Component Based Technologies Classification

Technical report 01/2010 version 1.0

Valentina Zadrija
Faculty of Electrical Engineering and Computing
University of Zagreb
Croatia
valentina.zadrija@fer.hr

Abstract—**This paper presents a survey of a subset of the state of the art component technologies. These technologies are evaluated according to the target application domain and possibilities of transformation into sythesizeable Program State Machines (PSM). Target application domain used for evaluation are digital signal processing (DSP) applications. According to the results of a survey, goal is to select a prominent component technology, which can be efficiently transformed into PSM.**

Keywords-component; component based software engineering; UML, MPSoC

I. GOAL

The goal of this paper is to categorize prominent component technologies in order to apply them for MPSoC design. Various component technologies are categorized with an aim to determine the most applicable for transformation into Program State Machines (PSM) Model of Computation (MoC), [1]. PSM are used as a model transformation target because they can be directly synthesized into MPSoC using Embedded Systems Environment toolset [2].

The remainder of the paper is organized as follows. Basic classification guidelines are given in Section II. The classification provides an overview of two concepts; the first is the Unified modeling language described in Section IV, and the second is component SW engineering technologies. Basic terminology and common features of various component technologies are described in Section V.

Section VI gives an overview of the component based technologies organized by the priority rule of transformation into PSM. For certain modeling technologies, there is some background in MPSoC design which is discussed in the *Related work* section. After each section, we give a short conclusion in the *Takeaways* section with a description of features which could be useful for MPSoC modeling.

Cross-comparison and overall discussion of described component technologies is given in Section VII. Finally, Section VIII concludes the paper.

II. APPROACH

Basic criteria for ranking the component technologies are: (i) target application domain, (ii) ease of transformation into PSM. Typically, target domain for modeling MPSoC are digital signal processing (DSP) applications. Specific requirements for modeling DSP applications are discussed in Section II-A.

Authors in [9] emphasize that when designing the SW systems it is important to choose the right architectural style.

Definition. Architectural styles are defined as reusable 'packages' of design decisions and constraints that are applied to an architecture to induce chosen desirable qualities.

Typical examples of the architectural styles include dataflow, shared memory, publisher-subscriber and event-based styles. This means that we have to choose the component technology which can efficiently describe DSP applications and that the specification process will not be complicated in comparison to the specification by means of PSM.

A. Modeling DSP applications

A class of DSP applications includes a variety of audio and image processing algorithms. DSP applications typically include single or several computationally intensive functions like Discrete Cosine Transformation (DCT). There are several issues to be discussed when designing such applications for implementation on the MPSoC platform.

From SW design point of view, there is an issue of modeling hierarchy in DSP applications. Behavioral hierarchy is used to decompose application into sequential or concurrent sub-processes, while structural hierarchy decomposes system into a set of interconnected components. DSP applications like JPEG or H246 do not expose a high level of structural hierarchy. However, modeling applications like MP3 decoder with two channels follows the structural hierarchy. For example, each channel could be modeled as a component containing the *Alias reduction*, *IMDCT* and *DCT* components, Fig 1.

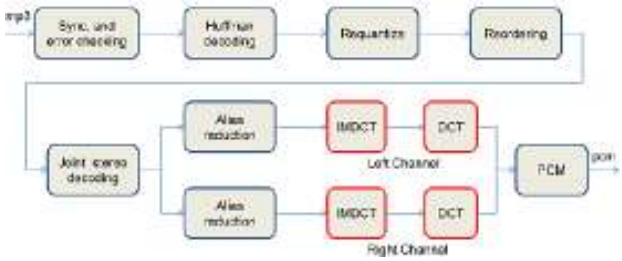

Figure 1. MP3 decoder application

From platform point (MPSoC) of view, there is an issue of parallelism of these applications. DSP application can efficiently be parallelized to exploit the data- and function-level parallelism. Function-level parallelism is achievable on MPSoC platforms, where computationally intensive functions may be offloaded onto separate processors. On the other hand, data-level parallelism corresponds to partitioning the input data over multiple instances of the same function code.

B. Program state machines

In the original form, Program state machines (PSM) can be seen as a combination of process-based Model of Computation (e.g. Kahn Process Networks) and hierarchical state-based models. PSM presents a hierarchical composition of levels, where at each level there is either: (i) composition of concurrent processes communicating through channels; or (ii) state machine. In our approach, we restrict our interest on a subset of PSM covering the only concurrent processes communicating through channels.

III. CBSE OVERVIEW

There are several classifications based upon we can categorize component technologies. We propose following two categorizations.

First classification is based upon the organizations that introduced these technologies shown in Table I. There are three organizations that are leaders in the component based SW engineering, namely Object Management Group (OMG), Microsoft and Sun Microsystems (nowadays Oracle). OMG has devised a number of component based technologies including Common Object Request Broker Architecture (CORBA), CORBA Component Model (CCM) and Model Driven Architecture (MDA). CORBA and CCM are highly adopted distributed systems; however they impose large overheads on the runtime platform. Therefore, they will not be discussed in this survey because target platform are MPSoC, which stringent constraints on the runtime platform. In addition, OMG announces the new standards and profiles for Unified Modeling language (UML), which is de facto standard for modeling and documenting both software as well as HW. UML is described in more detail in Section IV. On the other hand, Microsoft introduces the Component Object Model (COM), COM+, .NET and Object Linking and Embedding (OLE) as a middle platform for Microsoft developed applications. Finally, Sun introduces JavaBeans for desktop components and Enterprise Java Beans for distributed systems. Both of these approaches, Microsof and Sun cannot be applied for MPSoC.

