A Classification Framework for Component Models

Ivica Crnkovic, Séverine Sentilles, Aneta Vulgarakis, Michel Chaudron

Mälardalen University, Sweden

PROGRESS
A national Swedish Strategic Research Centre
What is component?

• The component case
  – Many definitions
  – Some known ones:
    • software component is a unit of composition with contractually specified interfaces and context dependencies only. A software component can be deployed independently and is subject to composition by third parties.
      Szyperski

    • A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard
      Heineman and Councill

  – Intuitive perception may be quite different at different levels (model, implementation, run-time)
Different solutions

Many CB models (with many different characteristics) exist today.
Questions

– What is common to component models?

– It is possible to identify common principles and common features?

– Is it possible to utilize/instantiate these principles for particular component models in particular domains?
Goal

• Propose a classification framework for component models
  – Identify characteristics and categorize them
  – Illustrate its use by providing a survey of a number of component models
Definitions:
Software Component – Component Model

Definition:
• A Software Component is a software building block that conforms to a component model.
• A Component Model defines standards for
  – (i) properties that individual components must satisfy and
  – (ii) methods, and possibly mechanisms, for composing components.
Some definitions first…

CBS = <C,B,P>

- C = \{C_i\} - components
- B = \{B_j\} - bindings
- P = system platform

- Bindings
  - Between components and the platform -> components deployment
  - Between components – components binding

A Component model defines standards for (i)properties that individual components must satisfy methods for composing components.
More definitiones

- **Component Specification**
  \[ C = \langle \{\text{Interfaces}\}, \{\text{Properties}\} \rangle \]

- **Component compliance to component model**
  \[ C \models CM \Rightarrow I, P \models CM \]

- **Component Composition:**
  \[ A = \langle C_1 \oplus C_2 \rangle \Rightarrow I = \langle I_1 \oplus I_2 \rangle \land P = \langle P_1 \oplus P_2 \rangle \]

- **Interface composition (BINDING):** \[ I(C) = I(C_1) \oplus I(C_2) \]
- **Property composition:** \[ P_i(C) = P_i(C_1) \oplus P_i(C_2) \]
Classification

• How to describe
  – (i) Commonalities?
  – (ii) Differences?

• Different approaches
  – Specification of Meta model
  – List of characteristics
  – Identification of categories and their characteristics
The Classification Framework - Categories

• Three dimensions
  – Lifecycle.
  – Construction.
  – Extra-Functional Properties.
The Classification Framework - Categories

• Three dimensions
  – **Lifecycle.** The lifecycle dimension identifies the support provided (explicitly or implicitly) by the component model, in certain points of a lifecycle of components or component-based systems.

• **Construction.** The construction dimension identifies (i) the component interface used for the interaction with other components and external environment, and (ii) the means of component binding (initiate communication) and (iii) communication.

• **Extra-Functional Properties.** The extra-functional properties dimension identifies specifications and support that includes the provision of property values and means for their composition.

• **Domains.** This dimension shows in which application and business domains component models are used.
Component lifecycle

- Requirements
- Modelling
- Implementation
- Packaging
- Deployment
- Execution

Component forms

- Specification
  - Interface
  - Models
  - Meta data

- Code
  - Source code
  - Executable code
  - Executable models

- Storage
  - Repository
  - Package
  - Meta data

- Installed Files

- Executable code
Lifecycle category

Different stages of a component lifecycle

- **Modelling.** The component models provide support for the modelling and the design of component-based systems and components.

- **Implementation.** The component model provides support for generating and maintaining code. The implementation can stop with the provision of the source code, or can continue up to the generation of a binary (executable) code.

- **Storage & Packaging.** Since components can be developed separately from systems, there is a need for their storage and packaging – either for the repository or for a distribution.

- **Deployment & Execution.** At a certain point of time, a component is integrated into a system. This activity happens at different points of development or maintenance phase.
Construction (I)

1. the component interface used for the interaction with other components and external environment
2. the means of component binding (initiate communication)
3. communication.

