The Progress Process Guidelines (PPG)

Rikard Land, Jan Carlson, Stig Larsson, Ivica Crnkovic

Mälardalen University, School of Innovation, Design and Engineering
PO Box 883, SE-721 23 Västerås, Sweden
+46 21 10 70 35

{rikard.land, jan.carlson, stig.larsson, ivica.crnkovic}@mdh.se

Abstract

This report proposes how the emerging model-driven and component-based paradigms can be utilized in embedded systems development to achieve a potentially high level of project monitoring and control, and thus reduce project risks. The guidelines are formulated as an extension of CMMI.

Keywords

Component-based development, Component-Based Software Engineering, Model-based development, Model-driven development, Embedded systems development, Guidelines, CMMI.

MRTC report, ISSN: 1404-3041, ISRN: MDH-MRTC-247/2010-1-SE

Mälardalen Real-Time Research Centre, Mälardalen University, September 2010

Table of Contents

PART I: Introduction and Example

Introduction

Model-driven development (MDD) [1] and Component-Based Software Engineering (CBSE) [2] are two approaches to software and systems development which have matured, and are increasingly used in combination. Usually, the benefits of these approaches are formulated in technical terminology, such as the software being correct by construction, possibility to predict the behavior of a system built from components with known properties [3], and ensuring consistence and removing some potential sources of introducing errors into the development process. There are only few examples of presentations connecting these technical advances to processes [4][5][6].

One important way to lift these technology-based benefits to a project/organization level is to investigate how these paradigms support risk reduction and a means for project monitoring and control. This has been summarized as the *CARMA* principle, as formulated in [7]:

- **Components.** Choose a component technology which supports compositional reasoning of component properties. As early as possible, define the components of your system (i.e. the architectural structure.)
- **Attributes.** Keep track of the attributes, i.e., properties of (components of) your system through component attributes. Use a tool that supports management of these properties, including automatic composition.
- **Requirements.** Refine your high-level system requirements into product requirements, and specify these in terms of the attributes which are analyzable with (tools supporting the) composition theories.
- **Milestones.** Formulate milestones (e.g. project gates) in terms of tuples: <expected value in relation to product requirement; method to generate this value>.
- **Analysis.** The verification analysis is performed at these milestones. In addition, the individual developers, architects, project manager, etc., may perform analyses of interest at any time; this resembles debugging in direct connection to implementation, which is informally done (i.e. not

mandated by a formal process) but an invaluable tool for the individual developer before passing the code (or, in model-driven development, the model) on to verification as part of the formal process.

This report describes a set of guidelines intended to make this principle concrete. The principle itself, and accompanying examples, have been published previously [7][8].

The guidelines have been given the name Progress Process Guidelines (PPG), after the PROGRESS Centre for Predictable Embedded Software Systems[1] where this research is being carried out.

The PPG structure is based on CMMI [9], but is not limited to organizations explicitly using CMMI (as clarified in section “Research Method” on page 8). Similarly, the guidelines stem from the methods, technologies, and tools developed at the PROGRESS centre, but are not limited to these (as described in section “Scope and Assumptions” below). Nevertheless, the PPG contains some references to ProCom, the specific component model developed at Progress, when ProCom provides specific and unique features. The first part of this report describes a conceptual product meta-model, where concepts and their relationships are defined which are used to specify the actual guidelines in the rest of the paper, and illustrates the intended use of PPG with an illustrative example on page 15. The second part of the report, from page 25 onward, is organized according to CMMI process areas, in alphabetical order, and extends it as described in section “Structure of the Guidelines” on page 10f.

Whether the CARMA principle and its interpretation PPG are useful in practice, is the topic for several current and future studies [7]. The present report can be seen as a support for these other empirical studies, intended to clarify the CARMA principle in detail within a well known terminology and structure. Other studies include process simulation, tool implementation, and pilot projects [7].

Introductory Example

A simple explanatory example of these concepts is presented in Figure 1. There may be a (product) requirement on the system’s response time, i.e. the total time from an external event occurs until an actuator is actuated. The system’s response time is dependent on (at least):

- Component interconnections (arrows in the figure)
- Each component’s internal execution time (in some appropriate unit independent of hardware; e.g. execution paths or some other behavioral model, ideally expressed in terms of input)
- Allocation of components to various hardware nodes (dashed arrows)
- Characteristics of the hardware (e.g. processor speed)
- The scheduling of the components

[1] See http://www.mrtc.mdh.se/progress

Similarly, memory usage, usage of other resources, and the overall functional behavior of the system, can in principle be specified as a composition of the ingoing component's properties (memory usage, behavior, etc.). However, to achieve reliable as well as useful results, there must exist reliable and practical methods used to generate component information as well as for aggregating the relevant component information. In practice, execution time analysis is very time-consuming, and often provides overly pessimistic results; only some of the most important parameters are listed above and in the figure.

The scope of PPG is, when a Model- and Component-based approach is suitable, and when proper tools and methods for development and analysis are available, to explain how to leverage on the potential benefits in terms of project management and control, and risk reduction in the project.

Details about the ProCom component model, as an example of a component model supporting this reasoning, can be found in [10][11], and details about the attribute framework in ProCom is provided in [12]. Other similar component models and UML extensions are: AADL [13], Autosar[2], SysML[3] and MARTE[4]; the applicability of PPG to these has not been investigated.

Scope and Assumptions

The PPG covers some parts of several process areas, but is intended to provide guidelines only for the parts that are related to working efficiently according to the CARMA principle. In particular, the PPG concerns only (in CMMI terms) *product requirements*, i.e. technically oriented versions of the requirements, and not *customer requirements*, which express customer needs. Nor does PPG concern how product requirements are derived from customer requirements.

These guidelines fundamentally builds on the assumption that a component technology is available so that for each component attribute of interest (e.g. behavior, timing properties, memory usage) there exist a way to analyze a primitive component, as well as a composition theory or analysis technology [14].

[2] See http://www.autosar.org/

[3] See http://www.sysml.org/

[4] See http://www.omgmarte.org/

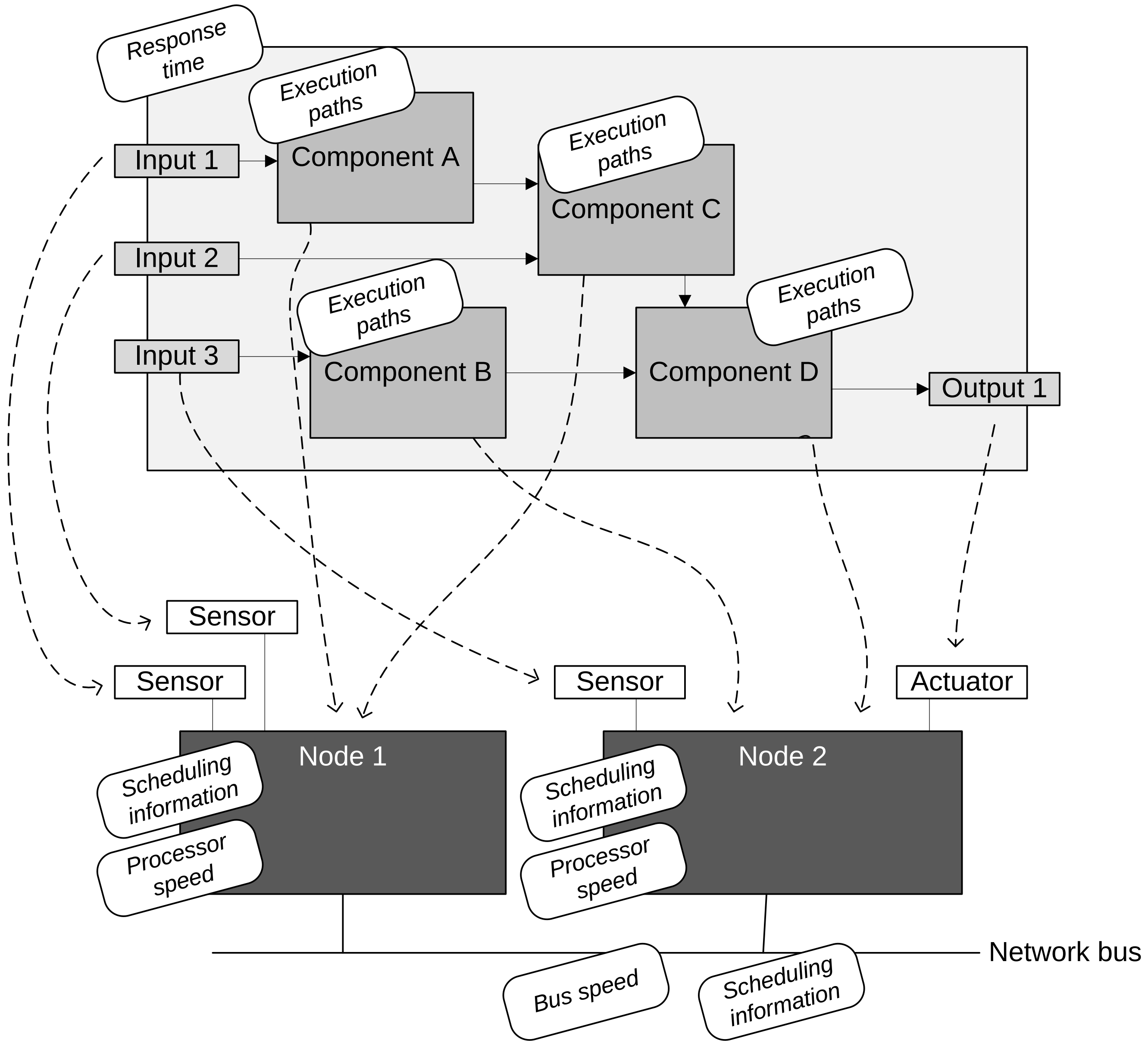

Figure 1: Example of aggregating component attributes. The upper part of the figure describes the software components, and the dashed lines show allocation to hardware elements at the bottom of the figure.