Second classification is based upon two conditions, i.e. target application domain and technology origin, either industry or academia, Table II. Target application domain (columns) can be general-purpose, distributed or real-time. General purpose component technologies include Fractal, SOFA and C2 style architectures. All of these technologies are developed in Academia. Distributed component models include CORBA and Enterprise JavaBeans (EJB). These models can be applied only to distributed systems and there is no point in modeling MPSoC with them, therefore they are not discussed in this paper.

Real-time component technologies target the automotive or aerospace systems or consumer's electronics. They can be further divided into the ones developed in Academia or Industry. Industrial real-time component technologies are generally designed according to the specific products and needs of companies that developed them. Therefore, these technologies are not used by the wide audience and only small subset of applications can be described using them. Therefore they will be described briefly as follows. Among industrial real-time technologies, we examine Koala, Rubus, PECOS, Autosar and AADL.

Koala [24] component technology is developed by Philips for consumer's electronics. Koala follows the product-line architecture with a focus on a explicit set of related products [9]. Product line architectures introduce *variation points* to create variation architectures which correspond to specific products. Variation points include design decisions that differentiate single product from another.

Rubus Component Model [25] is developed in Articus and it is used by Volvo Construction Equipment. This component technology incorporates several tools, e.g. scheduler, graphical environment for application design.

AADL [23] (Architecture analysis and design language) is an architecture design language intended for modeling automotive and avionics systems. AADL enables modeling, as well as schedulability and flow control analysis. However, because of this target application domain, which is not suitable for modeling DSP applications, it is not further discussed in this paper.

Academia - developed component technologies include PECT and Robocop. Robocop is not discussed because this component technology is not updated since 2006, while this is a survey of the state of the art component technologies.

TABLE I. ORGANIZATION-BASED CLASSIFICATION OF COMPONENT TECHNOLOGIES

OMG	Microsoft	Sun
CORBA CCM MDA	Component Object Model (COM) COM+ .NET OLE	JavaBeans Enterprise JavaBeans (EJB)

TABLE II. DOMAIN-BASED CLASSIFICATION OF COMPONENT TECHNOLOGIES

General-purpose	Distributed	Real-time	
Academia		*Academia*	*Industry*
Fractal SOFA 2 C2 style architecture-1996	CORBA Enterprise JavaBeans (EJB)	PECT (CMU) Robocop (Eindhoven) - 2006	Koala (Philips)-2000 Rubus (Volvo) PECOS (ABB) Autosar AADL(SAE)

IV. UML

Unified Modeling Language [3] is a standard tool for modeling and documenting software projects. UML includes thirteen basic diagrams for modeling both structure and behavior of the target SW product. For structural modeling UML enables following diagrams: Class diagram, Object diagram, Component diagram and Composite structure diagram. On the other hand, for modeling behavior there are: Use case diagram, Sequence diagram, Communication diagram, State diagram, Activity diagram, Deployment diagram, Package diagram, Timing diagram and Interaction overview diagram, [4].

UML diagrams are general and provide a standard syntax for describing a variety of aspects of software systems. However, basic UML provides very few semantics to go along with these syntax rules. For example, according to the interpretation, the same element in the diagram could have entirely different semantics.

1) *UML component diagrams*

As already mentioned above, UML component diagrams are used to specify the structure of particular SW system. In order to specify either the communication protocol that interface has to follow; or behavior of particular components, additional UML diagrams are needed.

For specification of the communication protocol, sequence diagrams are often used.

In order to cope with lack of semantics, various UML profiles are devised for domain specific modeling. UML profiles extend the existing UML models by specifying stereotypes. A stereotype is an extension of the vocabulary of the UML, allowing user to create new kinds of building blocks similar to existing ones but specific to the problem. Graphically, a stereotype is rendered as a name enclosed by guillemets (French quotation marks of the form « »), placed above the name of another element, Fig 2.

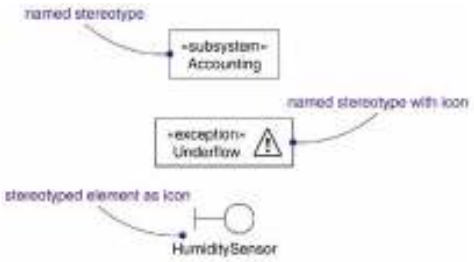

Figure 2. UML stereotype, [3]

There are four UML profiles of particular interest here, namely UML SysML, UML MARTE, UML SoC and UML SPT.

B. *UML SysML*

UML SysML [5] is a general UML profile for modeling engineering systems from both HW and SW point of view. SysML includes following UML diagrams: activity, use case, sequence and state machine for behavior description; and block, internal block and requirement diagrams for structure description, Fig 3. Block structure is extension of the class diagram, while internal block structure extends the component structure diagram. Main key points of the SysML are: (i) SW design is controlled by the requirements diagram, (ii) model elements are grouped into packages, and (iii) interoperability with UML through XMI interchange.

