Component-Oriented Engineering

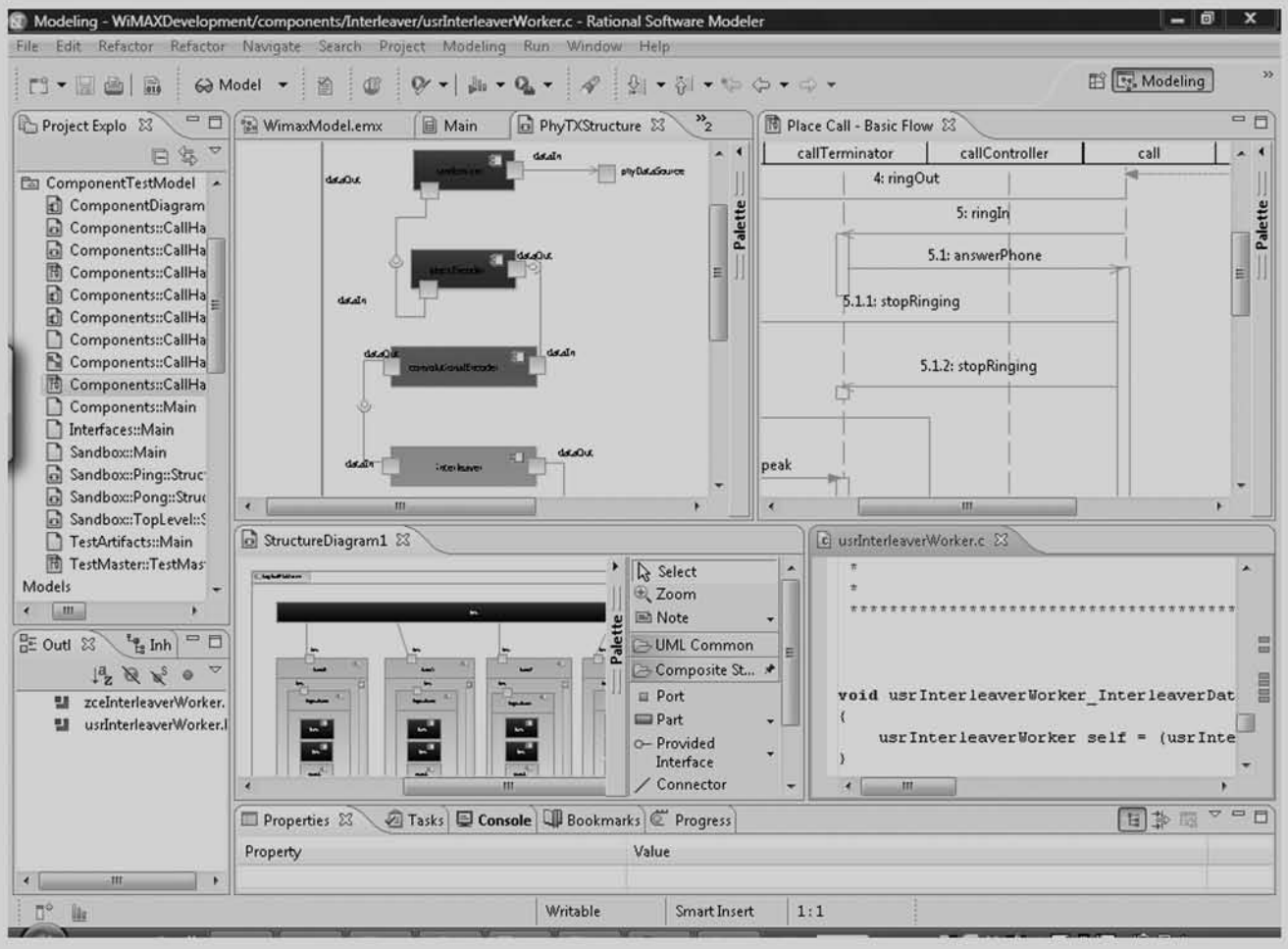
... the dawn of a new era in embedded software development productivity

Francis Bordeleau and Ross MacLeod

Zeligsoft

May 2008

zeligsoft


Modeling - WiMAXDevelopment/components/Interleaver/usrInterleaverWorker.c - Rational Software Modeler
File Edit Refactor Refactor Navigate Search Project Modeling Run Window Help
Model
Modeling
Project Explo
WimaxModel.emx
Main
PhyTXStructure
Place Call - Basic Flow
callTerminator
callController
call
4: ringOut
5: ringIn
5.1: answerPhone
5.1.1: stopRinging
5.1.2: stopRinging
peak
ComponentTestModel
ComponentDiagram
Components::Main
Interfaces::Main
Sandbox::Main
TestArtifacts::Main
Models
StructureDiagram1
usrInterleaverWorker.c
Select
Zoom
Note
UML Common
Composite St...
Port
Part
Provided Interface
Connector
Palette
void usrInterleaverWorker_InterleaverDat
usrInterleaverWorker self = (usrInte
Properties
Tasks
Console
Bookmarks
Progress
Property
Value
Writable
Smart Insert
1 : 1

