Comparison of Component Frameworks for Real-time Embedded Systems

Petr Hošek[1], Tomáš Pop[1], Tomáš Bureš[1,2], Petr Hnětynka[1], Michal Malohlava[1]

[1] Department of Software Engineering
Faculty of Mathematics and Physics,
Charles University, Malostranske namesti 25,
Prague 1, 118 00, Czech Republic
[2] Institute of Computer Science,
Academy of Sciences of the Czech Republic
Pod Vodarenskou vezi 2, Prague 8,
182 07, Czech Republic
{hosek, pop, bures, hnetynka, malohlava}@dsrg.mff.cuni.cz

Abstract. The usage of components brings significant help in development of real-time embedded systems. There have been a number of component frameworks developed for this purpose and some of them have already become well-established in this area. However, although the component frameworks share basic concepts and the general approach, they substantially differ in the range of supported features and maturity. This makes relatively difficult to select the right component framework and thus poses a significant obstacle in adoption of the component-based development for real-time embedded systems. To provide an overview, we present a survey in this paper, which illustrates distinguishing features of selected modern component-based frameworks for real-time embedded systems. The survey identifies features which are important for building systems from components in this area and compares these frameworks with respect to these features.

1 Introduction

With the growth of complexity of embedded systems and with the increasing stress on mass production and customization, the component-based software engineering is becoming increasingly important in this area. This is testified by growing number of component systems aiming at different application domains of embedded systems (be it AUTOSAR [**?**] in automotive industry, ROBOCOP [**?**] in consumer electronics, and emerging standards for component based systems engineering in space industry – as recently demanded by ESA[3] SAVOIR initiative).

Many years for which the components have been researched have however shown that a proper construction of a component systems is not an easy task and that it is rather difficult to balance features of a component system so that

[3] http://www.esa.int/

it can provide support and guidance through the whole application development life-cycle. This is due to many factors such as (1) the existence of two distinct but parallel flows of component and system development [**?**], (2) varying levels of concreteness (e.g. under-specified components such as UML package diagrams for showing application decomposition vs. rigorously defined development components such as with EJB), and (3) varying granularity of components (e.g. using components for modeling low-level signals vs. using components for representing independent subsystems [**?**]).

In this paper, we aim at investigating current component frameworks used in embedded and real-time systems, and at evaluating their suitability in development of such systems. We restrict ourselves to frameworks which distinguish components not only in high level design but rather throughout the whole development life-cycle. The reason is that the preservation of the components over the development life-cycle yields better traceability and allows late bindings, even to the extent of run-time reconfigurations and update [**?**].

There already exist a number of works that compare and/or evaluate component systems and define a classification of them, e.g. [**?**,**?**,**?**]. However, these works target component systems in general and do not take into account requirements for real-time domain, which have to be covered by a chosen component system (otherwise, the system could not be used for development of real-time applications).

The structure of the paper is as folows: in Section 2, we set the criteria for evaluating component frameworks in the domain of embedded and real-time systems. Section 3 presents selection of the frameworks chosen for evaluation and describes them. Section 4 evaluates described component frameworks according to the selected criteria and gives recommendations for usage in particular situations. Section 5 concludes the paper.

2 Evaluation Criteria

In order to evaluate the component frameworks, we define a set of criteria here. The criteria cover important concerns of component-based development and also explicitly stress demands of the domain of embedded and real-time systems.

For the general component-based development, we use the criteria below (related criteria are grouped together). Their selection was based on features, which are generally recognized as important for component-based development [**?**,**?**]. In addition to those criteria, we also implicitly assume the execution support, which means that the component framework allows execution of the component application – either by directly providing the execution environment, by offering synthesis of executable code or by allowing deployment to another existing component framework.