Authors in [9] argue that SysML does not solve the question of the lack of semantics in UML. SysML does also not define always precise semantics for the modeling elements it introduces.

1) *Takeaways*

UML SysML does not include the key diagram for component based design – component diagram. Basic structure unit is a block, which is an extension of a class.

C. *UML SoC*

UML SoC [6] profile targets modeling System-on-Chip. UML SoC stereotypes can be directly mapped into SystemC constructs. In addition to already defined basic UML diagrams, UML SoC redefines the structure diagram with specific SystemC-like graphical symbols. The following elements are used: (i) Module and Module Part, (ii) Port (Single or Multiple), (iii) Protocol Interface, (iv) Channel Part, (v) Connector, (vi) Protocol, (vii) Process, (viii) Clock

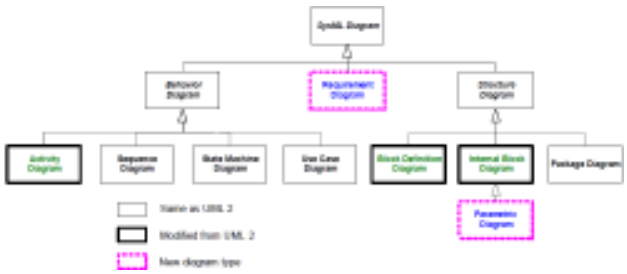

Figure 3. SysML diagrams, [5]

Port, (ix) Reset Port, (x) Clock Channel, (xi) Reset Channel and (xii) Data Type, Fig 4. According to the UML SoC specification, SystemC could be automatically generated. More generally, UML SoC can be seen as a graphical frontend for SystemC-based MPSoC design.

In addition, authors in [16] used the UML SoC to model 802.11.a physical layer transmitter and receiver described at instruction level. The system platform used for implementation is composed of an ARM processor and a dedicated hardware coprocessor that implements an FFT operation, which is a small MPSoC.

1) Takeaways

As it can be seen, UML SoC enables co-design of HW and SW, while PSM model is only used to specify the application. A subset of the UML SoC used for application constructs specification include Processes and Channels. This is the same abstraction level as the PSM, and therefore, we conclude that UML SoC is not a suitable candidate for model transformation into PSM.

D. UML MARTE

UML profile for Modeling and Analysis of Real-time and Embedded systems (MARTE) [7] profile is intended for design of real time embedded systems. MARTE specification itself is free, while tools are proprietary for OMG and partners.

Specification is extensive; however there are few key concepts that characterize MARTE: non-functional properties, time modeling and allocation modeling, Fig. 5. Allocation modeling encompasses both spatial application to platform mapping, and on the other hand application scheduling, i.e. time allocation.

1) Relaetd work

In [18] MARTE is used as a high-level specification for MPSoC design. MARTE is used to describe both application, platform and application-to-platform mapping. Similar to the second goal of this study, the authors in [18] restrict the domain of MARTE to dataflow applications. However, the emphasis in this paper is not on MARTE, but on the exploitation of the data parallelism for the multimedia applications and applying the MARTE repetitive structure in order to achieve this. As a case study, H263 application is synthesize on two platforms. First platform calles QuadriPro is composed out of four processing units, a RAM and ROM and interconnect used to connect memories and processors. Second platform assumes a distributed memory. It is organized as toroidal 3x3 grid of processing nodes, where each node holds its own memory. Application task T is tiled into multiple application tasks T_i, where these new tasks are allocated onto platform.

2) Takeaways

In comparison to PSM, MARTE is a lower level model that incorporates notions of time and non-functional properties. Therefore, transformation of MARTE into PSM can not be one-to-one and MARTE key concepts have to be omitted.

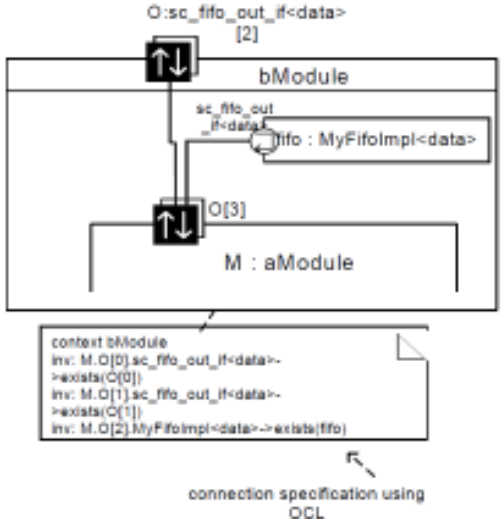

Figure 4. UML SoC example, [6]

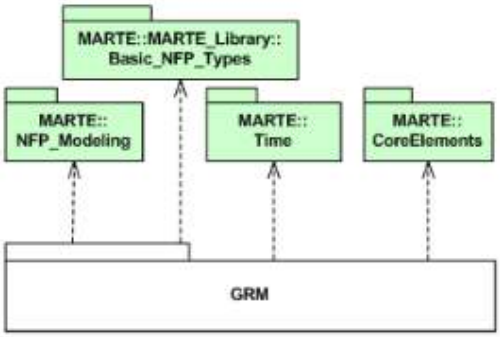

Figure 5. UML MARTE structure, [7]

E. UML SPT

UML SPT [8] is UML profile for schedulability, performance and time. Profile includes sequence diagrams that represent notion of time. Target domain are queue modeling, schedulability and performance measurement. The profile is intended for analysis, rather than design of applications.