Research Method

The PPG intends to describe a set of guidelines describing what is believed to be a good match between state-of-the-art research which has not yet found its way into a complete and integrated tool suite, and current practice in industry. This restricts the possibilities for empirical studies of industrial best practices, instead we are performing a set of complementary studies of various kinds, where the current report intends to set a baseline in a well-known terminology (CMMI) which can be related to. Other studies include process simulation, tool implementation, and hopefully pilot studies in a limited setting [7].

The input to these guidelines comes from the following sources:

- **Literature.** Study of literature on model-based and component-based development processes and roles [8]. This has led to the formulation of the CARMA principle [7]. The section "Related Work" below describes the related literature.
- **Interviews with researchers.** Several rounds of open-ended interviews were performed with PROGRESS researchers, in order to collect and understand the intended usage of various state-of-the-art methods, tools, and not-yet fully implemented ongoing research. This study consisted of some 20 formal interviews with a dozen researchers (in addition to other collaboration at various levels).
- **Industrial requirements specifications.** Industrial requirements specifications, architecture documents, and verification documents from several embedded system development projects have been specifically studied in order to achieve relevance[5]. The general conclusions are presented in section "Requirements in Industry Today" on page 10 below.
- **CMMI** [9] provides the PPG structure. Each process area, goal, etc. has been studied from the point of view of where and how each part of the conceptual product meta-model (defined and described on page 11ff) should be instantiated, and used. (For example, requirements development in the CMMI process area "Requirements Development" involves defining product component *attributes* with *required values*.)

Related Work

The principle behind Model-Driven Development (MDD) is to bridge the gap between various development artifacts such as requirements, architectural descriptions, lower-level designs, and implementation level [1][15][16]. MDD implicitly makes the development process more efficient

[5] These specifications were studied particularly for this purpose, in addition to general knowledge and experience from other projects and documentation of relevant systems. We are not allowed to share further information about these requirements specifications.

through automatic or semi-automatic model transformations, and the likelihood of errors being introduced is decreased. OMG's Model-Driven Architecture (MDA)[6] is one important instantiation of this principle, where the main objective is to achieve platform independence [16].

In the Component-Based Software Engineering (CBSE) paradigm [2], software systems are constructed from smaller-grained components, which only interact at well-specified interfaces, and are to a high degree self-contained and thus often reusable in several systems. One important goal for CBSE is to enable composition of component properties into higher level (e.g. system) properties [3][14]. However, component models and component technologies often regard a "component" mainly as a deployable entity [2], while the PPG focuses on the concept of *component identity* throughout the process, from early design to run-time. This is true for component models designed to also adopt the Model-Driven Development paradigm, and modeling languages which have adopted the CBSE paradigm, such as the *ProCom* component model [10][11] (which is the component model the PPG explicitly builds on), AADL [13], Autosar[7], SysML[8], MARTE[9], and EAST-ADL [4][17].

The exploratory analysis and milestone verification presented in this paper inherits the basic ideas from the concepts of daily builds, continuous integration, continuous verification, and test-driven development [18][19][20], and adapts them to fit the component-based approach. Other, parallel research efforts have described similar, although more high-level, guidelines for early verification based on the combined CBSE/MDD paradigm: the "methodology" of the TIMMO project [4] acknowledges that early estimates may be combined with results achieved with e.g. analysis tools; the SAVI project has also noted the possibilities of performing a "virtual integration" [5][6] early in the process. This report has developed and generalized this notion further, and describes the process explicitly as a reference text.

Descriptions of processes utilizing the advanced concepts of the CBSE paradigm (e.g. composition of component attributes throughout the process) are scarce [4], and are usually at a more abstract level [21]. In the field of MDD, process descriptions focus more on the division into platform development and application development [15][16], and the new roles required for this, such as a meta-team of language developers and code generator developers [22][23].

[6] See http://www.omg.org/mda

[7] See http://www.autosar.org/

[8] See http://www.sysml.org/

[9] See http://www.omgmarte.org/

Requirements in Industry Today

According to our investigation and experience, requirements specifications today contain a mix of what CMMI [9] labels *product requirements*, i.e. requirements specified in enough detail to allow specific pass/fail criteria to be specified [7]. The requirements specifications also describe *customer requirements*, i.e. the goals for the customer which is fairly imprecisely specified and difficult to verify unambiguously (such as “it shall be easy to upload a new software version”). The PPG focuses on product requirements, and for these, we have noted the following, which is relevant for the PPG and conceptual product meta-model described in section “Conceptual Product Meta-Model” on page 11:

- Many product requirements are specified in terms of execution steps, sometimes using some kind of dynamic diagram. Example: “During startup, register X shall first be read to determine the cause of the last shutdown/reset. If it is…”
- Some product requirements describe timing behavior of some execution steps. “Example: The first phase of startup shall take less than X ms; the second step Y ms; …”
- Some product requirements, but not many, are hardware specifications. Example: “The processor shall be of type X”; “the software image shall fit in X bytes of memory”.
- The requirements that are specified in the greatest detail (including the specifications on timing and resource usage as described in the bullets above), and are formulated to be unambiguous and verifiable, are the requirements that specify safety-related functions. This is due to the potentially catastrophic effects of a specification error, concretely meaning adhering to a safety standard, and passing the external assessment to achieve product certification.

Structure of the Guidelines

The structure of the guidelines (i.e. from page 25 onward) explicitly follows that of CMMI for development, version 1.2 [9]. This makes PPG easily accessible to readers familiar with CMMI, and allows the PPG to focus on the specifics of the component-based approach and the Progress methods and technologies. The PPG thus omits other, generally applicable guidelines and suggested practices. PPG should be perfectly applicable in organizations that do not explicitly follow or implement the CMMI; CMMI is only a choice for presentation of the PPG. This is similar to other CMMI extensions [24][25].

More specifically, the PPG is formulated as additions in the forms of *notes*, *amplifications*, *typical work products*, *references*, and/or *amplifications* [9]. (For the CMMI process areas (or specific goals, or specific practices) where PPG makes no additions, this is explicitly stated in the present report with “nothing added”, preserving the CMMI structure.)

Figure 2 describes schematically how the PPG adds to some elements of CMMI which are used to design a concrete organizational process (depicted in green):

- The contents of the CMMI [9] itself (in blue).
- The CARMA principles (in pink), which have been interpreted when constructing the PPG.
- The contents of PPG (in yellow). Each addition may be an addition of either a CMMI *process area*, a *specific goal*, or a *specific practice*.

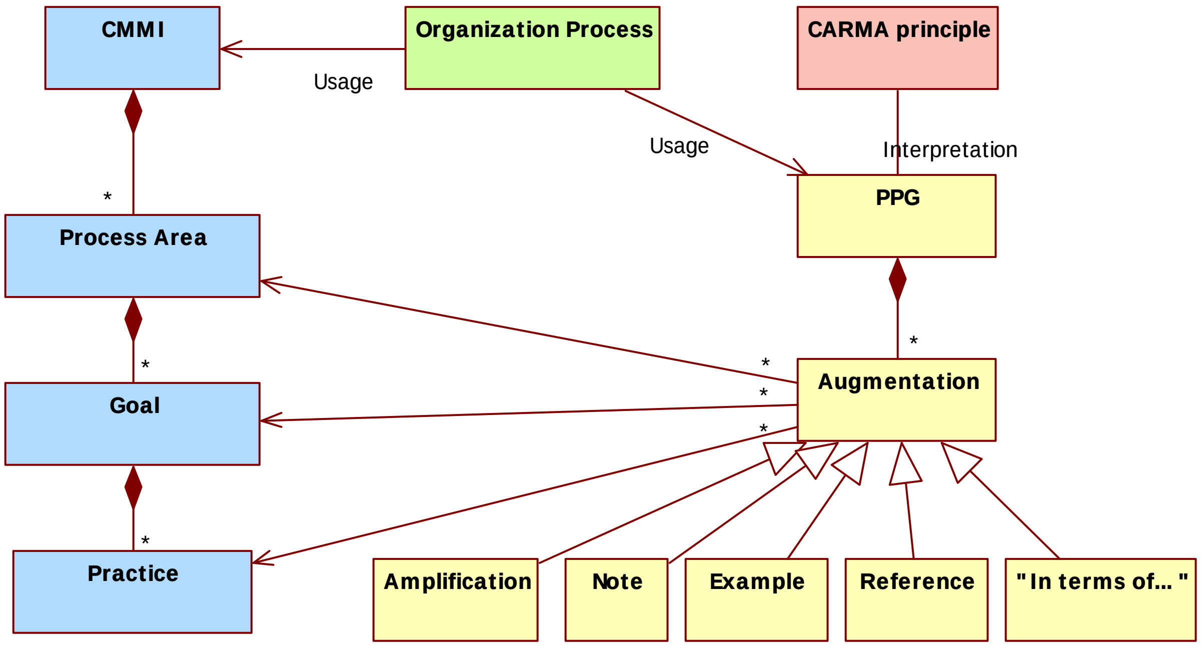

Figure 2: PPG in relation to CMMI and concrete organization processes.