The criteria are:

- existence of a well-defined component model with support for advanced features such as hierarchical nesting, multiple communication styles and behavior and/or resource usage modeling **(C1)**

- existence of development tools or a development environment (e.g. graphical IDE) **(C2a)**
- existence of a development methodology or at least guidelines for development process **(C2b)**
- support for distributed component applications **(C3a)**
- support for dynamic reconfiguration and/or update at run-time **(C3b)**
- existence of a documentation on a sufficient level **(C4a)**
- status of a development tools and component execution support (i.e. whether they are actively developed or abandoned) **(C4b)**

In order to take into consideration also the requirements coming from the domain of embedded and real-time system, we additionally define the following criteria. They are again based on general requirements put on embedded and real-time system [**?**,**?**]

- support for coupling with hardware **(C5a)**
- small or configurable memory footprint of the run-time environment or the synthesized glue code, which are necessary to components' execution and interaction **(C5b)**
- support for modeling real-time attributes (e.g. period, budget) and scheduling analysis **(C6a)**
- support for periodic and aperiodic tasks **(C6b)**
- support for real-time scheduling of components at run-time **(C6c)**

3 Component Frameworks Overview

In this section, we briefly describe component frameworks targeting development of real-time applications for embedded devices.

As we stated in Sect. 1, we have restricted ourselves to frameworks, which have some execution support. This leaves out purely modeling approaches (e.g. Marte). We have also deliberately omitted component frameworks for enterprise systems (e.g. CCM, EJB, COM, DCOM) as well as for user interfaces (e.g. JavaBeans) and web applications (e.g. JSF). These frameworks are not suitable for development of real-time embedded systems. Additionally, we do not cover modeling, analysis, and simulation tools like Mathlab/Simulink, Scade. These could be considered under some circumstances also as component based models but their concept of a component is basically on the level of logical or mathematical operator, which makes them incomparable with classical component models that rely on components with functional interfaces.

Based on this criteria, we have chosen the following frameworks: PECOS, MyCCM-HI, Progress, AUTOSAR, Pin, Koala, ROBOCOP, THINK, SOFA HI and BlueArX.

3.1 PECOS

PECOS (Pervasive Component Systems) [?] is a project targeting domain of field devices, i.e. reactive real-time embedded systems that are typically non-distributed systems reading periodically sensors, analyzing results, and react by influencing actuators [?].

PECOS uses a component model, whose syntax is described by a meta-model and its execution by Petri nets. The meta-model enables specifying hierarchical components, component types, and property bundles that hold extra-functional properties of components.

Composition checking is implemented by generating Prolog facts during composition process and then by verifying appropriate semantic rules against a set of Prolog queries. Timing schedules were planed to be generated from timing Petri nets.

PECOS does not seem to aim at providing development methodology or advanced development tools like IDE or shared component repository. Just composition tools are available for download, run-time environment and composition compilers are not available. Project was actively developed in the past; now, according to its web pages [?], it seems to be dead.

3.2 MyCCM-HI

MyCCM-HI [?] is based on OMG LwCCM [?]. MyCCM-HI offers composite components and it supports C and Ada95 languages.

The main idea behind MyCCM-HI is transformation of application model based on component approach (described in language called COAL) to lower-level AADL Language[?]. Then, Ocarina AADL compiler [?] can be used to produce executable files or Petri nets model of the application. Ocarina libraries are also used to perform schedulability analysis [?] using the Chedar tool [?]. Distributed inter-component communication is realized by Poly-ORB-HI – middleware specialized for high integrity systems. Mode based reconfiguration of inter-component connections is also supported.

Similarly to PECOS, neither MyCCM-HI seem to be aiming at producing support tools or methodology such as development IDE, direct support for repository to enable or simplify component reuse, and systematic development methodology. On the other hand, command line tools are in mature development stage. These tools and runnable examples are freely available.

3.3 PROGRESS/ProCom

Progress is an ambitious project aiming at providing theory and tools for a cost-efficient engineering and re-engineering of distributed component based software, mostly focused on embedded systems. Because Progress is primarily intended to be used in the vehicular, telecommunication and automation industry, strong emphasis is given to time analysis and reliability of modeled systems.