- Specification of
  - Interface

- Composition
  - Binding
  - interaction
Interface Specification

Categories

- Levels
  - Syntactic
  - Functional Semantic
  - Behavioral

- Specification language

- Distinguish
  - Provide
  - Require

- Interface type
  - Operation-based
  - Port-based
**Binding**

\[ C_i, C_j \models CM \Rightarrow I_i, I_j, P_i, P_j \models CM \]

**Horizontal**

\[ A = \langle C_i \oplus C_j \rangle \Rightarrow I_A = \langle I_i \oplus I_j \rangle \]

**Vertical**

\[ A = \langle C_i \oplus C_j \rangle \Rightarrow I_A = \langle I_i \oplus I_j \rangle \]

where \( I_A \models CM \)

(delegation, aggregation)
Binding & composition

\[ C_i, C_j \models CM \Rightarrow I_i, I_j, P_i, P_j \models CM \]

**Vertical**

\[ A = \langle C_i \oplus C_j \rangle \Rightarrow I_A = \langle I_i \oplus I_j \rangle \]
where \( I_A \models CM \)

**Composite component**

\[ A = \langle C_i \oplus C_j \rangle \Rightarrow A = \langle I_i \oplus I_j; P_i \oplus P_j \rangle \]
where \( I_A, P_A \models CM \)

(delegation, aggregation)
Binding (II)

Endogenous

Exogenous
Interactions

Architectural style
(client-server, pipe-filter)

Communication type
(synchronous, asynchronous)
Construction classification

- **Interface**
  - operation-based/port-based
  - provides/requires
  - The interface level (syntactic, semantic, behaviour)
  - distinctive features

- **Binding**
  - Horizontal, Vertical
  - Endogenous, Exogenous

- **Interaction**
  - Architectural Style
  - Communication type (synchronous/asynchronous)
Extra-Functional Properties

• Management of extra-functional properties
  – Does a component provide any support for extra-functional properties?
  – What are the mechanisms?
  – Which properties are managed?

• Composition of extra-functional properties
  – \( P(C1 \circ C2) = P(C1) \circ P(C2) \)
  – What kind of composition is supported?
  – Which properties?
Management of EFP

<table>
<thead>
<tr>
<th>Endogenous EFP management</th>
<th>Exogenous EFP management</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td><img src="image1.png" alt="Diagram A" /></td>
<td><img src="image2.png" alt="Diagram B" /></td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td><img src="image3.png" alt="Diagram C" /></td>
<td><img src="image4.png" alt="Diagram D" /></td>
</tr>
</tbody>
</table>

EFP Managed per collaboration

EFP Managed systemwide
EPF – compositions

\[ P_i(C) = P_i(C_1) \oplus P_i(C_2) \]

Problems:
1. Composition operators?
2. Influence of external factors
1. *Directly composable properties*. A property of an assembly is a function of, and only of, the same property of the components involved.

\[ P(A) = f(P(C_1), \ldots P(C_i), \ldots, P(C_n)) \]

2. *Architecture-related properties*. A property of an assembly is a function of the same property of the components and of the software architecture.

\[ P(A) = f(SA, \ldots P(C_i)\ldots), \ i=1\ldots n \]

\[ SA = \text{software architecture} \]
3 Derived properties. A property of an assembly depends on several different properties of the components.
- \( P(A) = f(SA, \ldots Pi(Cj), \ldots), i=1 \ldots m, j=1 \ldots n \)
- \( Pi = \text{component properties} \)
- \( Cj = \text{components} \)

4 Usage-depended properties. A property of an assembly is determined by its usage profile.
- \( P(A,U) = f(SA, \ldots Pi(Cj,U), \ldots), i=1 \ldots m, j=1 \ldots n \)
- \( U = \text{Usage profile} \)