1) Related work

Design flow that takes the UML SPT as a specification and generates mappings for Simulink-based MPSoC sythesis is proposed in [22]. Proposed synthesis tool automatically handles processor allocation, mapping of threads to processors, and insertion of required Simulink temporal barriers, ports, and dataflow connections.

F. UML Conclusion

UML is highly adopted modeling language with a variety of profiles specific for certain domains, e.g. real time systems, general-purpose engineering systems etc. Basic UML diagrams lack the concrete semantics. In addition,

behavior specification with UML diagrams is achieved through use-case, sequential or activity diagrams. Considering the design of DSP applications, it is easier to describe the application behavior using C processes, rather than these diagrams.

We have described four UML profiles, from which MARTE and SPT are driven more to the system analysis in terms of schedulability and performance, rather than system modeling. However, PSM is a MoC which provides means of the application modeling, only. If we alleviate this non-functional features from MARTE and SPT profiles, we obtain pretty much basic UML semantics. On the other hand, SysML is a general-purpose UML profile for modeling engineering system from both HW and SW point of view. The goal here is to describe the SW. In order to use it as MPSoC specification and consequently transform it to PSM, only a subset of SysML would be employed. Moreover, as already stated above, SysML does not solve the lack of semantics, it only broadens the UML for engineering systems. And finally, SoC presents the graphical environment for SystemC modeling of systems on chip from HW and SW point of view. In comparison to PSM, SoC is concerned with modeling lower level implementation details that take into account modeling platform elements (e.g. processors, memories and buses), application to platform mapping and communication routing.

Moreover, there are already several MPSoC implementations that take MARTE [18] or SPT [22] as a specification. Therefore, by applying these models would not yield a significant contribution. Because of all this, we conclude not to use UML as a specification for MPSoC

V. Common Features

This chapter provides a description of terminology and common features that are available in the state of the art component technologies. These features are discussed with respect to the goals of this paper. Further sections describe particular component technologies and emphasize which out of these features are supported and differences how these features are implemented.

A. Interfaces

Interfaces can contain either operations or variables. Operations correspond to the function calls with corresponding return type and parameters. Variables correspond to signals which can be exchanged between particular components. Components can implement various interfaces. From another point of view, interfaces can be required or provided services.

Required services present the inputs of the component, which are necessary for component's execution. Provided services present the output of component, which component delivers to the other components with compatible interface.

Compatibility of interfaces is another issue that has to be specified by the component technology. Some technologies impose loosely defined rules for communication. In such schemes, particular interface that are connected do not have to specify the same services, i.e. it is only important that they specify a common subset of services. From the point of DSP applications as well as MPSoC specification it is better to define compatibility as the equivalence of the interfaces.

B. Dymanic reconfiguration

Dynamic reconfiguration is a concept of adding new components and connectors or replacing the existing ones at runtime. From the aspect of applicability to the MPSoC platform, where the SW architecture has to be fixed in order to be downloadable and mapped onto platform.

C. Component repository

Component repository contains the already designed components, which can be used for overall system composition at the design time and during deployment. However, since MPSoC are resource constrained, only design-time composition is desirable. Component repository is a key feature that enforces the component reusability.

D. Component framework

Component framework is an optional facility used to support components during runtime. It offers a variety of features that enable the component communication and synchronization mechanisms, like mutual exclusion facilities. From the MPSoC - implementation point of view, component framework presents the additional overhead on the resource-constrained MPSoC platform and the goal is to choose the component technology independent of the underlying SW platform.

VI. Component Technologies

State of the art component technologies are described in following sections according to the order of preference with respect to the objectives stated in Section II.. Namely, we describe Fractal, SOFA 2, SOFA HI and PECT component technologies. For each component technology, we report what are components, how are interfaces defined, what communication protocol is used and what are the tools at our disposal. A short conclusion regarding the described technologies is given in the *Takeaways* section.

Another approach to component based design is presented in [9]. In this paper, component based design is seen as an approach for SW architecture. Basic SW architecture unit is denoted as a component, and SW architectures are designed by composing components that communicate through interfaces. With respect to that we give a short overview of a C2-style architecture.

A. Fractal

Fractal [12] is a general-purpose component based technology which can be used to design wide specter of applications from operating systems or middleware to graphical user interfaces. It is supported by the OW2 organization [11]. There are two key concepts characterizing this model: recusivity and reflexivity. Recursivity enables modeling of hierarchy, i.e. components can be nested in composite components - hence the "Fractal" name.

Reflexivity includes two key concepts: runtime SW system reconfigurability and monitoring. For example, number of subcomponents or attributes that describe particular component can be changed at runtime.

1) Components

Fractal components are structured in terms of content and controller, Fig 6. The content part defines a finite number of components, called sub components, which are under the control of the controller of the enclosing component. The controller part is more than the component interface in the sense of binding and communication; it embodies the broader control behavior associated with a particular component. The component model defines four kinds of control interfaces defined as follows:

- Binding Controller (BC): provides operations to bind and unbind interfaces of the component,
- Content Controller (CC): allows listing, adding and removing sub-components in the component content.
- Life-cycle Controller (LC): allows explicit control on the component execution (e.g. start and stop operations) and
- Attribute Controller (AC): allows reading and writing the component attributes from its outside.