PROGRESS component model (called ProCom [**?**]), which is based on SaveCCM [**?**] and Rubus [**?**], distinguishes two levels of granularity – ProSys and ProSave.

ProSave, the lower layer, operates with low-level, passive, hierarchically structured components. Computation on this level is based on the pipe-and-filter paradigm, the functionality of the ProSave component is described as a set of services. A component can have several independent (and possibly concurrently running) services. The communication between components is realized by data ports and triggering ports. Each service contains one input port group and several output port groups. Passivity means that components can not start any activity themselves. The services can be started only by triggering the input port.

ProSys, the upper layer, describes a set of concurrent subsystems. These subsystems can run potentially on several computation hardware units, called physical nodes. ProSys subsystem is composed of a set of concurrent functionality that can be either event driven (sporadic) or periodic. The only way, how ProSys subsystems can communicate with each other is sending asynchronous messages via channels. The channels are strictly typed. Channels with more message senders and more receivers are also allowed. A ProSys component may be modeled as a assembly of ProSave components.

PROGRESS is a currently running project. Thus, related analysis methods, deployment tools, etc., are not yet implemented. At present, there is only a prototype of the Eclipse-based IDE and documentation to the component model.

3.4 Autosar

AUTOSAR (Automotive Open System Architecture) [**?**] is an open industrial standard aiming at precise architecture description enabling many different manufacturers from the automotive industry to integrate their products together.

AUTOSAR distinguishes *atomic software components*, representing pieces of functionality, and *compositions*, representing logical interconnection of components. An atomic component is limited to one exucutable unit (ECU).

The standard supports two communication models – Client-Server (both blocking and non-blocking communication semantic are supported) and Sender-Receiver. AUTOSAR provides basic software support to components (including run-time environment, micro-controller abstraction or support for component communication).

Support for real-time properties is not clearly specified. Nevertheless, taking into account that AUTOSAR requires a real-time operating system, it is reasonable to assume that atomic components can use real-time primitives of the operating system.

AUTOSAR defines development process methodology; it describes development activities and their mutual dependencies.

3.5 Pin

Pin Component technology [**?**] is an initiative with the goal to provide freely distributable technology providing a basic set of features to design and implement predictable real-time software for embedded devices with support for the UML state-charts semantic.

Components in the Pin model are architectural units specifying stimulus-response behavior by a set of input ports (sink pins), output ports (source pins), and reactions to sink pins (this is very close to ProSave layer found in PROGRESS). Each Pin component consists of a container and custom code and is delivered as a dynamically linkable library with well specified interactions with environment and other components. Custom code logically consists of sink port message handlers and timeout handlers; for each reaction a single thread is created. In the current version of Pin, architectural topology is fixed at run-time, no dynamic reconfigurations are allowed, and system can not run in a distributed environment. Real-time features are provided via a support of an underlying external commercial environment.

Pin supports synchronous and asynchronous connectors, while message size and message queue lengths are fixed. Components are defined in CCL (Construction and Composition Language), the functionality is specified in the host language (C). Pin is currently ported to Windows NT and Windows CE operating systems.

Pin model is not connected to any development methodology and supporting tools like IDE or repository, but it is implemented and with a set of basic tools available for download.

3.6 Koala

Koala [**?**] is a component model for embedded devices developed by Philips[4]. Primary goals of Koala are to manage increasing complexity of software used mostly in consumer electronics and to manage its diversity.

The component model itself is inspired by COM and Darwin component models. Its basic elements are components defining set of provided and required interfaces. There is also a concept of *diversity interfaces*, which are used to capture variation points of components.

The component model supports hierarchical components (the called compound components). The components are implemented in the C language. Koala compiler is used to generate C header files responsible for connecting components together.

The component interfaces are statically connected to each other at design time, and Koala does not support reconfiguration. However, the component model offers different ways to handle diversity at design time such as switches which can be used to handle structural diversity.