5 System environment context properties. A property is determined by other properties and by the state of the system environment.
- \( P(S,U,X) = f(SA, \ldots Pi(Cj,U,X), \ldots), i=1 \ldots m, j=1 \ldots n \)
- \( S= \text{system, } X = \text{system context} \)
Classification summary

Component model

Lifecycle
- Modelling
- Implementation
- Packaging
- Deployment

At compilation
- Interface type
  - Operation-based
  - Port-based

At run-time
- Distinction of Provides / Requires
- Interface Language
- Interface Levels
- Distinctive features

Constructs
- Interface specification
  - Interface Language
  - Syntax
  - Semantic
  - Behaviour
- Distinction
  - Provides / Requires
- Interface Types
  - Operation-based
  - Port-based
  - Syntax
  - Semantic
  - Behaviour

Interaction
- Interaction Styles
  - Synchronous
  - Asynchronous
- Communication Type

Binding type
- Exogenous / Endogenous
  - Vertical
- Exogenous
  - Collaborative
  - Endogenous
  - Systemwide
- Endogenous
  - Collaborative
  - Exogenous
  - Systemwide

Extra functional properties
- Management
  - Systemwide
  - Exogenous
  - Collaborative
  - Exogenous
- Specification
- Composition
Illustration of the Classification Framework use

• Survey of 25 component models

• Selection of documentation for each component model
  – Satisfies criteria
  – Disposability the definition (Interfaces, composition)
  – Some points in the table have been subject our interpretation.
Component models evaluations

**Selection criteria:**

1. A component model includes a component definition;
2. A component model provides rules for component interoperability;
3. Component functional properties are unambiguously specified by component interface;
4. A component interface is used in the interoperability mechanisms;
5. A component is an executable piece of software and the component model either directly specifies its form or unambiguously relates to it via interface and interoperability specification.
Chosen component models
(25 component models)

- AUTOSAR
- BIP
- BlueArX
- CCM
- COMDES II
- CompoNETS
- EJB
- Fractal
- KOALA
- KobrA
- IEC 61131
- IEC 61499
- JavaBeans
- MS COM
- OpenCOM
- OSGi
- Palladio
- PECOS
- Pin
- ProCom
- ROBOCOP
- RUBUS
- SaveCCM
- SOFA 2.0
## Lifecycle table