Component communication is realized through the operation calls bound to interfaces.

Fractal enables the sharing of components between multiple composite components. However, when designing the DSP applications, this is not necessary useful.

In order to model provided and required services, interfaces are assigned roles. Interfaces are defined in Cecelia IDL which follows syntax similar to Java programming language. In order for C functions to implement the interfaces, specific concretcts are added in order to separate communication from computation.

Communication protocol is user-defined, i.e. user has the liberty to define and implement the communication calls to the interface methods.

2) Ralated work

In [26], EMBera component based technology, which is based upon Fractal, is implemented on two MPSoC platforms. EMBerra components are active entities, each one having a separate thread of control. Components communicate through asynchronous message passing mechanism, where provided and required interfaces are implemented through (i) blocking send and receive, and (ii)pointers to the FIFO data structure, named mailbox. Depending on the target implementation platform, implementation of provided and required interfaces differs.

MJPEG decoder was used as a benchmark when implementing it on two platforms. First platform is 16-core SMP Linux, which is standard x86 multiprocessor architecture. The 16-core platform is a Symmetric Multiprocessor eight dual core AMD Opteron 2.2 GHz and 2 MB of cache memory for each processor. An EMBera application is a Linux user process. A component is a data structure and a POSIX thread. This thread belongs to the Linux user process and provides an execution support for the code inside the component.

STi7200 MPSoC platform composed out of one 450 MHz general purpose RISC processor and four 400 MHz accelerators. General-purpose CPU and the accelerators communicate through shared memory. Both processor and accelerators runOS21, lightweight real-time multitasking operating system. OS21 tasks communicate through specific middleware which manages shared memory communication. The component provided interface is represented by a distributed object. The component required interface corresponds to pointers towards a distributed object. A connection between both interfaces is established using EMBX primitives to manage distributed objects. When a component needs to communicate through a required interface, it executes a send call implemented in OS21 middleware and thus updates the corresponding distributed object. As the distributed object represents a provided interface, the component providing interface needs to execute middleware *receive* in order to end the communication.

3) Tools

Fractal includes following four tools: (i) Fractal ADL, (ii) Fractal GUI editor, (iii) Fractal Explorer console and (iv) Fraclet.

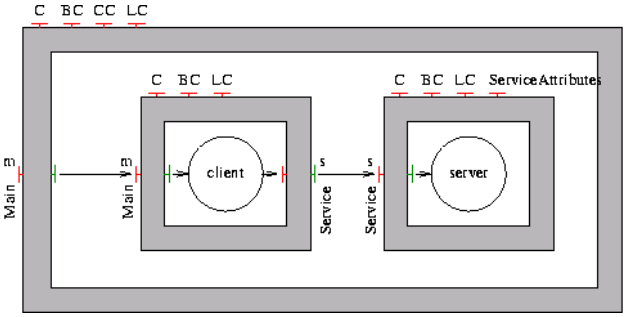

Figure 6. Fractal application example, [12]

Fractal ADL parser is a tool made of several Fractal components that can describe and parse Fractal ADL architecture definitions, and instantiate the corresponding components. Fractal GUI editor is a tool made of several Fractal components that provides a graphical tool to design and edit Fractal component configurations. Fractal Explorer console is a tool that provides a way for reconfiguring and managing Fractal-based applications at runtime. Fraclet provides an annotation-based programming model to leverage the development of Fractal components.

B. SOFA 2 and cousins

SOFA 2 [13] component technology provides ADL-based design, behavior specification and verification based on behavior protocols, software connectors supporting different communication styles and providing transparent

distribution of applications. SOFA HI is a light weighted version of the SOFA 2 component technology, intended for real-time embedded systems, [14].

1) Component

Similar to Fractal, components are either primitive or complex. Primitive components correspond to the single source code function, while complex components represent hierarchical composition of underlying components, Fig 7.

Interfaces include either variables or operation calls. Component can implement multiple interfaces. Interface definitions are written in Interface Definition Language (IDL). Each interface has to specify the communication style which it employs, either local method invocation or remote procedure call. Binding types include connections, delegation and subsumption. Delegation and subsumption bindings correspond to the relationship between component and subcomponent in the hierarchy. Delegation is straightforward, where several operations of the component are delegated and performed accordingly by the subcomponent. Subsumption relates to the concept where the interface of the component contains at least all the operations listed in the subcomponent required interface. That is to say that there could be some extra operations in the component's required interface, which are not used by the subcomponent.

At the time being, SOFA HI doesn't support the code generation techniques from defined components and interfaces, but authors in [14] state that in could be implemented. SOFA 2 enables code generation according to the Java programming language. Examples for the SOFA HI are given for space craft application [14].

2) Commuication protocol

Communication protocol according to which components exchange information using interfaces is not strictly specified. More specifically, SOFA introduces the concept of control part, which does not contribute to the functionality of the component, but it is used to maintain component's life cycle, component updates and component bindings. Set of controllers is extensible. For example, controllers that are mandatory for each component are *LifecycleController* which is in charge of controlling the component lifecycle, and *BindingController* which is in charge of connections with other components.