[4] http://www.philips.com/

The Koala component model targets simple embedded devices, therefore it is strongly focused on optimization. This, however, makes it really difficult to do the analysis of run-time properties.

Koala uses a global web-based repository to store interfaces and components.

3.7 ROBOCOP

The Robust Component Model for Consumer Electronic Products (ROBOCOP) [**?**] is a component model developed as an ITEA[5] project, which defines open, component-based architecture for the middleware layer in high-volume consumer electronic products.

The ROBOCOP architecture consists of different frameworks. The development framework defines a number of aspects relevant for the development and trading of components consisting of a number of different elements such as the stakeholder roles and their relations, the component model, and tooling. The run-time framework defines the execution environment for components.

The download framework enables dynamic upgrade and extension by allowing controlled downloading of components from a repository to a device, while the resource management framework provides mechanism for components to negotiate their resource needs. The resource management framework can be then used to specify for example timing properties if the implementation supports them.

Beyond the frameworks described, the ROBOCOP also defines components. A component in ROBOCOP is defined as a collection of models and relations between these models. This allows different concepts to be employed such as trading, composition and execution-time properties specification.

Functionality of components is encapsulated in services, each of which defines a set of provided and required interfaces and also third party bindings. The interfaces are considered as first-class entities.

At run-time, services are dynamically instantiated; these service instances are an analogy of objects. The interfaces, described using RIDL language, are represented by interface instances at run-time. They also supports interface inheritance. One object can be accessed through multiple interface instances by multiple clients.

The programming model strictly follows the Microsoft COM model; binary mappings to different programming languages can be defined. During run-time, the application is composed of executable components and Robocop Run-time Environment, which takes care of component creation.

3.8 THINK

THINK [**?**] is a C-implementation of the Fractal [**?**] component model targeted at embedded systems.

[5] http://www.itea2.org/

The original purpose of THINK was to simplify the development of kernels for embedded devices, but gradually it developed into a full-featured component system generally usable for embedded software development.

Because THINK is a Fractal implementation, each component provides the standard API for introspection, configuration and instantiation of component represented by different controllers.

Component functional code is written in the C language extended with annotations, called nuptC. Using THINK Compiler, different transformations and optimizations can be applied to the code.

Non-functional properties of components are managed using the extension of the THINK ADL, which allows specifying properties for any architectural entity. There are few already predefined properties and new properties may be added using THINK Compiler plug-in mechanism. These can be used to define additional views on component model, such as behavioral and reconfiguration view as described in [**?**].

Due to its origins, THINK offers Kortex library, which is a component library containing generic as well as hardware specific components that can be used to build and access operating systems services.

Part of the THINK project is also IDE based on Eclipse called thinkClipse. This offers a basic support for development of components using THINK component system.

3.9 SOFA HI

SOFA HI [**?**] is an extension of SOFA 2 component model [**?**] targeted at high-integrity real-time embedded systems.

The effort behind SOFA HI is to bring the knowledge of component systems gained from SOFA and SOFA 2 development into the real-time environment to speed up the development and lower the costs of high-integrity systems.

SOFA 2 is an advanced distributed component system employing hierarchically composed components. Moreover, SOFA 2 provides complete framework supporting all the stages of application development and deployment life-cycle. The component model itself is defined by means of its meta-model. The artifacts of the application component model are stored in repository, which manages their versioning and provides access to development tools and run-time environment.

SOFA 2 components are types specifying provided / required interfaces and observable behavior. Each component internally contains microcomponents which defines control part of component (in a similar way to Fractal / THINK).

As a profile of SOFA 2 targeted at real-time embedded systems, SOFA HI employs various restrictions on the component model in order to make it more predictable and lightweight. The component meta-model also supports specification of extra-functional properties such as component timing properties.

SOFA HI restricts dynamic architecture reconfigurations to mode switches at run-time only. As opposed to SOFA 2, SOFA HI disallows generation of connectors and controllers at run-time.