Conceptual Product Meta-Model

PPG defines a conceptual product meta-model of the information needed; see Figure 3. This meta-model is explicitly referred to from the PPG additions, and would form the data structure of tools to support the processes[10]. It is consistent with the ProCom model, but simplifies it where appropriate in order to presenting the PPG as concise as possible, and to not be unnecessarily restricted to the ProCom component model. For example, in ProCom there are different types of components, which is

[10] Which is also part of our research agenda, see [7] for more details.

not significant for PPG. Main references to ProCom are: the foundations of version 1.0 in [11], the attributes in [12], and the elaboration of allocation to virtual and physical nodes in [26].

The prefixes within parenthesis in the figure are references to the CMMI process areas where the concept is created. For example, creation of *Components* is the responsibility of the Technical Solution (TS) process area, while *Milestones* are created as part of Project Planning (PP).

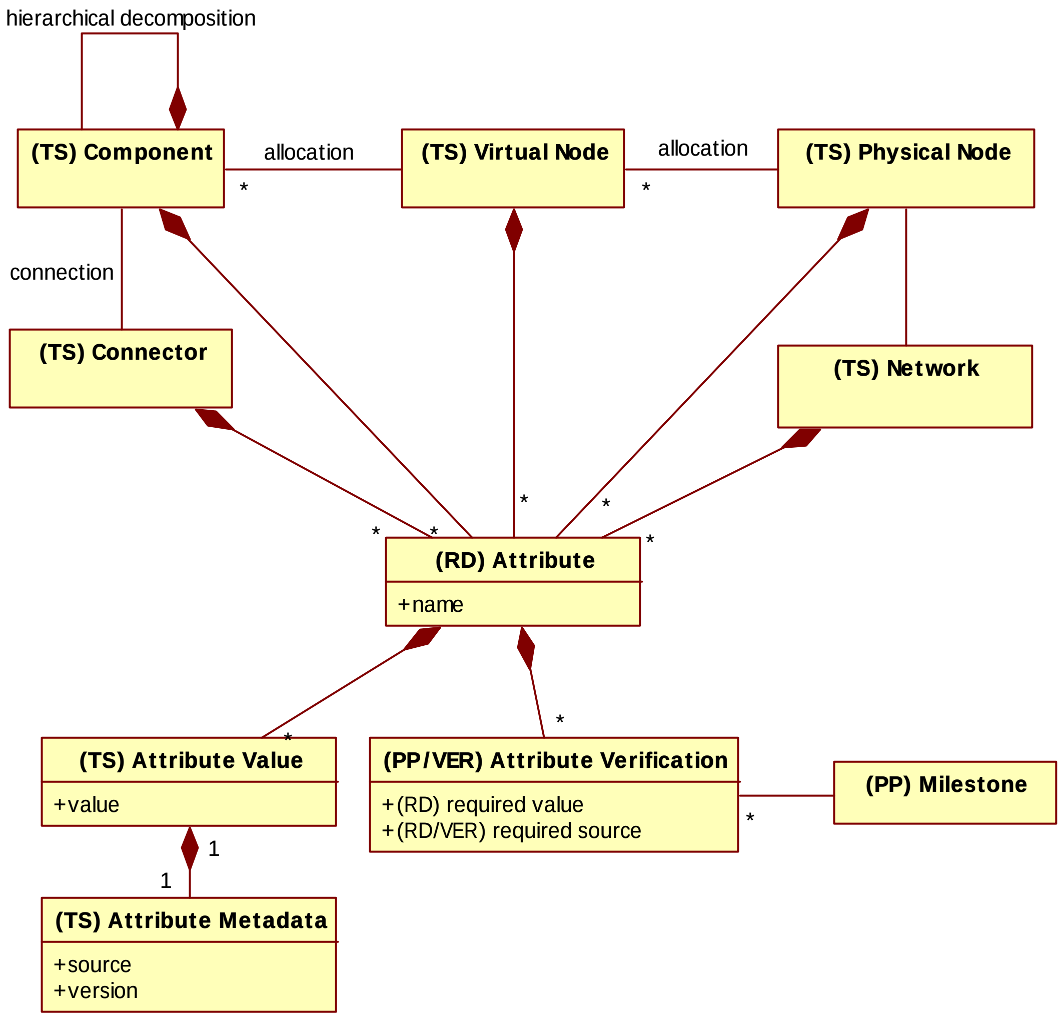

Figure 3: The PPG conceptual product model.

The basic building blocks of functionality are software ***Components***. *Components* may be constructed hierarchically from smaller components, thus the *hierarchical composition* self-relation. The system can be considered as the top-level component.

Components are connected with ***Connectors***. Mostly, connectors only describe how output from one component is used by another component forward data, but they may also be more complex and e.g. split one input into two outputs, merge several inputs, filter, buffer, or log the data on the channel.

Physical Nodes are models of the physical hardware nodes, connected with one or more ***Networks***.

As a level of abstraction between *Physical Nodes* and *Components*, ProCom introduces ***Virtual Nodes***, which are largely self-sufficient packages with e.g. their own internal scheduling, and are thus analyzable in isolation (to a certain extent), and are reusable entities.

Components and *Connectors*, *Virtual Nodes*, and *Physical Nodes* can all have any number of ***Attributes***. As described in section "Introductory Example" (page 5), in order to monitor the system requirement of an end-to-end response time, as the measured delay from sensor input to actuator output, relevant attributes to keep track of throughout the development could be: for *Components*, worst-case execution time (or average-case execution time, or whatever metric is most suitable for the definition of the requirement); for *Virtual Nodes*, resources such as memory or processor share, in order to postpone deployment decisions and make a subsystem reusable; for Physical Nodes, the processor speed.

An *Attribute* (say, "worst-case execution time") may have several ***Attribute Values***, with an actual ***value*** in an appropriate unit (say, 150 μs) and associated ***Attribute Metadata*** describing mainly the *source* of the value (e.g. static analysis, or measured on specific hardware) and the *version* of the owning entity (e.g. *Component*) the *value* refers to. Throughout the development, several *Attribute Values* may be created and continue to live side by side, used for different purposes. For example, in the middle of the project, the *Attribute* "worst-case execution time" for a *Component* may have the following *Attribute Values* associated to it: 160 μs for the current implementation, guaranteed by a static analysis tool (assuming some specific hardware); 125 μs, measured on the previous product (which used a slower processor); and 150 μs, estimated by an expert (for the new input domain and target hardware). The reason to maintain several *Attribute Values* for the same *Attribute* is that each has its particular usage and limitations: the value from static analysis is exact but parameterized (depending on the clock speed of the processor) and overly pessimistic for some purposes; moreover, it is at this stage limited since it refers to an implementation which is not yet finished (more features will be added before the *Component* is finalized, likely to require more processor cycles). On the other hand, the measured value refers to hardware used in a previous product but not the current. Another useful type of attribute is an assigned budget, based on the system requirement broken down and assigned to components. Such a budget is typically assigned or approved by the component responsible. For simplicity, we have in the text discussed *Attributes* and *Attribute Values* represented by single values; in general the value may be a model of some kind: a single value for static (program) memory requirement, a statistical distribution of execution time, a state chart model of functional behavior. All this is further illustrated with an in-depth example in section "Example" below (page 15).

As indicated by the CARMA acronym, the added value in terms of project management comes from connecting the *Component Attributes* with Requirements and Milestones of PPG, and regularly or on-demand performing an Analysis of the current status of the system. This is achieved by the remaining concepts: ***Milestones*** are the planned checkpoints at specific points in time during the development, with associated ***Milestone Verifications***. A *Milestone Verification* describes what the milestone means in terms of requirement fulfillment, and consists of the ***required value*** and ***required source*** for that *Milestone*. For example, an early *Milestone Verification* may define a *required value* of 150 μs for the *Attribute* worst-case execution time, using expert estimate as a *required source*, while a later in the development, a later *Milestone Verification* may specify a static analysis tool as a *required source*, with a *required value* of 75 μs (i.e. less than the required value for the finished product, since at this stage the implementation will be only partially finished). The requirements specification for the product is modeled as a final *Milestone*, for which the *required value* is the target value of the product, and the *required source* should use the implementation as its basis. For example, static analysis and running an automated test suite may be appropriate methods, while an expert estimate is probably not a reliable enough method at this stage.

It has already been said that *values* for *Attribute Values* may be complex, which also applies to the *required values* of *Milestone Verification*. Moreover, the *required value* is a model describing what *values* are considered fulfilling the *required value*. In the simplest case, the *required value* may be a predicate using a simple operator; for example, the *required value* for "response time of the system" could be "less than or equal to 500 ms", or "within the interval [200, 600] ms". For more complex models, such as functional behavior, there must exist methods to decide whether e.g. a state machine model (of the implementation) fulfils the required behavior as specified by another state machine model.