<table>
<thead>
<tr>
<th>Component Models</th>
<th>Modelling</th>
<th>Implementation</th>
<th>Packaging</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSAR</td>
<td>N/A</td>
<td>C</td>
<td>Non-formal specification of container</td>
<td>At compilation</td>
</tr>
<tr>
<td>BIP</td>
<td>A 3-layered representation: behavior, interaction, and priority</td>
<td>BIP Language</td>
<td>N/A</td>
<td>At compilation</td>
</tr>
<tr>
<td>BlueArX</td>
<td>N/A</td>
<td>C</td>
<td>N/A</td>
<td>At compilation</td>
</tr>
<tr>
<td>CCM</td>
<td>N/A</td>
<td>Language independent</td>
<td>Deployment Unit archive (JARs, DLLs)</td>
<td>At run-time</td>
</tr>
<tr>
<td>COMDES II</td>
<td>ADL-like language</td>
<td>C</td>
<td>N/A</td>
<td>At compilation</td>
</tr>
<tr>
<td>CompoNETS</td>
<td>Behaviour modeling (Petri Nets)</td>
<td>Language independent</td>
<td>Deployment Unit archive (JARs, DLLs)</td>
<td>At run-time</td>
</tr>
<tr>
<td>EJB</td>
<td>N/A</td>
<td>Java</td>
<td>EJB-Jar files</td>
<td>At run-time</td>
</tr>
<tr>
<td>Fractal</td>
<td>ADL-like language</td>
<td>Java (in Julia, Aokell)</td>
<td>C/C++ (in Think)</td>
<td>File system based repository</td>
</tr>
<tr>
<td>KOALA</td>
<td>ADL-like languages (IDL, CDL and DDL)</td>
<td>C</td>
<td>File system based repository</td>
<td>At compilation</td>
</tr>
<tr>
<td>KobrA</td>
<td>UML Profile</td>
<td>Language independent</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>IEC 61131</td>
<td>Function Block Diagram (FBD)</td>
<td>Structured Text (ST)</td>
<td>N/A</td>
<td>At compilation</td>
</tr>
<tr>
<td>IEC 61499</td>
<td>Function Block Diagram (FBD)</td>
<td>Language independent</td>
<td>N/A</td>
<td>At compilation</td>
</tr>
<tr>
<td>JavaBeans</td>
<td>N/A</td>
<td>Java</td>
<td>Jar packages</td>
<td>At compilation</td>
</tr>
<tr>
<td>MS COM</td>
<td>N/A</td>
<td>OO languages</td>
<td>DLL</td>
<td>At compilation and at run-time</td>
</tr>
<tr>
<td>OpenCOM</td>
<td>N/A</td>
<td>OO languages</td>
<td>DLL</td>
<td>At run-time</td>
</tr>
<tr>
<td>OSGi</td>
<td>N/A</td>
<td>Java</td>
<td>Jar-files (bundles)</td>
<td>At run-time and at compilation</td>
</tr>
<tr>
<td>Palladio</td>
<td>UML profile</td>
<td>Java</td>
<td>N/A</td>
<td>At run-time</td>
</tr>
<tr>
<td>PECOS</td>
<td>ADL-like language (C0Co)</td>
<td>C++ and Java</td>
<td>Jar packages or DLL</td>
<td>At compilation</td>
</tr>
<tr>
<td>Pin</td>
<td>ADL-like language (CCL)</td>
<td>C</td>
<td>DLL</td>
<td>At compilation</td>
</tr>
<tr>
<td>ProCom</td>
<td>ADL-like language, timed automata</td>
<td>C</td>
<td>File system based repository</td>
<td>At compilation</td>
</tr>
<tr>
<td>ROBOCOP</td>
<td>ADL-like language, resource management model</td>
<td>C and C++</td>
<td>Structures in zip files</td>
<td>At compilation and at run-time</td>
</tr>
<tr>
<td>RUBUS</td>
<td>Rubus Design Language</td>
<td>C</td>
<td>File system based repository</td>
<td>At compilation</td>
</tr>
<tr>
<td>SaveCCM</td>
<td>ADL-like (SaveComp), timed automata</td>
<td>C</td>
<td>File system based repository</td>
<td>At compilation</td>
</tr>
<tr>
<td>SOFA 2.0</td>
<td>Meta-model based specification language</td>
<td>Java</td>
<td>Repository</td>
<td>At run-time</td>
</tr>
</tbody>
</table>
## Lifecycle table