However, there are no strict rules how to define *BindingConroller* in order to handle the communication protocol. This is left to the application designer, Fig 8.

In the case of the SOFA 2 components, multiple requests that appear at the same time are handled using the underlying JVM. As opposed to SOFA 2, SOFA HI mediates accessing to the RTOS and hardware through a service accessible to all components, in order to keep control over all resources hardware interactions.

3) Runtime facilities

However, in comparison to previously described component technologies, connectors are runtime entities which are generated according to the component's interface description and configuration properties.

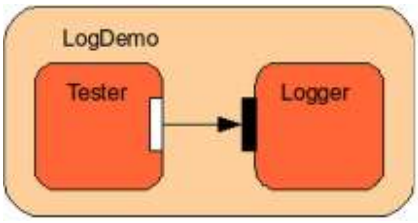

Figure 7. SOFA application example, [15]

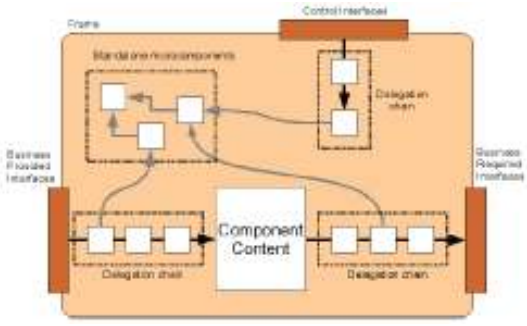

Figure 8. Control part of the SOFA component, [15]

In SOFA HI, dynamic reconfiguration is supported in a sense of replacing the existing components with the updated ones at the runtime. On the other hand SOFA 2 supports complete dynamic reconfiguration in sense of adding components and connectors.

4) Tools

SOFA includes following tools (i) Cushion, (ii) SOFA IDE and (iii) MConsole. Cushion is a text-based tool which allows development of SOFA 2 applications and manipulation with a repository. SOFA IDE is a graphical tool, plugin for Eclipse. MConsole is a plugin for Eclipse (as well as a standalone application) monitoring and maintaining SOFA 2 runtime environment.

5) Takeaways

When we abstract the dynamic reconfiguration facilities and Controller facilities that manage the life cycle of a component, we obtain a component model that is not much different from the basic UML component diagram semantics. Moreover, communication protocol is left to the user for definition, while underlying platform should take care of possible multiple simultaneous requests, either RTOS or JVM.

C. C2-style architecture

C2 architecture style [10] embodies complex layered style, where component communicate through shared connectors, Fig. 9. Component presents the encapsulated unit of behavior which can be connected to exactly one connector. Connectors can be connected to other connectors, where connectors can be seen as layers in the overall system architecture. This style is useful when modeling applications that conform the "Model View Controller" design pattern, e.g. applications including the graphical user interface. In such applications specific layer presents the graphical user

interface (view), data model behind the application an controller level that includes the logic that connects the view and data model. Components communicate through messages which can be either requests or notifications. Components are aware only of the components in upper layers towards which they send the requests for services. On the other hand, components send the notifications about their state downwards without the explicit knowledge of what lies beneath.

1) Takeaways

C2-style architectures are tailor-made for modeling graphical user interfaces. However, DSP applications are too simple to be modeled using layered architecture imposed by C2. When modeling DSP applications in this way, we would have either one layer with all components in it, or for each component specific layer. Communication protocol controlling the communication on a specific layer is not clearly specified.

D. PECT

Pervasive Component Technology (PECT) is a component technology that emphasize the analysis over specification. Specification is given within the Pin component language, [21].

1) Component

Components are specified according to the Pin component technology. Component is considered to be a compiled peace of code. Component behavior is specified through UML statecharts and component's are passive and need to be triggered to start the execution [20]. Inter-component communication is realized through set of pins that are connected using restricted set of connectors. Each pin has a set of predefined methods assigned that are used to control the component instance lifetime, configuration or reaction time.

2) Communication protocol

Communication protocol employed by PECT specified by the statechart representing the component behavior. For example, component C is comprised out of three states, namely *ready*, *work* and *log*, Fig 10. Edges of the statechart are guarded by conditions which include required pins (*toc*), while actions performed by state transitions include calling messages of provided pins (*tic* and *talk* pins). PECT enables the component framework used to support components at runtime, which provide the set of API calls for component communication. PECT component framework assumes family of Windows operating systems based on the Win32 API, i.e. Windows NT, Windows 2000, and Windows XP.

PECT enables formal verification and worst case execution time analysis through three reasoning frameworks, i.e. ComFoRT, λ_{ABA} and λ_{SS}.

3) Takeaways

Architectural style employed by PECT component technology is not suitable for modeling DSP applications, because it requires specifying the component via statecharts.