SOFA HI run-time as well as SOFA HI primitive components are implemented in the C programming language with the help of existing SOFA 2 tools and infrastructure. To ensure sufficient independence of the component implementation, SOFA HI defines an abstraction layer on top of the underlying OS and HW.

A wide range of existing development tools for SOFA 2 component model can be also used for developing SOFA HI application. They include Cushion as a command line development and management tool, SOFA 2 IDE for application modeling and development based on top of Eclipse as well as SOFA 2 MConsole for application deployment. There are also tools allowing formal analysis and verification of components.

3.10 BlueArX

BlueArX[**?**] is a component system developed and used by Bosch[6] intended for use in automotive domain especially in embedded devices.

The BlueArX focuses on the design time component model to support constrained domains considering various non-functional requirements while providing different views of a developed system.

The static view defines two types of components, an atomic component, which has an implementation, and a structural component. While atomic components represents leafs in the software architecture tree, the structural components represents nodes. Structural component may be composed of several atomic and/or structural components.

A component has interfaces, which are divided into two types – import and export interfaces – where import interface are required and export interfaces are provided by the component. A structural component can import or export a subset of interfaces from each atomic and structural component it is composed of. Connection between interfaces is implicit based on interface name. Communication between components is done using special type of variable called *message* where component specifies its message access properties in its interface description.

The dynamic view consists of component scheduling specification, which contains mapping of services to periodic or event-triggered tasks and the order of services inside these tasks. This information is used to generate a system schedule which is therefore used by the operating system called ERCOS. ERCOS is a specially designed operating system for automotive applications supporting cooperative and preemptive tasks.

The BlueArX component model also defines modes, which can be used to define either different scheduling or different control strategies of the real-time system. The modes are also referenced by the analytic interface, which allows to specify non-functional properties and semantic context information associated with components. The concept of modes is important because it allows to express real-time properties of processes much more precisely.

[6] `http://www.bosch.com/`

BlueArX also defines a simple development process composed of different steps and roles associated together with different activities of application development life-cycle.

The BlueArX component system also provides various development tools such as a tool, which can automatically generate annotations for AbsInt aiT[**?**] which allows to extract WCET of a component, XGen for semi-automatic extraction of mode candidates based on heuristic, and Artus-eMercury IDE built on top of Eclipse.

4 Evaluation

In this section, we evaluate the frameworks briefly described in the previous section. For better readability and comprehensibility, we divide the evaluation into several parts; each of them based on a different criterion defined in Section 2.

4.1 Component models and their features

In this section we evaluate the component models used in the frameworks (i.e. the criterion C1). Also we briefly consider models' features which are important from component-based development in general but not directly related to real-time and/or embedded systems.

Most of the considered frameworks offer a hierarchical component model, at least at design time. These are MyCCM-HI, PECOS, PROGRESS (on both ProSave and ProSys layer), AUTOSAR, Koala, and ROBOCOP. The Pin frameworks allows only a flat architectures without any hierarchy. Conversely THINK, SOFA HI, and BlueArX hierarchical components supports from design time to run-time.

Considering other advanced features of component models, SOFA HI offers first-class connectors with multiple communication styles. Koala and ROBOCOP also have connectors but they are used at design-time for modeling variability in component design. Most of the consider models provide a kind of formal specification or execution model of component behavior and its validation – some of them have full support for validation (Petri nets are used for MyCCM-HI and PECOS, behavioral protocols for SOFA HI), other have well defined execution model (UML state-charts for Pin, exact description in manual for ProCom). Fractal execution model has been described using Alloy specification language [**?**]; this technique could be also used for THINK as it is Fractal implementation.

In Table 1 a summary is presented.

4.2 Development support

In this section, we focus on existence of development tools and development methodologies (or at least development guidelines) for the selected frameworks (i.e. the criterion C2a and C2b).

