Using Prediction Enabled Technologies for Embedded Product Line Architectures

Magnus Larsson
ABB Automation
Technology Products
mlo@mdh.se

Anders Wall
Mälardalen University
Department of
Computer Engineering
awl@mdh.se

Christer Norström
Mälardalen University
Department of
Computer Engineering
cen@mdh.se

Ivica Crnkovic
Mälardalen University
Department of
Computer Engineering
icc@mdh.se

Abstract

Predicting the behavior of a product before it is built has been a long time struggle, especially for software based systems. For building software systems there are few methods that comply with the engineering methods established from physics where properties of construction can be determined before the actual assembly of a product. By taking the predictable assembly from certifiable components (PACC) approach our intention is define methods to predict certain properties. We conclude that product line architectures that build on top of a component technology can be build in a much more controlled way if the component technology is prediction enabled. The aim of this position paper is to investigate how embedded product line architectures can utilize a prediction enabled component technology to build products with known properties. We present a framework where we can reason about extra-functional properties in a uniformed way. We illustrate our approach by an example including several different extra-functional properties.

Keywords:
Real-time systems, Component-based development, Product-line architectures.

1 INTRODUCTION

Applying the concept of product-line architectures (PLAs) is one way to achieve component reuse and benefits from component-based development. A PLA from a software system's perspective is a common architecture, a set of common strategies, tools, and methods that are shared among several different products within a particular domain [1-4]. Thus, not only components are reused, but also the architecture and the design strategies that initially were chosen. Examples of such strategies are strategies for adding new features to an existing PLA and strategies for providing variability. A product line contains many products that in turn may have many different features. Typically, features realize a set of functional, and extra-functional requirement (e.g. quality of services, temporal constraints, etc.). Variations in features may be obtained in different ways; by applying variations in flexible software architecture, parameterization of existing components, or by using different implementations of components. In a PLA it is more likely that the software architecture is a constant, while flexibility is achieved through component variations. New functional, or extra-functional features will be implemented by adding new components or by using different variants of components. From the predictability point of view, obtaining new functional features of the products is straightforward as they come directly from the functional properties of components. On the opposite, the extra-functional properties of the products are almost unpredictable; for example components with new functional features can degrade quality of services. Also, a PL strategy can be focused on product families with the same functional properties, but different extra-functional properties, e.g., scalability, flexibility, and safety. For this reason, the ability to derive extra-functional properties from the properties of the components plays a significant role for PLA. Even more, as PLA identifies the variable parts and core (repeatable) parts, the findings from existing product versions can be taken as input to methods of the predictability technologies which results may be more accurate and provided in a simpler in way.

In this position paper we present our current project that will design a *prediction-enabled component technology* (PECT) for product line architectures in the real-time systems domain. The goal of this work is to provide a framework in which extra functional properties can be added via *analytical interfaces* to a component such that interesting properties of the composed systems can be expressed and analyzed in relation to properties of built in components. In this paper we shell demonstrate analytical interfaces by several examples that apply to the properties we analyze. However, the intention of this work is to provide a framework in which analytical interfaces can be added to the model such that any interesting property of an assembly can be expressed and analyzed. The complete set of defined analytical interfaces constitutes a component's *analytical model*.

We illustrate our approach by presenting a component model for embedded real-time system and using that model for illustrating how three completely different non-functional properties can be analyzed by using the same framework.

The remainder of the paper is the following. Section 2 gives a short overview of PLA model used for predictability technology. Section 3 introduces the definitions and terms based on the approach defined in [5-7]. Section 4 elaborates on the different properties of assemblies and the paper is concluded in section 5.

2 PRODUCT LINE ARCHITECTURE AND COMPONENT-BASED DEVELOPMENT

In this paper we assume a product line architecture to be based on a component technology. There are however different approaches of how products are assembled from components for a chosen set of features. Figure 1, shows, in a simplified manner, how concepts and vocabulary used in this paper relate. A product line consists of different products that are distinguished by different features but they also share a set of common features. In a component based context features are implemented by components whose attributes are specified by component credentials. Examples of such attributes are different temporal attributes such as frequencies. Another examples is component version dependency on other components. Credentials are introduced by Shaw [8] and represent a property, value and credibility. In this paper we do not utilize the complete concept of credentials as we leave out the statistical confidence for how certain properties were obtained.