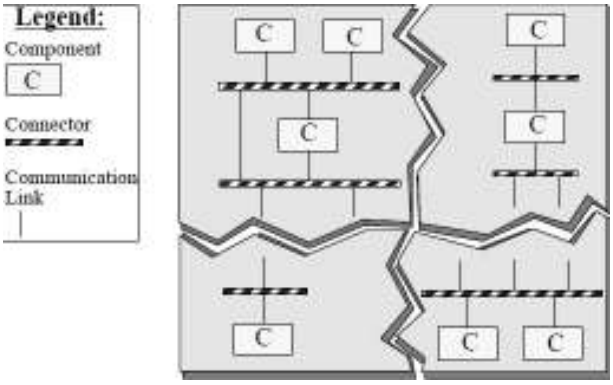

Figure 9. C2 architectural style, [10]

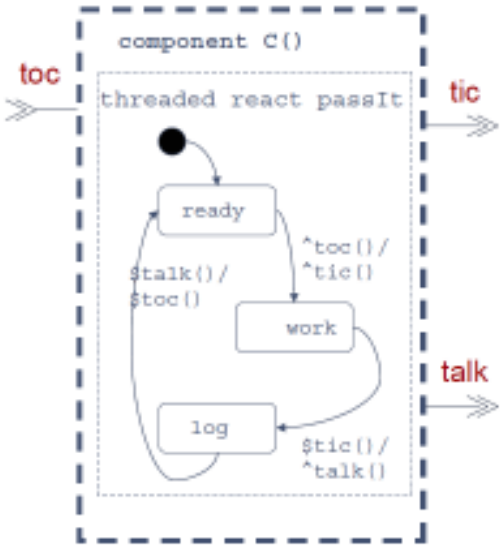

Figure 10. PECT component, [20]

That is because it is more natural to describe the DSP component's behavior through C processes. From this point of view, PECT is on the same level of abstraction, as the PSM. Authors in [19] argue that in comparison to the Program state machines, where sequential processes have clear notion of completion, states in the statecharts do not satisfy this property. Main contribution of PECT is that it enables formal verification, which is useful, but not crucial for our further work. In addition, PECT framework providing the underlying infrastructure that supports components at runtime, is bounded to the Windows infrastructure not suitable for embedded implementation.

VII. DISCUSSION

An overview of the presented technologies is given in Table III. We evaluate SOFA HI, Fractal and PECT technologies according to the following features: interface specification, interface definition language, communication protocol and code generation abilities.

From, the aspect of interface specification, we can see that both SOFA HI and Fractal represent interfaces through method calls and signals. PECT interfaces are called pins, which are assigned predefined message passing calls.

As interface description language, SOFA HI uses the XML, while Fractal uses the IDL definition. PECT uses the intermediate Pin component language

Communication protocol is a very important feature, because graphical component specification only represents structural organization. SOFA HI enables the user-defined communication protocol, where provided and required services (method invocations and signal transfer) are not strictly specified and organized according to user's need. By user-defined protocol, we assume the communication protocol that does not follow any predefined mechanism like statechart; it is rather an arbitrary peace of code written by user. From the implementation point of view, we will use SOFA HI to model dataflow applications and transform it into PSM. PSM uses the blocking reads and writes as a communication protocol. Therefore, we have to restrict the user defined SOFA HI communication protocol into some model that can be easily transformed to blocking reads and writes. From the SW engineering point of view, user-defined communication protocol is desirable, because when designing DSP applications, user wants to have freedom to define the system behavior. Similar to SOFA, Fractal also employs the user-defined communication protocol, while PECT communication protocol complies the statechar describing the component's behavior.

From the language mappings point of view, SOFA HI does not provide clearly defined mappings. However, SOFA HI is based upon SOFA 2 technology that provides mappings for Java programming language. Java requires the Java Virtual Machine environment in order to execute on any platform. From the point of implementation on the MPSoC platform, it is more efficient to use C or C++. Therefore, the authors would have to write the language mappings for C or C++. On the other hand Fractal provides wide range of language mappings, spreading from Java, C, .NET, Smalltalk to Python. PECT provides the mappings for C programming language and it is bounded to Windows operating system infrastructure, not suitable for implementation on embedded platform.

TABLE III. COMPARISON OF COMPONENT TECHNOLOGIES

	SOFA HI	**Fractal**	**PECT**
Interface	Methods & signals	Methods and signals	Predefined API calls
Interface definition language	XML	Cecelia IDL (for C)	Pin intermediate language
Communication protocol	User-defined	User defined	Statechart defined
Code generation	-	Java, C, .NET, SmallTalk, Julio	C

VIII. CONCLUSION

This paper gives an overview of the state of the art component engineering technologies and UML profiles. Component technologies are classified taking into account two conditions; (i) target domain, which is DSP applications, (ii) target platform, which is MPSoC, and (iii) target transformation model, which is Program state machines. As a result of discussion provided in the Section VII, that all the presented technologies offer similar common functionalities like separation of communication and behavior, dynamic reconfiguration and graphical definition. However, the conclusion is that Fractal is the most suitable candidate for modeling MPSoC systems. It provides flexibility required for modeling DSP applications from the SW engineering point of view, and at the same time provides mappings for the C programming language.

For further work, we propose design of algorithm for transformation of the Fractal into PSM, which is further synthesized for MPSoC implementation.

ACKNOWLEDGMENT

The work presented in this paper is developed within the Application-oriented Embedded System Technology project supported by the Unity through Knowledge Fund and Center for Embedded Computer Systems (CECS) at the University of California at Irvine. Authors wish to thank Prof. Daniel D. Gajski for advices and support.