Table 1. Evaluation of component models

	Hier. comp.	Connectors	Formal. behav./Ex. model
PECOS	design-time	yes	yes
MyCCM-HI	design-time	no	yes
PROGRESS	design-time	no	yes
AUTOSAR	fully	no	no
Pin	no	no	yes
Koala	design-time	yes	yes
ROBOCOP	design-time	yes	yes
THINK	fully	no	yes
SOFA HI	fully	yes	yes
BlueArX	fully	no	yes

To some extent, development tools exist for all of the selected frameworks. However, it is hard to evaluate them for frameworks like ROBOCOP and BlueArX since they are not publicly available. Since AUTOSAR is more a standard, so no tool support is required. For the other considered frameworks, tools are available, nevertheless, some of them are obsolete (Koala, Pin), incomplete (PECOS) or under development (PROGRESS, SOFA HI). Therefore, the ready-to-use tools are at this time available only for MyCCM-HI and THINK. Both of them transform the component descriptions into other technologies and then reuse the tool of these technologies. MyCCM-HI transforms the component description to AADL and use an existing compiler to create executable files. This approach is ideal from the point of view of reuse of existing tools, but it has also some significant disadvantages: at least, all the information about the structure is lost and the advantage of structure knowledge can not be used neither in any further phase of verification nor deployment process, nor at run-time. This could have important consequences, for example state explosion during verification process and restrictions in (at least theoretical) possibilities of dynamic reconfigurations at run-time.

A described development methodology is available for AUTOSAR and BlueArX.

Table 3 summarizes the subsection.

Table 2. Development support

	Devel. tools	Devel. methodology
PECOS	incomplete	no
MyCCM-HI	basic	no
PROGRESS	under development	no
AUTOSAR	no	yes
Pin	basic	no
Koala	basic	no
ROBOCOP	no	no
THINK	yes	no
SOFA HI	under development	no
BlueArX	yes	yes

4.3 Support of distributed and dynamic applications

Evaluation of support of distributed (the criterion C3a) and dynamic (the criterion C3b) applications is rather easy since the component model of a framework either supports it or not. Distributed applications are supported by MyCCM-HI, PROGRESS, AUTOSAR, and SOFA HI. The rest of the frameworks allows for non-distributed systems only.

By support of dynamic applications, we mean an ability to develop an application which can change its architecture at run-time, i.e. to add/remove components and/or bindings among them. The frameworks with such an ability are MyCCM-HI, ROBOCOP, SOFA HI, and BlueArX. ROBOCOP allows for partial dynamism only, since it offers dynamic upgrade and download of a component to a device. MyCCM-HI, SOFA HI, and BlueArX support dynamic applications via so called *modes* [**?**], i.e. an application can have several possible architectures and they can be switched among each other at some well-defined points.

Table 3 summarizes the subsection.

Table 3. Support of distributed and dynamic applications

	Distributed apps.	Dynamic apps.
PECOS	no	no
MyCCM-HI	yes	yes
PROGRESS	yes	no
AUTOSAR	yes	no
Pin	no	no
Koala	no	no
ROBOCOP	yes	partial
THINK	no	no
SOFA HI	yes	yes
BlueArX	no	yes

4.4 Status of the frameworks

The availability of documentation (the criterion C4a) and the overall status (the criterion C4b) are other important aspects.

At least a partial documentation exists for all of the considered frameworks. Nevertheless, in many cases such documentation consists only of several research papers (e.g. Koala, BlueArX, PROGRESS, SOFA HI, PECOS). On the other hand, specification of industrial systems like ROBOCOP and detailed documentation is not publicly available. AUTOSAR documentation is publicly available under AUTOSAR partnership license.

With respect to the status of the frameworks, unfortunately several of them seems currently to be abandoned. These are PECOS, Pin, Koala, and ROBOCOP, as there are news for them for rather longer time (results of ROBOCOP have been currently extended in the Space4U [**?**] and Trust4All [**?**] projects).