The flexibility in the PLA is accomplished through variation points that define the strategies for varying the systems behavior between products. Examples of variation points in a component based software systems are variations of components (either by adding new components, providing new component versions, or by parameterization of components' interfaces).

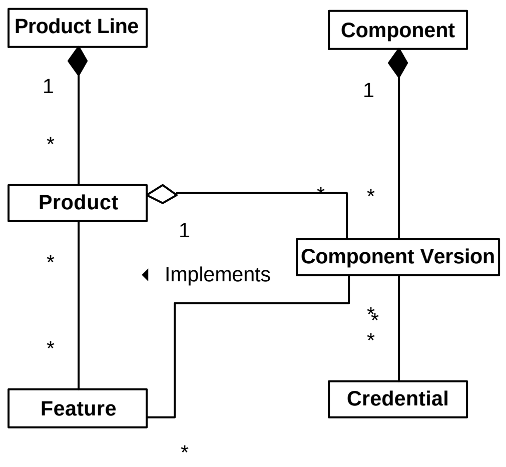

Figure 1. A UML model of the software contents of a product

The flexibility is not only achieved by functional properties but also by extra-functional properties. For instance, in the real-time systems domain we are interested in the temporal behavior of a system as it is considered correct only if it performs correct function at correct time, i.e. *temporal correctness*. Consequently, by adding the temporal domain we must not only manage functional flexibility but also temporal flexibility. For instance, the frequency with which a particular component executes may vary between a high-end product and a low-end product due different demands from the controlled process.

One of the main problems in constructing and maintaining a PLA is to express and verify product properties derived from the component properties. To be able to predict the product properties form the component properties, we define a *prediction-enabled component technology* (PECT) similar to the one proposed in [6]. In a PECT *constructive interface* is separated from *analytic interfaces*. While a constructive interface deals with operational (functional) properties, analytical interface describes extra-functional properties. An analytical property is very much the same as a credential by means of having extra information about a component.

3 COMPONENTS AND ASSEMBLIES

To be able to analyze properties of component-based products, we must first be able to specify the properties of the components and identify the communication between them. Different component models specify this to different extent. Most of them do not treat extra-functional properties. Our component model is based on the port-based object approach in which components are connected to each other by data ports that constitutes a components *data interface* [9]. This component model extends the expressiveness of port-based objects and is presented in a simplified manner hereinafter. For a more detailed description we refer to [10].

In Figure 2, our component meta-model is depicted in UML-fashion. Components have in and out ports which resembles the data interface. Also, a component encapsulates services, which provide the actual functional behavior. Besides having data interfaces, defined by their ports, components in the framework have two additional interfaces, *control interface*, and *parameterization interface*. The execution of, and synchronization among components is controlled through its control interface by associating a task to the interface. A task provides a thread of execution that is defined and restricted by a set of attributes, e.g. priority, frequency. A task in our framework can be based on any task model defined in by the used real-time operating system (RTOS). A task is a runtime mechanism and hence, it is a constructive part of a component. However, note that some of the attributes of a task are required when, together with some analytical properties, analyzing temporal properties of an assembly. The parameterization interface defines the points of variation of a component's behavior.