The Interaction of Entities over Time

From PPGs point of view, it is important to note that at any given time during development, *Components* and *Connectors*, *Virtual Nodes*, and *Physical Nodes* may be created and developed independently. In order to finalize and deploy a system, *Components* are allocated to *Virtual Nodes*, and *Virtual Nodes* are allocated to *Physical Nodes*, but at an arbitrary point in time during development, any given *Component* may or may not be allocated to a *Virtual Node*, any given *Virtual Node* may or may not have *Components* allocated to it, and may or may not be allocated to a *Physical Node*, and any given *Physical Node* may or may not have *Virtual Nodes* allocated to it. A *Virtual Node* will "inherit" properties from the physical node it is allocated to, such as its processor speed, or available static or RAM memory, but it may also derive requirements on the allocation to *Physical Nodes* from *Attributes* of the *Components*, such as the required share of a processor in order to be able to meet its budgeted execution time.

The allocation, if automatic, is thus a complex optimization problem where *Virtual Nodes* are allocated to *Physical Nodes* and the connections between *Virtual Nodes* are resolved and scheduled to *Networks*, and the resulting system *Attributes* have to meet their *required values*. From a verification point of view, however, it is sufficient to ensure that all constraints are met by the allocation.

The Allocation of Entities to CMMI Process Areas

The PPG defines that each entity (and/or attribute of entities) of the conceptual product meta-model is under the responsibility of a certain CMMI process area. This is indicated in Figure 3 with prefixes within parentheses, and briefly described in the next paragraph. This is intended to serve as a guide into the second part of the report which follows the CMMI structure (i.e., from page 25 onward).

Components and *Connectors*, *Virtual Nodes*, *Physical Nodes*, and *Networks* are created as part of the Technical Solutions (TS) process area. The *Attributes* of interest are defined as part of the Requirements Development (RD) process area, but the *Attribute Verifications* are the responsibility of Project Planning (PP) and Verification (VER). The *required values* and *required sources* of the *Attribute Verifications* are the responsibility of Requirements Development (RD) to specify, partly with some planning and resources under the Verification (VER) process area. The *Attribute Values* and *Attribute Metadata* are a result of the product development, i.e. belong to the Technical Solutions process area.

Example

To explain the guidelines and the conceptual product model, we outline the development of the hypothetical system of Figure 4 and Figure 5, an electronic stability control system of a car. There are four “wheels speed” components as well as four “brake valves” components, of which only one of each is shown (FL = Front Left). During construction the analyses will help in identifying an allocation of software components of Figure 4 to the two nodes in Figure 5, taking into account scheduling of components on the CPUs, and of messages on the bus, so that all the requirements are satisfied (examples of timing and memory consumption given below).

Figure 4 was previously used as an example system in [8][11], here the example is extended. All numbers and other details are introduced for the purpose of illustration, and not to accurately model this kind of system.

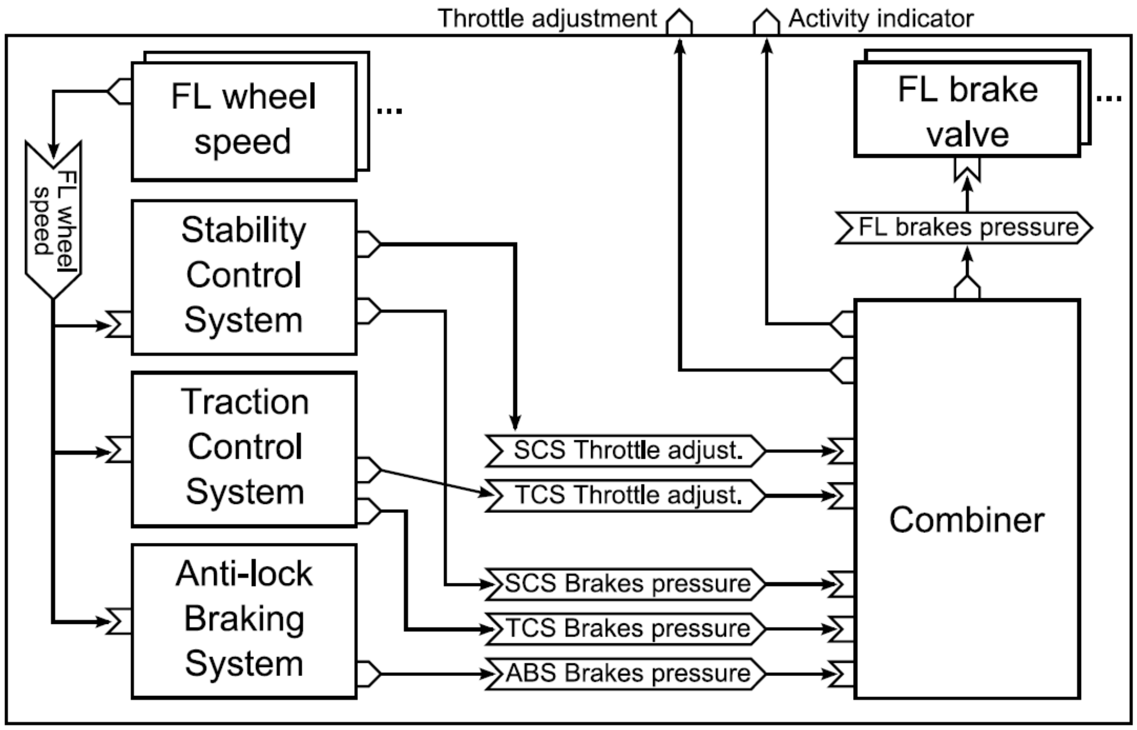

Figure 4: Software component design of an electronic stability control (ESC) subsystem of a car.

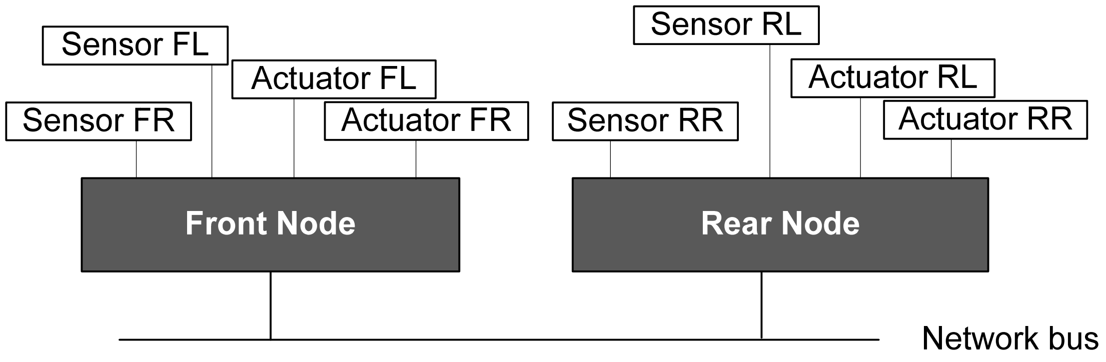

Figure 5: Hardware design of the same electronic stability control (ESC) subsystem. (FL = Front Left, FR = Front Right, RL = Rear Left, RR = Rear Right)

Requirements

Within the scope of the process area Requirements Development (RD), the customer requirement(s) are re-formulated as a number of product requirements, of which we use four as an example:

- **Functionality.** The intended behavior of the system is described as a timed automaton (not described here).
- **Static memory consumption.** The size of the software image is limited to 1024kb, given by the intended hardware. A small memory overrun would require significantly more expensive hardware.
- **Dynamic memory consumption.** Similarly, the maximum size of the dynamic memory (RAM) used during execution is limited to 512 kb, given by the intended hardware. A small memory overrun would require significantly more expensive hardware.
- **Response time.** The maximum response time of the system, i.e. the maximum allowed time from sensor input to actuator output, is given by experience from control engineers to 15 ms.

The requirements are specified in Table 1.

Table 1: Requirements of the example system.

Attribute Name	Required Value
Functionality	Behavior specified by a timed automaton (not shown here)
Static memory consumption	Max 1024 kb per node
Dynamic memory consumption	Max 512 kb per node
Response time	Max 15 ms

Also in this early phase, the structures depicted by Figure 4 and Figure 5 are essentially settled, as part of early project planning, including budgeting for hardware acquisition. The milestones described below refer back to both the requirements and this architecture.

Planning of Milestones

Within the scope of the process area Project Planning (PP), four milestones are defined in the project plan where the requirements will be followed up in *milestone verification*. Therefore this plan is done in cooperation with the process areas Verification (VER) and Requirements Development (RD).