<table>
<thead>
<tr>
<th>Component Models</th>
<th>Modelling</th>
<th>Implementation</th>
<th>Packaging</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSAR</td>
<td>N/A</td>
<td>C</td>
<td>N/A</td>
<td>At compilation</td>
</tr>
<tr>
<td>BIP</td>
<td>A 3-layered representation: behavior, interaction and priority</td>
<td>Source code, implementation in BIP language</td>
<td>N/A</td>
<td>At compilation</td>
</tr>
<tr>
<td>CCM</td>
<td>Abstract model: OMG-IDL, Programming model: CIDL</td>
<td>Language independent.</td>
<td>Deployment Unit archive (JARs, DLLs)</td>
<td>At run-time</td>
</tr>
<tr>
<td>Fractal</td>
<td>ADL-like language (Fractal ADL, Fractal IDL), Annotations (Fractlet)</td>
<td>Julia, Aokell (Java) Think(C/C++) FracNet(.Net)</td>
<td>File system based repository</td>
<td>At run-time</td>
</tr>
<tr>
<td>KOALA</td>
<td>ADL-like languages (IDL, CDL and DDL)</td>
<td>C</td>
<td>File system based repository</td>
<td>At run-time</td>
</tr>
<tr>
<td>EJB</td>
<td>N/A</td>
<td>Java binary code</td>
<td>EJB-Jar files</td>
<td>At run-time</td>
</tr>
<tr>
<td>Component Models</td>
<td>Interface type</td>
<td>Distinction of Provides / Requires</td>
<td>Distinctive features</td>
<td>Interface Language</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-----------------------------------</td>
<td>----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>Operation-based, Port-based</td>
<td>Yes</td>
<td>AUTOSAR Interface*</td>
<td>C header files</td>
</tr>
<tr>
<td>BIP</td>
<td>Port-based</td>
<td>No</td>
<td>Complete interfaces, Incomplete interfaces</td>
<td>BIP Language</td>
</tr>
<tr>
<td>BlueArX</td>
<td>Port-based</td>
<td>Yes</td>
<td>N/A</td>
<td>C</td>
</tr>
<tr>
<td>CCM</td>
<td>Operation-based, Port-based</td>
<td>Yes</td>
<td>Facets and receptacles Event sinks and event sources</td>
<td>CORBA IDL, CIDL</td>
</tr>
<tr>
<td>COMDES II</td>
<td>Port-based</td>
<td>Yes</td>
<td>N/A</td>
<td>C header files State charts diagrams</td>
</tr>
<tr>
<td>CompoNET S</td>
<td>Operation-based, Port-based</td>
<td>Yes</td>
<td>Facets and receptacles Event sinks and event sources</td>
<td>CORBA IDL, CIDL, Petri nets</td>
</tr>
<tr>
<td>EJB</td>
<td>Operation-based</td>
<td>No</td>
<td>N/A</td>
<td>Java Programming Language + Annotations</td>
</tr>
<tr>
<td>Fractal</td>
<td>Operation-based</td>
<td>Yes</td>
<td>Component Interface, Control Interface</td>
<td>IDL, Fractal ADL, or Java or C, Behavioural Protocol</td>
</tr>
<tr>
<td>KOALA</td>
<td>Operation-based</td>
<td>Yes</td>
<td>Diversity Interface, Optional Interface</td>
<td>IDL, CDL</td>
</tr>
</tbody>
</table>
## Constructs table – Binding & interaction