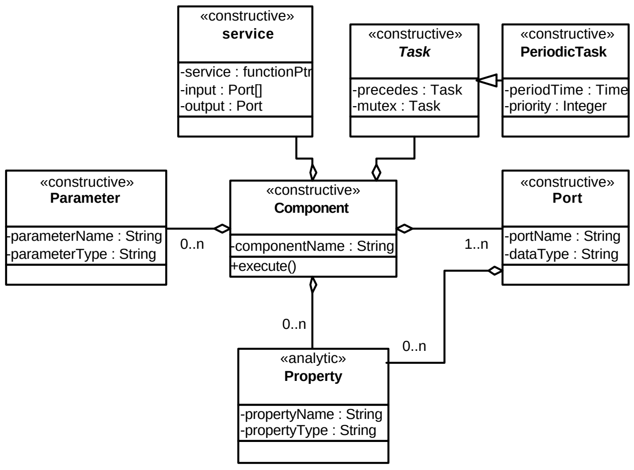

Figure 2. The component model

The property class that is stereotyped as analytic provides the information needed by the different analyses we are interested in performing on an assembly. An analytical component property usually does not have a correspondence in a component instance. A typical example of such a property would be the execution time of a service of a component. The execution time is derived from the source code, or by measurements, for the purpose of modeling and analysis of a system and has no correspondence as such in the runtime.

For further discussions we need definitions of certain terms in our component model. In this model we shall emphasize the real-time properties. Formally we define the constructive part of the component model depicted in Figure 2 as:

Definition 1 A component c is a tuple $\langle f, P, I, O, C, s_c\rangle$, where f is the service encapsulated by c, P is the set of parameters, I is the set of in-ports, O is the set of out-ports, C is the control interface and s_c is the state of component c. ÿ

A component's state is updated by the service within a component and remains in between consecutive executions of a component.

An assembly is a specific configuration of a set of components that also defines the components interconnections. The union of all its component's states gives the state of an assembly. Formally we define an assembly as:

Definition 2 An *assembly A* is a tuple $\langle C(A), R^{*}\rangle$, where $C(A) \subseteq C$ is the set of components in A, and R^{*} is the set of relations valid between $C(A)$ in A, and C is a set of all components encapsulated in the product ÿ

Note that an assembly does not necessary corresponds to a product. While in some cases we are interested in properties of the product, in some cases we may want to analyze properties of a sub-part of the complete product. I both cases we will refer to an assembly. An assembly is only a conceptual- and analytical view of a complete product that exists for the analysis of a particular property, and has not necessarily a constructive correspondence.

In order to construct an assembly, we must be able to connect components with each other via some relation. In our definition of an assembly we have three kinds or relations among components that belongs to the set R, *precedence*, *mutual exclusion* (mutex), and *data-flow connections*.

Precedence and mutual exclusion specify the synchronization among tasks that controls the execution of components. Formally we define precedence and mutual exclusion as:

Definition 3 A precedence relation, $\rightarrow$, is a binary, transitive relation among a pair of tasks $\langle t_{i,}\ t_j\rangle \in T{\times}T$, such that if $t_i \rightarrow t_j$, then t_j may start its execution earliest at the end of t_i's execution and $i \neq j$. ÿ

Definition 4 A mutual exclusion relation, $\otimes$, is a binary, symmetric relation among pair of tasks $\langle t_{i,}\ t_j\rangle \in T{\times}T$, such that if $t_i \otimes t_j$, then neither t_i nor t_j is permitted to execute while the corresponding party, or a transitively related party is executing and $i \neq j$. ÿ

Besides synchronization, we can also specify data-flow relations among components in an assembly. Data-flow connections specify the data that are exchanged between components in an assembly through their ports. We define the data-flow relation as:

Definition 5 A data flow connection =, is a binary, anti-symmetric relation among pair of ports on components, $\langle c_i.i_x\ c_j.o_y\rangle \in C.I{\times}C.O$, such that if $c_i.i_x = c_j.o_y$ then c_i's in port i_x is connected to c_j's out port o_x. ÿ

In next section we will describe how to predict properties of an assembled real-time system product from a PLA perspective.

4 PROPERTIES OF AN ASSEMBLY IN A PRODUCT LINE PERSPECTIVE

The intention of our work is to provide a framework in which new properties of an assembly could be taken into consideration and predicted. The general idea is that if the model has to be extended with a new predictable property, new analytic properties can be defined and new property theories be developed. For instance, if we require an assembly to be type correct, i.e. the types of connected data ports are correct, we must add a method for checking this property and doing so require an analytical property on data ports which carries the type information. Furthermore, we are using the prediction technologies in a product line perspective, i.e. we will discuss properties that are important when developing and maintaining product line architectures. There is usually a component technology associated with a product line. Components in such a PLA conform to a particular component model. We strive to make the supporting component technology prediction enabled and hence simplify the way products are assembled from components residing in a defined repository. By having a PECT as a base for product line architectures we can predict certain properties of the products before deploying them.