- **MS1 (high-level design).** At this milestone, system functionality shall be broken down to functionality of individual components, and each component should also be assigned a budget for memory usage and timing requirements (and other requirements not discussed in the example). These component attribute *values* shall at this milestone be confirmed by the persons responsible for each subcomponent and when composed into system attributes they should meet the target criteria specified in Table 2.
- **MS2 (70% complete).** This milestone is some way into the development. Much, but not all, of the functionality will be available. Some components may have been selected from pre-existing components, such as existing parts of previous generations of systems, or COTS components offered by other vendors. At this point, it is still feasible to consider dropping some advanced features, and reconsider acquiring more powerful (and more expensive) hardware. Since everything is not implemented, the timing and memory footprint of the implementation will be analyzed with some swift methods, and the results should be ca 70% of the requirements on the final system (see Table 2).
- **MS3 (feature complete).** At this point in time, all features should be implemented, but there is still time after this milestone to fine tune e.g. control parameters, allocate software components differently, schedule bus messages and CPU tasks differently, and optimize the code if needed. As Table 2 shows, it is allowed that the current memory footprint and timing are slightly too large at this point in time, since from experience it is known that the code can be optimized with some percent thanks to optimization, re-allocation and re-scheduling. The strongest analysis methods are used to verify these properties, to avoid surprises at the milestone MS4 where these methods will be used.
- **MS4 (delivery).** This is the final milestone, where the system is finally verified. The target values equals the product requirements, and a combination of the strongest analysis methods are used to verify that the implementation meets these requirements.

Table 2: Milestones, with required values for the various system properties, in relation to required final values, and required source of these values.

Attribute Name	**Milestone ID**	**Required Value (brief explanation and/or motivation)**	**Required Source**
Functionality	MS1 (high-level design)	Behavior specified by a timed automaton, not shown here (Product requirement)	Manual Review

Attribute Name	Milestone ID	Required Value (brief explanation and/or motivation)	Required Source
	MS2 (70% complete)	Behavior specified by a timed automaton, not shown here (Product requirement)	Composition of design models where possible, manual review of other parts
	MS3 (feature complete)	Behavior specified by a timed automaton, not shown here (Product requirement)	Automatic analysis of implementation
	MS4 (delivery)	Behavior specified by a timed automaton, not shown here (Product requirement)	Automatic analysis of implementation
Static memory consumption	MS1 (high-level design)	Max 1024 kb per node (Product requirement)	Expert estimate of final system
	MS2 (70% complete)	Max 768 kb per node (Ca 70% of product requirement, since the code will continue to grow)	Compilation and synthesis of the existing parts
	MS3 (feature complete)	Max 1088 kb per node (Ca 105% of product requirement, since it is assumed that final optimization will reduce this value)	Compilation and synthesis
	MS4 (delivery)	Max 1024 kb per node (Product requirement)	Compilation and synthesis
Dynamic memory consumption	MS1 (high-level design)	Max 512 kb per node (Product requirement)	Expert estimate of final system
	MS2 (70% complete)	Max 350 kb per node (Ca 70% of product requirement, since the code will continue to grow)	Static analysis on some parts, (only partially implemented)
	MS3 (feature complete)	Max 560 kb per node (Ca 110% of product requirement, since it is assumed that final optimization will reduce this value)	Static analysis combined with measurements

Attribute Name	Milestone ID	Required Value (brief explanation and/or motivation)	Required Source
	MS4 (delivery)	Max 512 kb per node (Product requirement)	Static analysis combined with measurements
Response time	MS1 (high-level design)	Max 15 ms (Product requirement)	Expert estimate of final system
	MS2 (70% complete)	Max 12 ms (Ca 70% of product requirement plus some slack allowed for message passing on the bus)	Static analysis on some parts, (only partially implemented)
	MS3 (feature complete)	Max 17 ms (Ca 110% of product requirement, since it is assumed that final scheduling and allocation can solve this)	Static analysis combined with measurements
	MS4 (delivery)	Max 15 ms (Product requirement)	Static analysis combined with measurements

Milestone MS1 (high-level design)

At milestone *MS1 (high-level design)*, the overall software structure is defined and considered realistic (Figure 4). The hardware layout of the system is also defined (not shown in a figure, but similar to the lower part of Figure 1). The relevant system properties (i.e. the requirements on the system, as specified by Table 1) are apportioned to subcomponents. As an example, the second and third column of Table 3 shows the estimated dynamic memory usage and execution time of each component. (For simplicity, we only show two attributes of the four we have specified to follow up; the general description is identical for these as well.) With tool support, the components are allocated to either of the *Front Node* or *Rear Node* in a search for an allocation where all values for milestone verification at this milestone can be met. In the example, such an allocation was found, as described in the fourth column of Table 3, and the last row shows that the values of dynamic memory consumption and response time, aggregated from component attribute values, are within the criteria for this milestone (as specified in Table 2).

Table 3: Values for two attributes of the ESC subcomponents, at milestone MS1.

Component Name	**Dynamic memory consumption (Source)**	**Execution or response time (Source)**	**Allocation**
Wheels speed	48 kb (Expert estimate)	Execution time = 1 ms (Expert estimate)	One instance allocated to each node (need to be co-located with sensors)
Stability Control System	192 kb (Expert estimate)	Execution time = 4 ms (Expert estimate)	Front Node
Traction Control System	256 kb (Expert estimate)	Execution time = 6 ms (Expert estimate)	Rear Node
Anti-lock Braking System	96 kb (Expert estimate)	Execution time = 5 ms (Expert estimate)	Front Node
Combiner	48 kb (Expert estimate)	Execution time = 0.5 ms (Expert estimate)	Rear Node
Brake Valves	48 kb (Expert estimate)	Execution time = 1 ms (Expert estimate)	One instance allocated to each node (need to be co-located with actuator)
ESC	Front Node = 480 kb Rear Node = 496 kb (Composition of expert estimates)	Response time = 14.5 ms (critical path is 10.5 ms, plus 2 ms twice for network lag)	N/A

Milestone MS2 (70% complete)

The project progresses over time to milestone *MS2*. During this phase, some design and implementation decisions are being made:

- For the *Wheels speed* component, there are three potential COTS components available. These are investigated and one of these is chosen. If the component comes as a white box, i.e. with all desired information, including source code, it can be used for the static memory analysis required in some of the milestones. The same is true if the component is packaged as a black box but comes with models of its memory usage, which can be used in the composition theory. (This also requires there are certification mechanisms in place, so that the legal implications are clear regarding the extent to which the system development organization can trust these assertions.) In a less than ideal world, the component comes as a black box and with insufficient information, in which case the system development organization has to rely on thorough testing. In this case, the type of specification methods required in some milestones has to be re-negotiated – or, if measurements are not considered safe enough, the COTS will have to be replaced.

- The *Stability Control System*, *Traction Control System*, and *Combiner* require new development. Of these, *Stability Control System* is outsourced to a subcontractor, while the others are implemented internally. The *Combiner* is straightforward to implement, and implementation is finished within a month, well before milestone MS2.
- The *Anti-lock Braking System* and *Brake valves* will be reused from the previous generation of the car. No modifications are needed, other than those required to function in the ProCom environment. However, it may happen that the source code does not follow some required restrictions by the memory usage analysis tool, and for these components the verification method also has to be re-negotiated, with either the result that for example measurements is a qualified specification method, or that the source code has to be modified so as to fulfill the requirements of the analysis tool, or that the component will be re-implemented completely in ProCom (reusing the previous design). In the example, assume that *Brake valves* may be statically analyzed, while the *Anti-lock Braking System* cannot but will be wrapped as a ProCom component with as small modifications as possible nevertheless, and the project will therefore have to rely on measurements of this component.

During development, developers and the architect perform exploratory analysis whenever they need to explore some "what-if" scenarios (like "would it be fine to implement an algorithm that executes faster but requires more memory?") or to assure themselves that the current state of the work products will pass the next milestone.

The same procedure for milestone verification as for MS1 is repeated, however with the tools and methods required for this milestone (according to Table 2). We assume the software components are allocated to the same hardware nodes as for MS1. When compared with the target value for each attribute for this milestone (given by Table 2), it can be noticed that neither the memory limit (max 350 kb per node) nor the response time (12 ms) are met. This is an indication that the final product is running the risk of not meeting its requirements of 512 kb per node and 15 ms response time. There are a number of possible solutions to this:

- It may be decided to switch to more powerful hardware, i.e. faster and equipped with more memory. This solution has severe cost implications.

- Some feature, not yet implemented, may be dropped. For example, some complicated algorithm in the *Traction Control System* component may be replaced with a simpler one which will be faster and require less memory. This may have schedule implications.
- It may be noticed that some components are fully implemented, while others are slightly lagging behind schedule. When the target values for milestone MS2 were decided, it was assumed that the implementation would be ca 70% feature complete. If acknowledging that the implementation is somewhere around 80% feature complete, it may be decided that there is in reality a low risk for not meeting the target values for dynamic memory consumption and response time.

After discussions with relevant stakeholders, a decision selecting one of these options (or some other), and the project continues.

Table 4: Values for two attributes of the ESC subcomponents, at milestone MS2.

