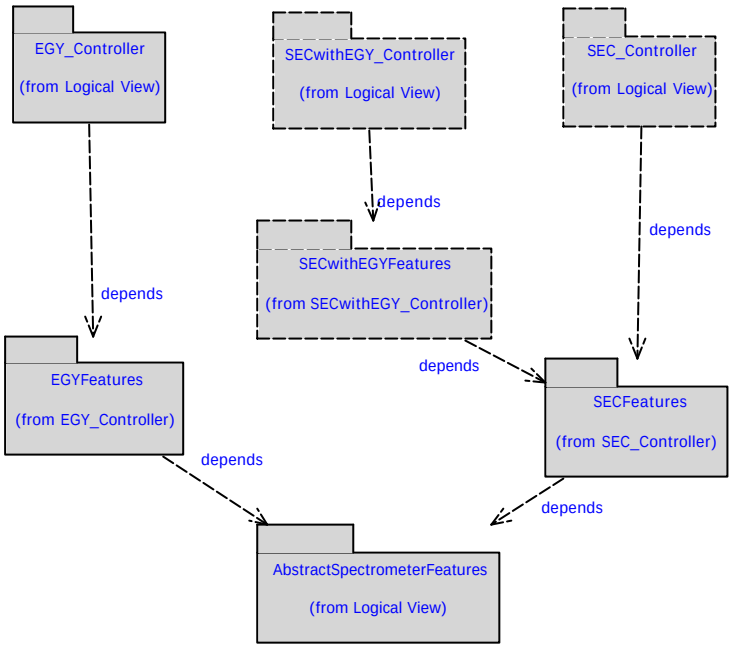

Figure 6. Mapping the product features into packages.

Our strategy could be applied for the initiation of PLA. The technique is based on the abstract features of all the considered products. To achieve both precision and abstractness we specify the reusability aspects in terms of groups. A reusability *with* reuse is a commonality held uniformly across the given group of products of the PL. A reusability *for* reuse is a variability true of only some members of the PL. In our opinion the reusability aspects are associated with the commonality and variability analysis method SCV [10]. In PLA, both aspects of reusability are present. The diversity view of PLA presents the '*with* reuse' aspect in the concrete components used by each of the product members. In the diversity view diagram the number of the 'uses' relations associated to one concrete component measures the degree of reusability in the PL of that component. For example, the degree of reusability for the SEC_DataAcquisition component is 2 and for Coordinated_Control it is 1. The *with* reuse aspect of reusability is described in the PLA by the abstract features of the PLA. The abstract features encapsulated in three main abstract components MeasurementController, DataManagement and DataAcquisition, are completely reused in all the product members.

The PLA diversity view contains both aspects regarding reusability encapsulated in components. From the point of view of other reusable assets, which could be represented by an architecture view, the organization in packages of features could be a possible solution. Figure 6 describes the mapping of product features into packages and the relations between packages for the spectrometer controller PL. The AbstractSpectrometerFeatures package has the highest degree of reusability, but also the highest degree of dependability. The abstract features depend on the commonality between EGY and SEC features. A change in the problem domain of one of the three products is mostly reflected in the degree of reusability of the abstract domain features.

Modifiability. Modifiability is one of the important structural requirements for PLA. According to [3], modifiability is the ability to make changes quickly and cost effectively. Modifications to a system can be categorized into:

- *Extensibility*, which is the ability to acquire new features,
- *Deleting unwanted capabilities*, to simplify the functionality of an existing application,

- *Portability*, for adapting to new operating environments, and
- *Restructuring*, which means rationalizing system services, modularizing, and/or creating reusable components.

However, each of these categories could be analyzed separately if its identity has been established in a set of well-defined sub-characteristics, possibly in a quality model form.

Modifiability Strategy. Scenario-based assessment is particularly appropriate for qualities related to software development, which are specific to PLA. Software qualities such as maintainability, reusability, modifiability, adaptability and portability can be expressed very naturally through change scenarios. However, scenario-based evaluation depends on the objectivity and creativity of the analyst who defines and executes them. In [1] the use of scenarios for evaluating architectures is recommended as one of the best industrial practices. Our strategy is based on the SAAM, but improved through the introduction of guidelines for PLA analysis. Our method considers PL-specific techniques such as commonality analysis, which systematically models the required similarities and differences among PL members. The analysis method consists of five important steps, which are: *deriving of change categories from the problem domain, scenario identification, PLA description, evaluation of the effect of the scenarios on the architecture elements, and scenario interaction.*

We defined twelve change scenarios for different categories of changes. The effect of the scenarios and the required PLA view are summarized in [13]. A good architecture design must provide a good localization of changes. Most of the changes required by scenarios were applied to one component, which indicates a good decoupling of concerns. By structuring the PLA in abstract components, which encapsulate common features of the PL members and concrete components, which in turn represents specialization of the variable features, the effects of the change scenarios are minimized and localized.

For the moment, only scenarios could be used in PLA analysis for modifiability. One problem with scenario-based analysis is that the result and the expressiveness of the analysis are dependent on the selection of the scenarios and their relevance for identifying critical assumptions and weaknesses in the architecture. There is no fixed minimum number of scenarios whose evaluation guarantees that the analysis is meaningful. According to this, we tried to use five categories of possible changes in general hardware, specific hardware, functionality, non-functional requirements and software technology. A helpful strategy in scenario elicitation is the analysis of commonality and variability. This is not a part of the architecture analysis method, but it is considered a pre-condition of it. One aim of the analysis should be to show how flexible PLA is in order to handle the anticipated changes provided by the variability of the products. Another aim is to analyze, what is the potential of the PLA to be adapted to changes in common features.