There are several realistic scenarios describing activities that a product line may undergo during its lifetime. We have not identified all possible scenarios but highlighting some relevant cases and propose examples of properties that are interesting from their perspective.

Scenario 1: New features will eventually be added to a product line or a specific product within the product line. This new feature might be implemented by a set of new components as well as new versions of old components already existing as part of the reusable assets in the product line. Doing this, there is a potential risk that components could end up being incompatible with components already used in the product, both with respect to version and variants. This scenario is also related to maintenance of a product that may alter the characteristics of a particular component. This change of characteristics is possibly acceptable for one particular product, but what are the consequences in the rest of the product line?

Scenario 2: As we operate in the real-time systems domain, we are also interested in predicting the temporal behavior of an assembly. Adding component to-, or changing components in a product or product line, may violate the temporal constraints in the system. The reason for violating the temporal constraints could be an over-utilization of the available resources in the system architecture. A big share of existing real-time systems are embedded systems, thus resources are usually restricted.

The scenarios discussed above also apply to the assembly of a new product, based on pre-existing reusable components. We have to make sure that the product is feasible both with respect to the functional behavior and the temporal behavior.

We will refer to the analysis of relevant properties of assemblies in a product line prospective as *impact analysis*. Thus, we want to analyze the impact of an change, e.g. installing new features in a product, maintaining existing components, construct a completely new product based on reusable assets within the product line.

2.1 Assembly properties

To illustrate predictability of assemblies for the specified component model, we shell discuss two concrete examples of assembly's properties from a real-time product line's point of view: *consistent*, and *end-to-end deadlines*.

The consistent property, *A.consistent*, is related to a capability to predict consistency of an assembly. An assembly is considered consistent if the versions of each component are correct according to the specification of a product in the product line. The specified features of a product determine which components, and in particular which components version should be included in a product. To be able to guarantee consistency we need to specify what versions of components a product depends on. Thus, two analytic properties need to be added to the analytic model, a *version* identifier and a *depends_on*. The depends_on property includes a list of version pairs stating what components and what versions are needed to make it work.

This idea of having version dependencies is very similar to how .NET assemblies use meta-data to describe dependencies to other assemblies [11]. Dependencies can be expressed and assured using OCL constraints for the components. A new constraint has been added to all components that state how the dependencies shall be evaluated and regarded analyzing the assembly.

The second example of properties is related to temporal constraints. The temporal correctness is of vital importance in the real-time systems domain. Moreover, the temporal requirements on a real-time system are seldom presented in terms of the temporal attributes provided by the RTOS or as simple deadlines for individual components. Typically they are considered on a higher level; for instance jitter constraints for the control performance, end-to-end deadlines, response times, etc. Designing a real-time system is partly a matter of transforming such high-level temporal requirements to the attributes available in the task model at run-time, typically considering priorities and frequencies. In our approach the high-level temporal requirements are specified as properties on an assembly, e.g. end-to-end deadline, and the implementation of those requirements, e.g. frequencies, priorities, execution times, are specified as analytical properties on components.

A concrete example of a temporal property is *end-to-end deadline*. An end-to-end deadline, *A.e2e*, specifies a temporal requirement on a set of components. It defines the maximum distance between start of the execution of the first component and the completion of the last component. Typically, the end-to-end property requirements in hard real-time systems must be meet, while in soft real-time systems a particular confidence of meeting the requirement may be sufficient. Statistical verification of a prediction theory can be performed to show how reliable the prediction actually is, e.g the confidence in the estimated worst-case execution time.

Verifying that a temporal property of the assembly is feasible, we verify that our temporal implementation is correct. However, this verification is correct under the assumption that all prerequisites are correct (For example, the execution time of a component, which is a component property). Consequently, the correctness of a property of an assembly depends on the confidence we have in analytical properties. The concept of credentials as presented in [8] includes a notion of confidence associated with a component property. The execution time can be statically analyzed given the source code, or empirically measured in runtime [12]. Empirical validation of the prediction theory is also needed to prove the soundness of the theory.