PROGRESS and SOFA HI are currently under heavy development, while MyCCM-HI and THINK are rather stable, maintained, and further developed. The AUTOSAR and BlueArX are used in the industry, however, again since they are not publicly available, any other conclusions are not possible.

Table 4 summarizes the section.

Table 4. Status of the frameworks

	Status
PECOS	abandoned
MyCCM-HI	ready-to-be-used
PROGRESS	under development
AUTOSAR	ready-to-be-used
Pin	not actively developed
Koala	abandoned
ROBOCOP	abandoned
THINK	ready-to-be-used
SOFA HI	under development
BlueArX	ready-to-be-used

4.5 Coupling with hardware and suitability for embedded systems

All of the considered frameworks are intended for embedded systems by allowing low-level coupling with hardware and by aiming at low memory footprint and other necessary features for embedded systems (i.e. criteria C5a and C5b). Coupling with hardware is typically provided by run-time environment, which provides an abstraction over a supported set of hardware platforms. For example the implementation of Pin framework relies on OS services of Windows NT and Windows CE; SOFA-HI, PECOS and Progress require other particular real-time operating systems. Similarly, AUTOSAR requires micro-controller abstraction as defined in its specification.

4.6 Support of real-time applications

The frameworks Koala, ROBOCOP, and THINK do not offer support for real-time application at all (i.e. they do not meet any criterion from C6a, C6b, and C6c). For AUTOSAR, the support of real-time properties is not clearly specified, however, since it is used for real-time applications in industry, it can be assumed that it supports them. The rest of the considered frameworks primarily target real-time systems and satisfy all three criteria (C6a, C6b, and C6c).

Table 5 summarizes the section.

Table 5. Support of real-time applications

	Attr. and analysis	Support for periodic. aperiodic tasks	Schedulability analysis
PECOS	yes	both	was planned
MyCCM-HI	yes	both	yes
PROGRESS	yes	both	is planned
AUTOSAR	yes	not specified	yes
Pin	yes	aperiodic	external
Koala	no	no	no
ROBOCOP	no	no	no
THINK	no	no	no
SOFA HI	yes	both	is planned
BlueArX	yes	both	yes

4.7 Summary

As apparent from the sections above, there is no clear winner framework suitable for everything. However, with some limitations, the best frameworks can be chosen.

For the automotive domain, the clear winners are AUTOSAR and BlueArX. The downside is that they are not publicly available, however, for the intended domain it is not an issue. Additionally, AUTOSAR can be considered as a little bit better since it is designed and developed by a consortium of companies while BlueArX is backed by a single company.

For non-automotive domain, from the short time point of view, the options are MyCCM-HI or THINK as both of them are ready-to-be used, with tool support, and publicly available. A downside of THINK is that it does not support real-time properties, however there is also a new project MIND [**?**] that has been started recently. Its attempt is to build a new framework for industrial use based on THINK, which will support also RT properties.

From the long-term perspective, most promising technologies seems to be the SOFA-HI and PROGRESS frameworks as they target a clear model-driven approach of design and development. Moreover, SOFA-HI builds on existing SOFA 2 development environment, which comprises a large tool-set including graphical Eclipse-based IDE, graphical deployment, run-time console, shared component repository, and various analysis tools.

5 Conclusion

In this paper, we have overviewed a number of state-of-the-art component frameworks suitable for building real-time embedded systems. Our aim was to provide some orientation among them, as they significantly differ in offered features and maturity. To provide common criteria, we have consulted existing literature and identified the features which are important for building real-time embedded

systems using components. We have evaluated the discussed component frameworks and compared them with respect to the defined criteria. The results of the evaluation show that there is no single universal "winner", therefore we have formulated recommendations based on the intended use of a component framework. These recommendations therefore provide quite a useful guide in selection of a suitable component technology according to the specific requirements of each application. The presented results also open space for further research in this area of software research.