Component Name	**Dynamic memory consumption (Source)**	**Execution or response time (Source)**	**Remarks**
Wheels speed	47 kb (Measurement)	Execution time = 1 ms (Measurement)	COTS
Stability Control System	98 kb (Static analysis)	Execution time = 4 ms (Static analysis)	Partially implemented, according to plan
Traction Control System	124 kb (Static analysis)	Execution time = 3 ms (Static analysis)	Partially implemented, lagging behind plan
Anti-lock Braking System	97 kb (Measurements)	Execution time = 3 ms (Measurements)	Previous generation, have not been fully wrapped as ProCom component
Combiner	48 kb (Static analysis)	Execution time = 0.5 ms (Static analysis)	Fully implemented
Brake Valves	48 kb (Measurements)	Execution time = 1 ms (Measurements)	Wrapped as ProCom

Component Name	Dynamic memory consumption (Source)	Execution or response time (Source)	Remarks
ESC	Front Node = 385 kb Rear Node = 362 kb (Composition of measurements and static analysis)	Response time = 12.5 ms (critical path is 8.5 ms, plus 2 ms twice for network lag)	(Inherits every weakness of ingoing composed values.)

Milestone MS3 (feature complete)

Milestone *MS3 (feature complete)* is performed in the same manner. First, it is ensured that the implementation is on according to schedule, i.e. is actually complete with respect to features. With the help of some tools, an allocation is (hopefully) found which allocates the software components to the hardware nodes so that all the milestone verification criteria of Table 2 are met, as guaranteed by the method specified as *required source* for MS3, and of course comparing with the attribute *required values* of MS3. If no such allocation is found, the same or similar solutions as for MS2 need to be discussed.

Milestone MS4 (delivery)

Similarly, milestone *MS4 (feature complete)* is performed by using the methods specified for MS4 in Table 2, comparing with the attribute *required values* of MS4 (which are the same as the product requirements). Hopefully, problems should have been identified and resolved earlier in the project, so that the milestone verification passes satisfactory (otherwise, the project will be delayed, and some decision involving one or more of a combination of more powerful hardware, removal of features, optimization, or other ways to make all requirements be met, need to be made).

PART II: CMMI Process Areas

Causal Analysis and Resolution (CAR)

Nothing added.

Configuration Management (CM)

Note. The approach requires a mature use of configuration management to keep track of correct versions of the various artifacts.

Note. We expect that for some milestones the need to create a formal baseline is not very strong. For example, in the early phase where budgets and estimates are predominant, the assigned values are not tightly connected to specific versions of the components. In later stages, when e.g. static analysis and extensive testing is used to provide guarantees, there is a strong need to create component and product baselines.

Decision Analysis and Resolution (DAR)

Nothing added.

Integrated Project Management +IPPD (IPM+IPPD)

Nothing added.

Measurement and Analysis (MA)

Nothing added.

Organizational Innovation and Deployment (OID)

Nothing added.

Organizational Process Definition +IPPD (OPD+IPPD)

Nothing added.

Organizational Process Focus (OPF)

Nothing added.

Organizational Process Performance (OPP)

Nothing added.

Organizational Training (OT)

Nothing added.

Product Integration (PI)

Note. It can be expected that any component-based approach, with proper tool support, makes integration a less effort-consuming task [27]. The user will be aware of incompatible components already during design; indeed with a proper tool it is not even possible to connect incompatible interfaces. Similarly, missing connections are detected interactively during design and implementation. In short, some integration activities are covered in the *Technical Solution* process area, and the *Product Integration* process area will require less effort. However, there will still be many qualified integration tasks to do, which however may be performed earlier in the development process. For example, the actual deployment to hardware is not addressed in the current version of PPG, nor is the integration of newly developed components (which by construction follow the rules of the component model) with legacy systems. Also, the process needs to consider the relationship to software platforms such as operating systems.

Reference. See process area Technical Solution (TS) concerning how PI is addressed during design and implementation.

Project Monitoring and Control (PMC)

Amplification (Software Engineering). Monitor whether *milestone verifications* have been conducted as specified, and whether the results are satisfactory. Otherwise, some corrective action must be taken, which may e.g. involve renegotiating requirements, schedule, or product features.

Reference. For project (re-)planning, see process area PP.

Reference. For requirements changes, see process areas RD, REQM.

Reference. To manage the risks of not meeting the set goals of a *milestone verification*, see process area RSKM.

Reference. See more details on *milestone verification* under process area VER, in particular SG 3.

Reference. Process area PPQA SP 2.1 describes that noncompliance issues need to be communicated to the proper level of management for resolution.

Project Planning (PP)

Reference. The definition of milestones should be done in cooperation with RD, and TS, and VER.

SG 1 Establish Estimates

SP 1.1Estimate the Scope of the Project

Nothing added.

SP 1.2Establish Estimates of Work Product and Task Attributes

Nothing added.

SP 1.3Define Project Lifecycle

Typical Work Products. Definition of major and minor milestones for the project. (In terms of the Conceptual Product Meta-Model, this corresponds to instantiating a number of *Milestone Verification*s and specifying *required source*.)

Reference. The definition of milestones should be done in cooperation with RD, TS, and VER.

SP 1.4Determine Estimates of Effort and Cost

Nothing added.

SG 2 Develop a Project Plan

Nothing added.

SG 3 Obtain Commitment to the Plan

Nothing added.

Process and Product Quality Assurance (PPQA)

Note. The whole verification-intensive approach is intended to increase product quality, as well as provide the visibility necessary for efficient quality assurance.

SG 1 Objectively Evaluate Processes and Work Products

SP 1.1Objectively Evaluate Processes

Nothing added.

SP 1.2Objectively Evaluate Work Products and Services

Typical Work Products. *Milestone Verification* results and reports. The *Milestone Verifications* is the primary means to regularly collect the progress on work products.

SG 2 Provide Objective Insight

Nothing added.

Quantitative Project Management (QPM)

Note. The data collected during verification activities help to quantitatively answering questions about the project status.

Requirements Development (RD)

Note. The PPG concerns product requirements, i.e. requirements expressed in technical terminology which can be measured and verified. The PPG does not concern customer requirements, i.e. requirements expressing customer goals, and which typically relates to product validation.

Typical Work Products. In terms of the Conceptual Product Meta-Model, attributes are specified the attributes on systems (and possibly components) that have requirements. Specify a *Required Value* for the attribute. "Value" should here be interpreted in a broad sense; as described in section "Conceptual Product Meta-Model" (p. 11ff) values include e.g. intervals, distributions, and more complex models such as state machine models.

Reference. The definition of milestones should be done in cooperation with PP, TS, and VER.

Note. Not explicitly included in the PPG are additional conditions required for the attribute values to be meaningful. For example, to claim that a specific early milestone has been reached, the design models must be "detailed enough", meaning that the component structure has to be "fine-grained enough".

Reference. The definition of milestones should be done in cooperation with PP, VER, and TS.

Note. First and foremost, product requirements will be specified for the "root component", i.e. the "system", and its immediate children. It may also be specified for constituent components, provided that it these requirements are related to some customer requirements. From a supplier chain perspective, one may want to specify components to be developed by subcontractors as strict "requirements" even if they from a product perspective are merely "budgets".

Requirements Management (REQM)

Nothing added.

Risk Management (RSKM)

Note. *Exploratory analysis* and *milestone verification* are important methods of reducing risks in the project.

Reference. *Exploratory analysis* is described in TS SP 3.1, and *milestone verification* in the VER SP 3.1.

Supplier Agreement Management (SAM)

Nothing added.

Technical Solution (TS)

Note. On the distinction between “design” and “implementation”: for some recent approaches, including model-driven development [15][16] and the component-based approach as described in the introduction, the distinction between design and implementation is blurred. For example, in the component model ProCom, from the moment a component has been created in a design tool, it can be used to generate its runtime representation. However, in ProCom the primitive components are implemented with source code, and there is thus a distinction between designing such a component's interface and providing its implementation. On the other hand again, if a system is built solely from existing primitive (and composite) components, all design work can effectively also be seen as implementation work. Also in these cases, we believe that at a conceptual level a distinction can still be made between design tasks, in the sense “important decisions that will influence much of the downstream work”, and implementation.

Reference. The definition of milestones should be done in cooperation with PP, RD, and VER.

SG 1 Select Product Component Solutions

Nothing added.

SG 2 Develop the Design

Note. ProCom and related Progress technologies strongly support this goal, since interface elements are first-class entities and there is strong tool support to design interfaces and detect interface mismatches.

SP 2.1Design the Product or Product Component

Nothing added.

SP 2.2Establish a Technical Data Package

Nothing added.

SP 2.3Design Interfaces Using Criteria

Typical Work Products. Specification of signals in the system. (In addition to the *component* interfaces connected with *connectors*, the *connectors* themselves may be seen as signals in the system which should be first-class entities and managed explicitly.)

SP 2.4Perform Make, Buy, or Reuse Analyses

Note. ProCom and related Progress technologies strongly support this practice, in the case where there are potential ProCom components to reuse: it is possible to explore “what-if” scenarios by plug the components into the architecture and analyze the system properties.

SG 3 Implement the Product Design

SP 3.1Implement the Design

Amplification. Implementation and design consists of (at least) the following major tasks:

- Component internal design and implementation, which may be either
    - Subdivision into components (composite)
    - COTS or some other package from a third party (primitive or composite)
    - Imported legacy code (primitive)
    - A small, relatively simple implementation of source code (i.e. primitive components; this is how the primitive components in ProCom are implemented)
- Defining explicit connectors
- Connecting components with connectors
- Development of hardware structure (in several levels, “virtual” and “physical” [26])
- Allocation of software components to hardware nodes (in two steps: to “virtual nodes” which are then allocated to physical nodes)

Note. To manage the inclusion of legacy components, methods are required that are capable of safely encapsulating system parts into legacy subsystem compartments which have some analyzable and predictable properties, but not all properties that components constructed completely in the component language has. For example, there may be mechanisms to suspend the wrapped legacy code if it executes for too long, which gives a strict upper bound on execution time, which is necessary to not interfere with other components on the same node; however this suspension makes it impossible to guarantee that the component will always produce an output, which may affect the system throughput and response time.