The change scenarios did not affect the PLA diversity view. This view confirms the stability of the PLA in the domain, delimited by the considered product members. The diversity view of the PLA is useful for reusability analysis, but it is not recommended for modifiability. The results of the analysis depend not only on the views of the architecture, but also on the level of detail of the component descriptions. By using only the conceptual view the effects of the change scenarios are reduced. On the detailed functional decomposition view, which has been developed with the help of a CASE tool, the effect is more relevant. The interaction of unrelated scenarios is lower and reveals a good separation of concerns when the functional decomposition is detailed.

5. Conclusions about our practice with analysis of product line architecture

Our practice presented in this paper identifies and demonstrates a method for analysis of the architecture of a PL initialized in a revolutionary approach. The analysis strategy based on scenarios could be considered mature enough to verify PLA against anticipated changes in requirements of various product members. In addition, the diversity view introduced for reusability considerations indicates in a structured form the reusability degree of each component for each product in the PLA. This view is a stable element for the PLA in the domain.

Our experience shows that the PLA analysis method should be applied iteratively, while the PLA design becomes more detailed. The purpose of the evaluation is to analyze the architecture to identify potential risks, by predicting the quality of the PL before it has been built. Iterative methods promote analysis at multiple resolutions as a means of minimizing risk to acceptable levels of time and effort. Areas of high risk are analyzed more profoundly (simulated, modeled or prototyped) than the rest of the architecture. Each level of analysis helps determine, where to analyze more deeply in the next iteration.

The case study also requires performance, safety and reliability for each product. These quality requirements are common features for all products, but other PL domains could include variability in these aspects. These

variable features must be considered from the PLA design perspective, in order to minimize the risk that the final software products do not conform to these quality attributes. For PLA evaluation these aspects are still an open question. It is important to estimate what is the degree of reuse of PLA and what are the reusable assets, when the variability of these run-time qualities is considered.

Acknowledgement

We gratefully acknowledge Tuomas Ihme from VTT Electronics, Oulu, Finland, who has provided us an initial model of a spectrometer controllers family.

References

1. G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northop, A. Zaremski. *Recommended Best Industrial Practice for Software Architecture Evaluation*. CMU/SEI-96-TR-025, 1997.
2. P. America, H. Obbink, R. van Ommering, F. van der Linden, *CoPAM: A Component-Oriented Platform Architecting Method for Product Family Engineering*. Software Product Lines - Experience and Research Directions, Procs. of the First Software Product Lines Conference, Kluwer Academic, pp. 167-181, 2000.
3. L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, Addison-Wesley Ed., 1998.
4. PO. Bengtsson, J. Bosch, *Scenario-based Architecture Reengineering*, Proceedings of the 5th International Conference on Software Reuse (ICSR5), 1998.
5. PO. Bengtsson, J. Bosch. *Architecture Level Prediction of Software Maintenance*. Proceedings of Third European Conference on Software Maintenance and Reengineering, Amsterdam, Netherlands, pp. 139-147, March, 1999.
6. J. Bosch, *Evolution and Composition of Reusable Assets in Product-Line Architectures: A Case Study*, Procs of the First Working IFIP Conference on Software Architecture, February 1999.
7. J. Bosch, *Design and Use of Software Architectures - Adopting and Evolving a Product-Line Approach*, Addison Wesley, ISBN 0-201-67494-7, 2000.
8. L. Bratthall, P. Runeson, *A taxonomy of Orthogonal Properties of Software architecture*, Proceedings of the Second Nordic Software Architecture Workshop (NOSA'99), 1999.
9. F. Buschmann, R. Meunier, P. Sommerland, M. Stal. *Pattern-oriented Software Architectures, a System of Patterns*. Chichester: Wiley & Sons, 1996.
10. J. Coplien, D. Hoffman, D. Weiss, *Commonality and Variability in Software Engineering*, in IEEE Software (November/December 1998), pp. 37-45.
11. R. Day, *Quality Function Deployment. Linking a Company With Its Customers*, Milwaukee, Wisconsin: ASQC Quality Press, ISBN 0-8739-202, 1993.
12. L. Dobrica, E. Niemelä, *A Survey on Software Architecture Analysis Methods*, submitted to IEEE Trans. on Soft. Eng. Journal, 2000.
13. L. Dobrica, E. Niemelä, *A strategy for Analysing Product Line Software Architectures*, Research Report, VTT Publications, 2000.
14. T. Ihme, *A ROOM Framework for the spectrometer Controller Product Line*, in Proceedings of Workshop on Object technology for Product Line Architecture, pp. 119-128, ESI-199-TR-034.
15. R. Kazman, M. Klein, M. Barbacci, H. Lipson, T. Longstaff, S. J. Carrière, *The Architecture Tradeoff Analysis Method*, Proceedings of ICECCS, Monterey, CA, 1998.
16. R. Kazman, G. Abowd, L. Bass, P. Clements, *Scenario-based Analysis of Software Architecture*, IEEE Software,pp. 47-55, Nov. 1996.
17. T. Kishi and N. Noda, *Aspect-Oriented Analysis for Product Line Architecture*, in Software Product Lines - Experience and Research Directions, Procs. of the First Software Product Lines Conference, Kluwer Academic Publishers, pp. 135-147, 2000.
18. P. B. Krutchen, *The 4+1 View Model of Architecture*, IEEE Software,pp. 42-50, Nov. 1995.
19. P. Lalanda, J. Bosch, R. Lerchundi, S. Clerki, *Object Technology for Product-Line Architectures*, ECOOP'99 Workshops, LNCS 1743, Springer-Verlag, Berlin, 1999.