REFERENCES

[1] D. Gajski, F. Vahid, S. Narayan, and J. Gong. Specification and Design of Embedded Systems. Prentice-Hall, July 1994

[2] Center for Embedded Computer Systems (CECS). Embedded System Environment, Center for Embedded Computer Systems, University of California, Irvine.[Online] http://www.cecs.uci.edu/~ese, 2008.

[3] G. Booch, J. Rumbaugh and I. Jacobson, The Unified Modeling Language User Guide SECOND EDITION, 2nd ed, Addison Wesley Professional, 2005

[4] Object Management Group (OMG). Unified modeling language (UML). [Online] http://www.uml.org/

[5] Object Management Group (OMG). OMG Systems Modeling Language (SysML). [Online] http://www.omgsysml.org/

[6] Object Management Group (OMG). UML Profile For System On A Chip (SoC), V 1.0.1. [Online] http://www.omg.org/technology/documents/formal/profile_soc.htm

[7] Object Management Group (OMG). The Official OMG MARTE Web Site, Modeling and Analysis of Real-time and Embedded systems. [Online] http://www.omgmarte.org/

[8] Object Management Group (OMG). UML® Profile For Schedulability, Performance, And Time, Version 1.1. [Online] http://www.omg.org/technology/documents/formal/schedulability.htm

[9] R. Taylor, N. Medvidovic and E. Dashofy, Software Architecture: Foundations, Theory, and Practice. John Wiley & Sons, ©2009 736 pages.

[10] N. Medvidovic et al. Using Object-Oriented Typing to Support Architectural Design in the C2 Style. In Proceedings of the ACM SIGSOFT '96 Fourth Symposium on the Foundations of Software Engineering. p.24-32, ACM SIGSOFT. San Francisco, CA, October, 1996.

[11] OW2. OW2 consortium, Leading opensource middleware, [Online] http://www.ow2.org/

[12] OW2 consortium, The Fractal Component Model specification draft, Februray 2004, [Online] http://fractal.ow2.org/specification/index.html

[13] Bures, T., Hnetynka and P., Plasil, F.: SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model, Proceedings of SERA 2006, Seattle, USA, IEEE CS, ISBN 0-7695-2656-X, pp.40-48, Aug 2006

[14] Prochazka, M., Ward, R., Tuma, P., Hnetynka and P., Adamek, J.: A Component-Oriented Framework for Spacecraft On-Board Software, Proceedings of DASIA 2008, May 2008

[15] O. Černý, P. Hošek, M. Papež, and V. Remeš. SOFA 2 Users guide, [Online] http://sofa.ow2.org/docs/pdf/users_guide.pdf

[16] Mueller, W., Rosti, A., Bocchio, S., Riccobene, E., Scandurra, P., Dehaene, W., and Vanderperren, Y. 2006. UML for ESL design: basic principles, tools, and applications. In Proceedings of the 2006 IEEE/ACM international Conference on Computer-Aided Design (San Jose, California, November 05 - 09, 2006). ICCAD '06. ACM, New York, NY, 73-80.

[17] Martin, Grant; Müller, Wolfgang (Eds.). UML for SOC Design, 2005, XII, 272 p., Hardcover, ISBN: 978-0-387-257

[18] J. Dekeyser, I. Quadri, Abdoulaye Gamatié. Tutorial: Using the UML profile for MARTE to MPSoC co-design dedicated to signal processing. In Colloque International Télécom 2009 et 6cmes Journées JFMMA, Agadir, Morocco, March 2009.

[19] Gajski, D., Abdi, S., Gerstlauer, A., Schirner, G. “Embedded System Design. Modeling, Synthesis and Verification”, 2009, 358 p.,ISBN: 978-1-4419-0503-1

[20] S. Hissam, J. Ivers, D. Plakosh and K. C. Wallnau, Pin Component Technology (V1.0) and Its C Interface,Technical Note CMU/SEI-2005-TN-001, April 2005

[21] Hissam, S., Moreno, G., Stafford, J., and Wallnau, K. 2003. Enabling predictable assembly. J. Syst. Softw. 65, 3 (Mar. 2003), 185-198

[22] Brisolara, L. B., Oliveira, M. F., Redin, R., Lamb, L. C., Carro, L., and Wagner, F. 2008. Using UML as front-end for heterogeneous software code generation strategies. In Proceedings of the Conference on Design, Automation and Test in Europe

[23] Singhoff, F., Legrand, J., Nana, L., and Marcé, L. 2005. Scheduling and memory requirements analysis with AADL. In Proceedings of the 2005 Annual ACM Sigada international Conference on Ada (Atlanta, GA, USA, November 13 - 17, 2005). SigAda '05. ACM, New York, NY, 1-1

[24] Van Omering, R.; Van der Linden, F.;Kramer, J.; ”The Koala component model for consumer electronics software”, I*EEE Computer,* 33(3), pp. 78-85, March 2000

[25] Norstorm, C et al; ”Experiences from Introducing State-of-the-art Real-time Techniques in the Automotive Industry”, Proceedings 8th IEEE Int’l conference on engineering of Computer-based Systems. IEEE 2001.

[26] C. Prada-Rojas, V. Marangozova-Martin, K. Georgiev, J-F. Mehaut, M. Santana, "Towards a Component-Based Observation of MPSoC," Parallel Processing Workshops, International Conference on, pp. 542-549, 2009