Reference. For COTS and third party packages, as well as importing legacy code, see also TS SP 2.4.

Typical Work Products. Component *implementations*.

Amplification (Software Engineering). Perform *exploratory analysis* at any time, by comparing any value for attributes with their required values, using any information in the conceptual model available at the time. See Figure 6. (The assignment of *values* to *component attributes* could be of several kinds, for example explicit and manual, such as when an architect enters an expert estimate, or explicit and automatic, such as when someone executes an analysis tool on a component, or possibly implicit and automatic, such as when a composite component is analyzed with a tool which traverses each subcomponent and assigns attribute values before composing these values for the containing component.)

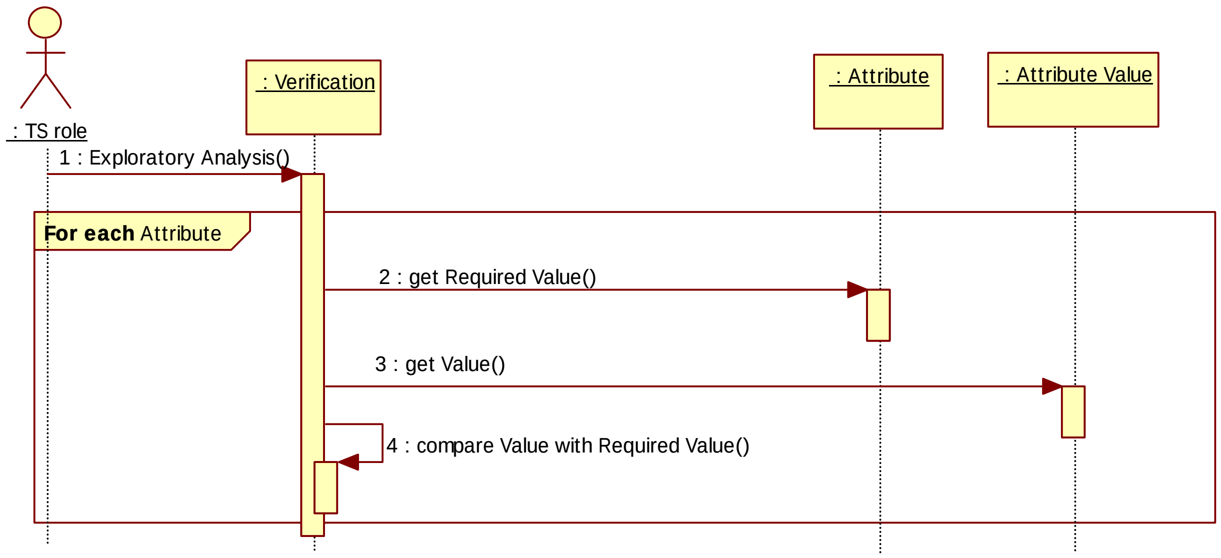

Figure 6: Exploratory Analysis[11].

[11] Note on the figure: To keep the sequence diagrams simple, only the main flow is shown. For example, if some test fails as much information as possible should be shown to the user, or, at least, made available, and it may or may not make sense to continue traversing other attributes. Also, to keep the sequence diagrams simple, it is assumed that the correct attribute value is used, out of the potentially many with different *sources* and *versions*.

Note. In exploratory analysis, the current values of attributes, which may be any combination of estimates, values produced by tools based on incomplete implementations, etc. may be used exploratory to answer questions like "how near the limit for memory consumption are we right now?" There are no absolute guidelines how to perform this type of analysis; in some cases it is reasonable to use the average of several values, or use the most pessimistic of all values for an attribute, or add 20% to measured values to add a safe margin, or because some features are not implemented yet, or decrease 20% from some measured values because it is believed that low-level optimization at the end will achieve this. (See the example on page 15ff.)

Reference. Exploratory analysis can be seen as more relaxed version of *milestone verification* (see process area Verification (VER)). (All the decisions described above as informal for exploratory analysis are formalized in milestone verification.)

SP 3.2Develop Product Support Documentation

Nothing added.

Validation (VAL)

Nothing added.

Verification (VER)

According to the CARMA principle (see page 4), verification is performed not only at the end, but regularly, as specified by milestones, utilizing the component-based approach, supported by proper analysis tools.

SG 1 Prepare for Verification

Amplification. To follow up the product requirements, define milestones, consisting of (at least):

- Date of the milestone.
- Specify, for each *attribute* to be verified at that milestone:

  - *Required value*.
  - *Required source*. Examples of such sources are: expert estimates, static analysis, measurements, model checking.
- General criteria which are assumptions for the required attribute values to be valid indications of progress. Examples include, for an early milestone, that the architecture is mature enough, e.g. in terms of granularity of components, or, for a later milestone, that implementation has progressed according to plan in terms of e.g. feature completion.

Note. As described in the section "Conceptual Product Meta-Model" on page 11ff, a "required value" may be a predicate including a somewhat complex criterion, such as "less than *X*", "within a the range of", or "equivalent with" or "a subset of" a certain behavioral model, etc.

Note. As the example on page 15fff illustrates, these values are typically specified in some relation to the product requirement, such as 70% of the product requirement.

Reference. The definition of milestones should be done in cooperation with PP, RD, and TS.

SP 1.1Select Work Products for Verification

Amplification (Software Engineering). When all components are implemented in ProCom, verification should be as automated and efficient so that in principle all components should be subject to extensive verification of the properties of interest.

SP 1.2Establish the Verification Environment

Note. When using ProCom, there should be a backbone of Progress methods and technologies which are applicable for verification of many properties that are of interest for many components. However, it should be expected that there are both third-party tools compatible with, or integrated in, the Progress tool suite, as well as separate verification tools. All tools to be used have to be specified, as well as their interaction, and procedures for how and when to use them.

SP 1.3Establish Verification Procedures and Criteria

Typical Work Products. Specification of *required value* and *required source* for each attribute, for each milestone.

SG 2 Perform Peer Reviews

Nothing added.

SG 3 Verify Selected Work Products

SP 3.1Perform Verification

Amplification. Perform *milestone verification* as planned. See Figure 7.

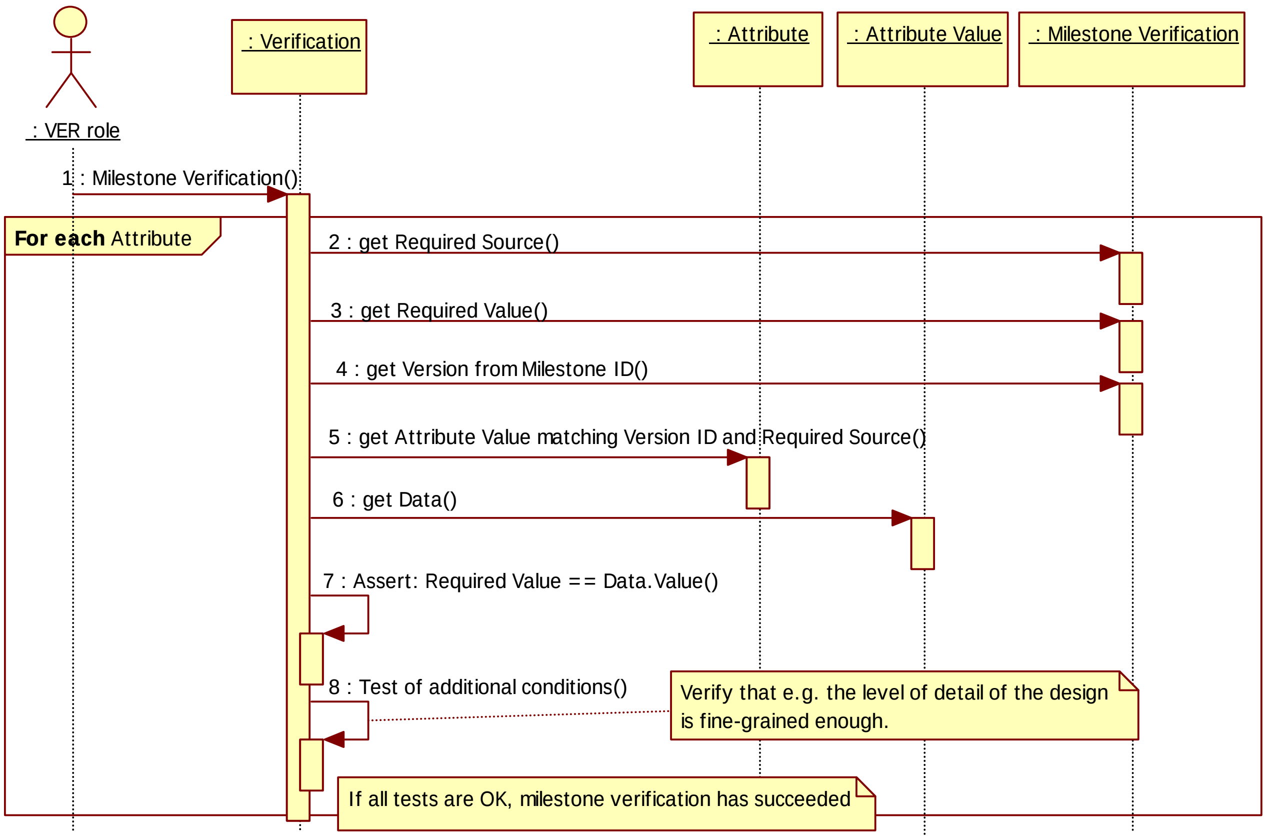
: VER role
: Verification
: Attribute
: Attribute Value
: Milestone Verification
1 : Milestone Verification()
For each Attribute
2 : get Required Source()
3 : get Required Value()
4 : get Version from Milestone ID()
5 : get Attribute Value matching Version ID and Required Source()
6 : get Data()
7 : Assert: Required Value == Data.Value()
8 : Test of additional conditions()
Verify that e.g. the level of detail of the design is fine-grained enough.
If all tests are OK, milestone verification has succeeded

Figure 7: Milestone Verification. (See footnote for Figure 6.)