<table>
<thead>
<tr>
<th>COMPONENT MODELS</th>
<th>INTERACTION STYLES</th>
<th>COMMUNICATION TYPE</th>
<th>BINDING TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXOGENOUS</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>Request response, Messages passing</td>
<td>Synchronous, Asynchronous</td>
<td>No</td>
</tr>
<tr>
<td>BIP</td>
<td>Triggering Rendez-vous, Broadcast</td>
<td>Synchronous, Asynchronous</td>
<td>No</td>
</tr>
<tr>
<td>BlueArX</td>
<td>Pipe&amp;filter</td>
<td>Synchronous</td>
<td>No</td>
</tr>
<tr>
<td>CCM</td>
<td>Request response, Triggering</td>
<td>Synchronous, Asynchronous</td>
<td>No</td>
</tr>
<tr>
<td>COMDES II</td>
<td>Pipe&amp;filter</td>
<td>Synchronous</td>
<td>No</td>
</tr>
<tr>
<td>CompoNETS</td>
<td>Request response</td>
<td>Synchronous, Asynchronous</td>
<td>No</td>
</tr>
<tr>
<td>EJB</td>
<td>Request response</td>
<td>Synchronous, Asynchronous</td>
<td>No</td>
</tr>
<tr>
<td>Fractal</td>
<td>Multiple interaction styles</td>
<td>Synchronous, Asynchronous</td>
<td>Yes</td>
</tr>
<tr>
<td>KOALA</td>
<td>Request response 3-Feb-11</td>
<td>Synchronous</td>
<td>No</td>
</tr>
<tr>
<td>Component Models</td>
<td>Management of EFP</td>
<td>Properties specification</td>
<td>Composition and analysis support</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------</td>
<td>--------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>BlueArX</td>
<td>Endogenous per collaboration (A)</td>
<td>Resource usage, Timing properties</td>
<td>N/A</td>
</tr>
<tr>
<td>EJB 3.0</td>
<td>Exogenous system wide (D)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Fractal</td>
<td>Exogenous per collaboration (C)</td>
<td>Ability to add properties (by adding “property” controllers)</td>
<td>N/A</td>
</tr>
<tr>
<td>KOALA</td>
<td>Endogenous system wide (B)</td>
<td>Resource usage</td>
<td>Compile time checks of resources</td>
</tr>
<tr>
<td>KobrA</td>
<td>Endogenous per collaboration (A)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Palladio</td>
<td>Endogenous system wide (B)</td>
<td>Performance properties specification</td>
<td>Performance properties</td>
</tr>
<tr>
<td>PECOS</td>
<td>Endogenous system wide (B)</td>
<td>Timing properties, generic specification of other properties</td>
<td>N/A</td>
</tr>
<tr>
<td>Pin</td>
<td>Exogenous system wide (D)</td>
<td>Analytic interface, timing properties</td>
<td>Different EFP composition theories, example latency</td>
</tr>
<tr>
<td>ProCom</td>
<td>Endogenous system wide (B)</td>
<td>Timing and resources</td>
<td>Timing and resources at design and compile time</td>
</tr>
<tr>
<td>SaveCCM</td>
<td>Endogenous system wide (B)</td>
<td>Timing properties, generic specification of other properties</td>
<td>Timing properties at design time</td>
</tr>
<tr>
<td>SOFA 2.0</td>
<td>Endogenous system wide (B)</td>
<td>Behavioural (protocols) Composition at design</td>
<td></td>
</tr>
</tbody>
</table>
Domains

Applications and business domain of the Component Models

• **General-purpose:**
  – Basic mechanisms for the production and the composition of components
  – Provide no guidance, nor support for any specific architecture.

• **Specialised:**
  – Specific application domains (i.e. consumer electronics, automotive, …)
## Domains

<table>
<thead>
<tr>
<th>Models</th>
<th>AUTOSAR</th>
<th>BIP</th>
<th>BlueArX</th>
<th>CCM</th>
<th>COMDES II</th>
<th>CompoNETS</th>
<th>EJB</th>
<th>Fractal</th>
<th>KOALA</th>
<th>KobrA</th>
<th>IEC 61131</th>
<th>IEC 61499</th>
<th>JavaBeans</th>
<th>MS COM</th>
<th>OpenCOM</th>
<th>OSGi</th>
<th>Palladio</th>
</tr>
</thead>
<tbody>
<tr>
<td>General-purpose</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specialised</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General-purpose</th>
<th>PECOS</th>
<th>Pin</th>
<th>ProCom</th>
<th>Robocup</th>
<th>RUBUS</th>
<th>SaveCCM</th>
<th>SOFA 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>General-purpose</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specialised</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• From the results we can recognize some recurrent patterns such as
  – general-purpose component models utilize client-server style
  – Specialized domains (mostly embedded systems) pipe & filter is the predominate style.
  – Composition of extra-functional properties is rather scarce.
  – Behaviour & Semantic rarely supported
  – Almost never repository

• Summary
  – The classification framework helps in understanding component models principles
  – Enables comparison
  – Can be used as a check-list when development new component models
Literature

- Ivica Crnkovic, Séverine Sentilles, Aneta Vulgarakis, Michel Chaudron,
  *A Classification Framework for Component Models*

- Ivica Crnkovic, Magnus Larsson:
  *Classification of Quality Attributes*