The properties introduces above are of completely different nature. *Consistent* are typically a property of a complete product. *End-to-end deadline* only concerns a subset of components in a complete product assembly. Moreover, there can be several end-to-end deadline requirements within the same assembly with respect to a subset of components from the full assembly.

2.2 The end-to-end temporal property

Figure 3 shows an example where four components have been instantiated from the model presented in Figure 2. The infrastructure in which those components will execute (the RTOS) has a scheduling policy based on fixed priorities. The task model consequently specifies the level of priority and the frequency of each task. When defining an assembly we also must specify how the assembly is build. There are not only the properties of the components that determine the properties of an assembly, but also the assembly architecture; we must define how the assembly is built. For example, in a pipe-filter architecture the dataflow between components (i.e. the precedence relations) must be specified. In this example we define the precedence property and ports connections. We also add an analytical property that specifies how many times components are supposed to be executed.

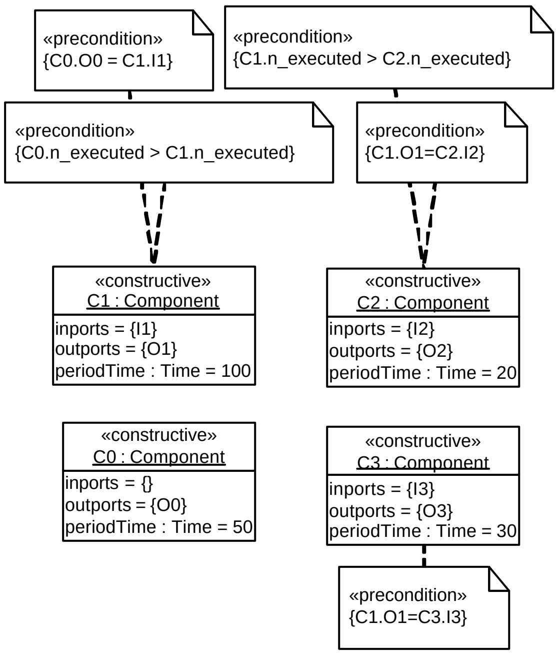
«precondition» {C0.O0 = C1.I1}
«precondition» {C1.n_executed > C2.n_executed}
«precondition» {C0.n_executed > C1.n_executed}
«precondition» {C1.O1=C2.I2}
«constructive» C1 : Component
inports = {I1}
outports = {O1}
periodTime : Time = 100
«constructive» C2 : Component
inports = {I2}
outports = {O2}
periodTime : Time = 20
«constructive» C0 : Component
inports = {}
outports = {O0}
periodTime : Time = 50
«constructive» C3 : Component
inports = {I3}
outports = {O3}
periodTime : Time = 30
«precondition» {C1.O1=C3.I3}

Figure 3. Four components with precedence and connection relations specified using constraints

Component c_1 has two preconditions, the first one express the precedence relation and the second the connection of ports.

The figure shows four components where c_1 reads the out ports of c_0 and c_2, c_3 reads the out ports of c_1. c_0 precedes c_1 and c_1 precedes c_2, while c_3 can execute independently (i.e. $c_0 \rightarrow c_1$ and $c_1 \rightarrow c_2$). Below is the components described according to definition 1:

$c_0 = \langle f, P_0, \emptyset, \{o_1\}, f(\emptyset, \{o_1\}), \tau_0, s_0\rangle$

$$c_1 = \langle g, P_1, \{i_1\}, \{o_2, o_3\}, g(\{i_1\}, \{o_2, o_3\}), \tau_1, s_1\rangle \quad (1)$$

$c_2 = \langle h, P_2, \{i_2\}, \{o_4\}, h(\{i_2\}, \{o_4\}), \tau_2, s_2\rangle$

$c_3 = \langle x, P_3, \{i_3\}, \{o_5\}, x(\{i_3\}, \{o_5\}), \tau_3, s_3\rangle$