SP 3.2Analyze Verification Results

Reference. See process areas PMC for how to use the verification results.

Conclusions

This report describes the CARMA (Components, Attributes, Requirements, Milestones, Analysis) principle for development of software-intensive systems according to the Model-Driven Development (MDD) and Component-Based Software Engineering (CBSE) paradigms. It does so by detailing out the requirements on a development process in the form of a CMMI extension, thus providing guidelines in the form of a reference text. The guidelines are a product of the PROGRESS Centre for Predictable Embedded Software Systems, and the main goal of these guidelines is to describe how to utilize the characteristics of a strong component model, accompanied with proper analysis methods and tools, to design a process aimed at risk reduction and project monitoring and control.

The guidelines are based on several types of background research activities, ensuring relevance of these guidelines. As the development of these guidelines is done in parallel with development of suitable methods and technologies assumed to be used, the guidelines are not yet validated.

An example scenario (page 15ff) illustrates that the CARMA principle, accompanied with proper methods and tools for allocation and analysis, is potentially a powerful principle to monitor the project and highlight potential risks which can then be handled explicitly. It also shows that the assignment of target values for milestones suffers from lack of complete knowledge of actual events and decisions during the project, and that they therefore may be re-negotiated or ignored during the project, after some careful consideration.

At this stage, they should be seen as a vision towards which industrial processes can be adopted, as suitable, when the required languages, methods, and tools, are maturing and adopted. For example, it is not known where the optimal tradeoff is between the effort required to formulate proper milestones and perform the actual milestone verifications, and the risk reduction gained.

Further work include simulating a development process model which embeds the characteristics set forward in this report, and providing tool support, as an extension to the PROGRESS suite of development tools, and thus explore the benefit (if any) by adopting the CARMA principle with simulations and pilot studies. Hopefully, as tools are maturing, we hope to study industries which adopt the MDD/CBSE approach, and evaluate how the PPG can be adopted and what benefit it brings in practice.

Acknowledgements

This work was partially supported by the Swedish Foundation for Strategic Research (SSF) via the strategic research centre PROGRESS. The authors would like to thank Jörgen Hansson at the SEI for his mentioning “Virtual Verification” in a talk, which brought some structure to our early ideas. See “Related Work” section on page 8. We also would like to thank Branka Pavetić for collaboration which has included some parts of the interview study mentioned in section “Research Method” on page 8. We also would like to thank all the researchers at the Progress centre which have participated in the interviews as well as informal and fruitful discussions.

References

[1] Bran Selic, "The Pragmatics of Model-Driven Development," *IEEE Software*, vol. 20, no. 5, 2003.

[2] Clemens Szyperski, Dominik Gruntz, and Stephan Murer, *Component Software – Beyond Object-Oriented Programming*, Second Edition ed.: Addison-Wesley , 2002.

[3] Kurt C. Wallnau, "Volume III: A Technology for Predictable Assembly from Certifiable Components," Software Engineering Institute, Technical Report CMU/SEI-2003-TR-009, 2003.

[4] Nico Feiertag, Arne Hamann, Daniel Karlsson, Jörg Kemmerich, Stefan Kuntz, Henrik Lönn, Elke Löschner, Jennifer Neumüller, Nadym Salem, and Martin Schlager, "TIMMO Timing Model : Methodology Version 2," TIMMO, Deliverable D7 2009.

[5] Peter H. Feiler and Jörgen Hansson, "Toward Model-Based Embedded System Validation through Virtual Integration," *DoD Software Tech News*, vol. 12, no. 4, January 2010.

[6] Peter H. Feiler, Jörgen Hansson, D. de Niz, and L. Wrage, "System Architecture Virtual Integration: An Industrial Case Study," Software Engineering Institute, CMU/SEI-2009-TR-017, 2009.

[7] Rikard Land, Jan Carlson, Stig Larsson, and Ivica Crnkovic, "Project Monitoring and Control In Model-Driven and Component-Based Development of Embedded Systems : The CARMA Principle and Preliminary Results," in *5th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, Athens, Greece, 2010.

[8] Rikard Land, Jan Carlson, Stig Larsson, and Ivica Crnković, "Towards Guidelines for a Development Process for Component-Based Embedded Systems," in *Workshop on Software Engineering Processes and Applications (SEPA) in conjunction with the International Conference on Computational Science and Applications (ICCSA)*, Yongin, Korea, 2009.

[9] Mary Beth Chrissis, Mike Konrad, and Sandy Shrum, *CMMI Second Edition : Guidelines for Process Integration and Product Improvement*. Boston: Addison Wesley, 2007.

[10] Séverine Sentilles, Aneta Vulgarakis, Tomáš Bureš, Jan Carlson, and Ivica Crnković, "A Component Model for Control-Intensive Distributed Embedded Systems," in *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, Berlin, 2008, pp. 310-317.

[11] Tomáš Bureš, Jan Carlson, Ivica Crnković, Séverine Sentilles, and Aneta Vulgarakis, "ProCom - the Progress Component Model Reference Manual, version 1.0," Västerås, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-230/2008-1-SE, 2008.

[12] Séverine Sentilles, Petr Štěpán, Jan Carlson, and Ivica Crnković, "Integration of Extra-Functional Properties in Component Models," in *12th International Symposium on Component Based Software Engineering (CBSE 2009)*, vol. Springer, 2009.

[13] As-2 Embedded Computing Systems Committee, "Architecture Analysis & Design Language (AADL)," Standard Document Number AS5506, 2009.

[14] Scott A Hissam, Gabriel A Moreno, Judith Stafford, and Kurt Wallnau, "Packaging Predictable Assembly with Prediction-Enabled Component Technology," Pittsburgh, CMU/SEI-2001-TR-024, 2001.

[15] Thomas Stahl and Markus Völter, *Model-Driven Software Development : Technology, Engineering, Management.*: John Wiley & Sons, 2006.

[16] Anneke Kleppe, Jos Warmer, and Wim Bast, *MDA Explained : The Model Driven Architecture: Practice and Promise*. Boston: Pearson Education, 2003.

[17] ATESST Consortium, "EAST ADL 2.0 Specification," Draft Report 2008.

[18] Philippe Kruchten, *The Rational Unified Process : An Introduction*, 3rd ed. Upper Saddle River: Addison-Wesley, 2004.

[19] Kent Beck, *EXtreme Programming EXplained: Embrace Change.*: Addison Wesley, 1999.

[20] Paul Duvall, Steve Matyas, and Andrew Glover, *Continuous Integration: Improving Software Quality and Reducing Risk.*: Addison-Wesley Professional, 2007.

[21] Ivica Crnković, Michel Chaudron, and Stig Larsson, "Component-based Development Process and Component Lifecycle," in *International Conference on Software Engineering Advances (ICSEA'06)*, Tahiti, 2006.

[22] Holger Krahn, Bernhard Rumpe, and Steven Völkel, "Roles in Software Development using Domain Specific Modelling Languages," in *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM' 06)*, Portland, Oregon, 2006.

[23] Jan Øyvind Aagedal and Ida Solheim, "New Roles in Model-Driven Development," in *Proceedings of Second European Workshop on Model Driven Architecture (MDA)*, Canterbury, England, 2004.

[24] Defence Materiel Organisation, Australian Department of Defence, "SAFE, V1.2 : A Safety Extension to CMMI-DEV, V1.2," SEI technical note CMU/SEI-2007-TN-006, 2007.

[25] Fergal McCaffery, John Burton, and Ita Richardson, "Improving software Risk Management in a Medical Device Company," in *ICSE Companion*, 2009.

[26] Jan Carlson, Juraj Feljan, Jukka Mäki-Turja, and Mikael Sjödin, "Deployment Modelling and Synthesis in a Component Model for Distributed Embedded Systems," in *36th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Lille, France, 2010.

[27] Stig Larsson, "Key Elements of Software Product Integration Processes," Västerås, PhD thesis, 2007.