There are many views of one assembly depending on the relations of components. In our example we have two views, one is for precedence of components and another that shows how the components are connected through ports. The assembly in our example according to definition 2 is

$$\begin{aligned} A = \langle\ & \{c_0, c_1, c_2, c_3\}, \\ & \{R_{precedence} = \{c_0 \rightarrow c_1, c_1 \rightarrow c_2\}, \\ & \{R_{Connection} = \{(o_1, i_1), (o_1, i_2), (o_2, i_3)\}\}\rangle. \end{aligned} \quad (2)$$

One view of the assembly is the one

$$A_{Precedence} = \langle \{c_0, c_1, c_2, c_3\}, R_{Precedence} \rangle. \qquad (3)$$

The other view is

$$A_{Connection} = \langle \{c_0, c_1, c_2, c_3\}, R_{Connection} \rangle. \qquad (4)$$

We shell analyzed a high-level requirement of the assembly, namely end-to-end deadline, *A.e2e*.

An end to end deadline constraint can be defined as a property on the assembly A.e2e which can be calculated as

$$A.e2e = \text{Max}(\text{ResponseTime}(c_2), \text{ResponseTime}(c_3)) - \text{StartTime}(c_0). \qquad (5)$$

An end-to end deadline is consequently constraining the maximum time interval between start of the first component in an assembly and the finish of the last component in the assembly.

Calculating the response time of components based on the attributes provided in a fixed-priority based RTOS is done with *response time analysis* [13]. However, different methods must be utilized if a different scheduling policy is provided by the RTOS, e.g. earliest-deadline-first. Thus, the definition of a particular property may vary due to mechanisms provided by the infrastructure in which the system will execute.

In our particular example we are using fixed priority scheduling in which we calculate the response time of component c_i, $R(c_i)$, as:

$$R^{n+1}(c_i) = c_i.e + B(c_i) + \sum_{\forall c_j \in hp(c_i)} \left\lceil \frac{R^n(c_i)}{c_j.T} \right\rceil c_j.e \qquad (6)$$

where B is the blocking time and $hp(e_i)$ is the set of components having tasks with higher priority than component i.

The end-to-end property is a typical example of a property that may be defined on only part of a complete product. In Figure 3 it can be seen that c_0, c_1 and c_2 are connected with the precedence relation but c_3 can execute anytime when in the ready queue. It is of importance to be able to calculate the e2e property for c_0, c_1 and c_2 only. Our proposal is that the property shall be defined for parts of the assembly with respect to a relation. In our example we can say that c_3 is independent from the other components with respect to precedence. Hence A.e2e over $\{c_0, c_1, c_2\}$ can be calculated with the response time of c_2. By having this notation it is possible to define properties that reflects parts of the assembly.

2.3 The version consistency property

We illustrate the problem of adding a new component to a product line by continuing the previous example. We introduce a new component c_4 which is dependent on the execution of c_3 and the output from c_3 and c_2. Such a component is presented in Figure 4. The component c_4 also express its version relation. The component c_4 express that it is dependent on a version of c_3 by having a set of dependencies called depends_on. The runtime can use a precondition to verify that the correct version of c_3 is in c_4's list depends_on. Verification can be performed in runtime or for prediction of consistency in the product line before an assembly is deployed.

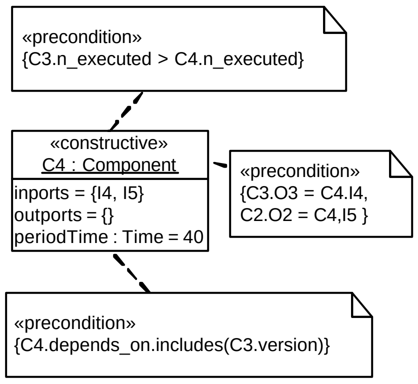

Figure 4. A new component c_4 is added to represent a new feature of a product

Before the new component is added we want to see what impact it has to the system. For instance we want to calculate A.consistent and A.e2e over $\{c_0, c_1, c_2\}$ and $\{c_3, c_4\}$.

The consistency of all versions in an assembly can be calculated with the following formula. The property consistent is of type Boolean.

$$\begin{aligned} &A.\text{consistent iff } \forall\ c.\text{consistent} \wedge c \in A \\ &c.\text{consitent iff } c.\text{depends\_on.IsEmpty} \vee \\ &\forall\ c_i \in c.\text{depends\_on} : c_i \in A \end{aligned} \qquad (7)$$

That is, the assembly is consistent if all components in the assembly are consistent. A component is consistent if it has no dependencies or if all dependants exist in the assembly. The formula holds also if the component model supports side-by-side execution of different versions of a component.

5 CONCLUSION

In this paper we have proposed the use of a prediction-enabled component technology for developing and maintaining component based product line architecture in the real-time system's domain. We have extended an existing component model with analytical interfaces that specifies the properties needed for predicting the different properties of a component assembly. As examples of properties that are interesting from a real-time product line architecture's point of view, we define the end-to-end deadline property and the type consistent property.

We have introduced the concept of impact analysis. In the impact analysis the effect of introducing new components in a product line architecture is predicted. The new components could be due to the introduction of new features in the product line or maintenance of existing components that potentially alter the characteristics of a component.

The ideas are presented in the paper as concrete examples of two properties on assemblies. However, the presented methodology is supposed to be the base to a general framework in which new assembly properties could be included as the need for them emerges. As a consequence of introducing a new assembly property, new analytical properties on the components may be needed.

As future work we will develop the property theories presented in this paper further as well as the framework concept. As the base for this work we will implement the component model and provide a tool for specifying and analyzing systems based in the component model. Such a tool should support the framework ideas. Thus, it must provide means for extending the component model with required analytical properties and to express properties on assembled products.

6 REFERENCES

[1] Bosch J., "Component Evolution in Product-Line Architectures", In *Proceedings of International Workshop on Component Based Software Engineering*, 1999.

[2] Bosch J., *Design & Use of Software Architectures*, Addison-Wesley, 2000.

[3] Clements P. and Northrop L., *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[4] Dashofy E. M. and van der Hoek A., "Representing Product Family Architectures in an Extensible Architecture Description Language", In *Proceedings of The International Workshop on Product Family Engineering (PFE-4), Bilbao, Spain*, 2001.

[5] Crnkovic, I., Schmidt, H., Stafford, J., and Wallnau, K. C., Anatomy of a Research Project in Predictable Assembly, 2002.

[6] Hissam, S. A., Moreno, G. A., Stafford, J., and Wallnau, K. C., Packaging Predictable Assembly with Prediction-Enabled Component Technology, report Technical report CMU/SEI-2001-TR-024 ESC-TR-2001-024, 2001.

[7] Wallnau K. C. and Stafford J., "Ensembles: Abstractions for A New Class of Design Problem", In *Proceedings of 27th Euromicro Conference*, 2001.

[8] Shaw M., "Truth vs Knowledge: The Difference Between What a Component Does and What We Know It Does", In *Proceedings of 8th International Workshop on Software Specification and Design*, 1996.

[9] Stewart D.B., Volpe R.A., and Khosla P.K., Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects, *IEEE Transaction on Software Engineering*, volume 23, issue 12, 1997.

[10] Wall A. and Norström C., "A Component Model for Embedded Real-Time Software product-Lines", In *Proceedings of 4th IFAC conference on Fieldbus Systems and their Applications*, 2001.

[11] Thai T. and Lam H., *.NET Framework*, O´Reilly, 2001.

[12] Lim S.S., Bae Y. H., Jang C. T., Rhee B. D., Min S. L., Park C. Y., Shin H., Park K., and Ki C. S., An Accurate Worst-Case Timing Analysis for RISK Processors, *IEEE Transaction on Software Engineering*, volume 21, issue 7, 1995.

[13] Audsley N.C., Burns A., Richardson M. F., Tindell K., and Wellings A. J., Applying New Scheduling Theory to Static Priority Preemptive Scheduling, *Software Engineering Journal*, volume 5, issue 8, 1993.