Component-Oriented Engineering

... the dawn of a new era in embedded software development productivity

Francis Bordeleau and Ross MacLeod

Improving embedded software development productivity is rapidly becoming the overwhelming mantra of product companies across a wide range of industries. The increasing importance of software to the value of products in such diverse industries as automotive and consumer electronics has lead to a huge growth in embedded software, as well as a corresponding growth in delivery and quality challenges. Recognizing the need for new software development approaches to successfully address these growing challenges, industry is increasingly looking to modeling and software component technologies to help them succeed. Industry specific component frameworks such as the Software Communications Architecture (SCA) for software defined radios, and AUTOSAR (AUTomotive Open System ARchitecture) for the automotive industry, are being promoted as a means of moving towards an integration model of development in order to improve productivity and quality. Within this context, applications may be largely assembled from pre-existing components, rather than being developed each time from first principles. Model Driven Development (MDD) and Component-Based Development (CBD) have promised such a paradigm for some time, but have not fully delivered on that promise. This paper cites some reasons for the gap within the embedded software domain and proposes a new approach to help organizations realize the promise.

1 BACKGROUND

Component-Based Development (CBD) is now well established in the IT industry. Leading component-based technologies include J2EE, Microsoft .NET and IBM Websphere. A component is an encapsulated unit of functionality with a well-defined interface that allows it to connect to other components, and be independently deployed. Component-based applications are defined by assembling components.

The main benefits associated with component-based technologies include: reduced system development time and cost, enhanced quality, and reduced system evolution and maintenance cost. Over the past decade as standard component-based specifications have been developed, the importance of CBD has grown rapidly in the embedded system industry. Examples include both industry independent specifications such as the CORBA Component Model (CCM) and OSGi, as well as domain-specific specifications such as the AUTomotive Open System ARchitecture (AUTOSAR) in the automotive industry and the Software Communications Architecture (SCA) in the Software Defined Radio (SDR) industry.

Other industry segments like telecommunications and consumer electronics have been using component-based technologies without adopting standard specifications.

Coincidently, Model Driven Development (MDD)[1] has gained popularity with pioneering efforts from companies such as Rational, ObjecTime, Telelogic and I-Logix. The term MDD refers to a variety of development approaches that are centered on the use of software models as a primary form of expression.

1. The OMG has coined the term Model Driven Architecture (MDA) to refer to a variety of model-based development techniques built around UML and model transformation techniques. MDA introduces the concepts of Platform Independent Model (PIM) and Platform Specific Model (PSM). The transition between PIM and PSM is achieved using well-defined reusable model transformations.

Models can be analyzed and validated, and code may in turn be generated from the models, which range in completeness from system skeletons to fully deployable products.

Industry has, however, been slow to embrace these technologies despite their promise to significantly improve developer productivity. Concerns with key issues, like scalability, performance and runtime overhead, and imposed programming models, has led those with demanding system performance requirements to opt for the status quo and to continue to use custom tailored and handcrafted solutions.

Zeligsoft has been focused on CBD and MDD since its founding in 2002. The company is led by a group of industry veterans who have been leaders in these fields over the past couple of decades, and collectively they have logged over 100 years of experience. Zeligsoft has witnessed the typical challenges with these approaches and has defined a methodology — referred to as Component-Oriented Engineering — and a toolset to address these shortcomings and unleash the full power of model driven and component-based development within the embedded software space.

2 COMPONENT-ORIENTED ENGINEERING (COE)

Component-Oriented Engineering is a software development methodology focused on addressing the challenges of complex embedded systems characterized by distributed applications and heterogeneous multiprocessor platforms[2]. COE derives its inspiration and key attributes from three main sources:

- **Component-Based Development (CBD)** — COE has been created to support the development of component-based software through the entire development process. COE directly addresses development issues associated with component definition, application design, application deployment on platforms, and testing at different levels.
- **Model Driven Development (MDD)** — COE is an advanced MDD methodology for embedded systems. It is centered on the use of models to specify the different system aspects. COE provides capabilities to analyze and validate system properties early and often in the development process, and provides advanced generation capabilities that automatically generate optimized code, XML descriptor files, documentation and other types of models.
- **Agile Software Development (ASD)** — COE builds on the core principles of ASD to provide a user-centric, lightweight methodology that enables developing software in a highly iterative fashion, driven by a test-based approach to ensure quality and reduce risks usually associated with conventional development methodologies.

COE is agnostic with respect to the approach used to specify behavior models, and is open to any behavior model that respects the defined protocols and platform constraints. It thus enables greater reuse of existing software assets.

Additionally, because of the key importance of QoS in embedded systems, COE places a strong emphasis on this aspect — in particular real-time and performance properties. As such, support for QoS analysis and validation form an integral part of the methodology and are used throughout the process.

Component-Oriented Engineering is unique in terms of its:

1. Explicit support for a platform concept
2. Scenario-driven development approach
3. Use of domain specializations to support customization

2. Platforms are heterogeneous with respect to processors (single-core, multicore, GPP, DSP, FPGA), communication mechanisms, RTOS, middleware, etc.

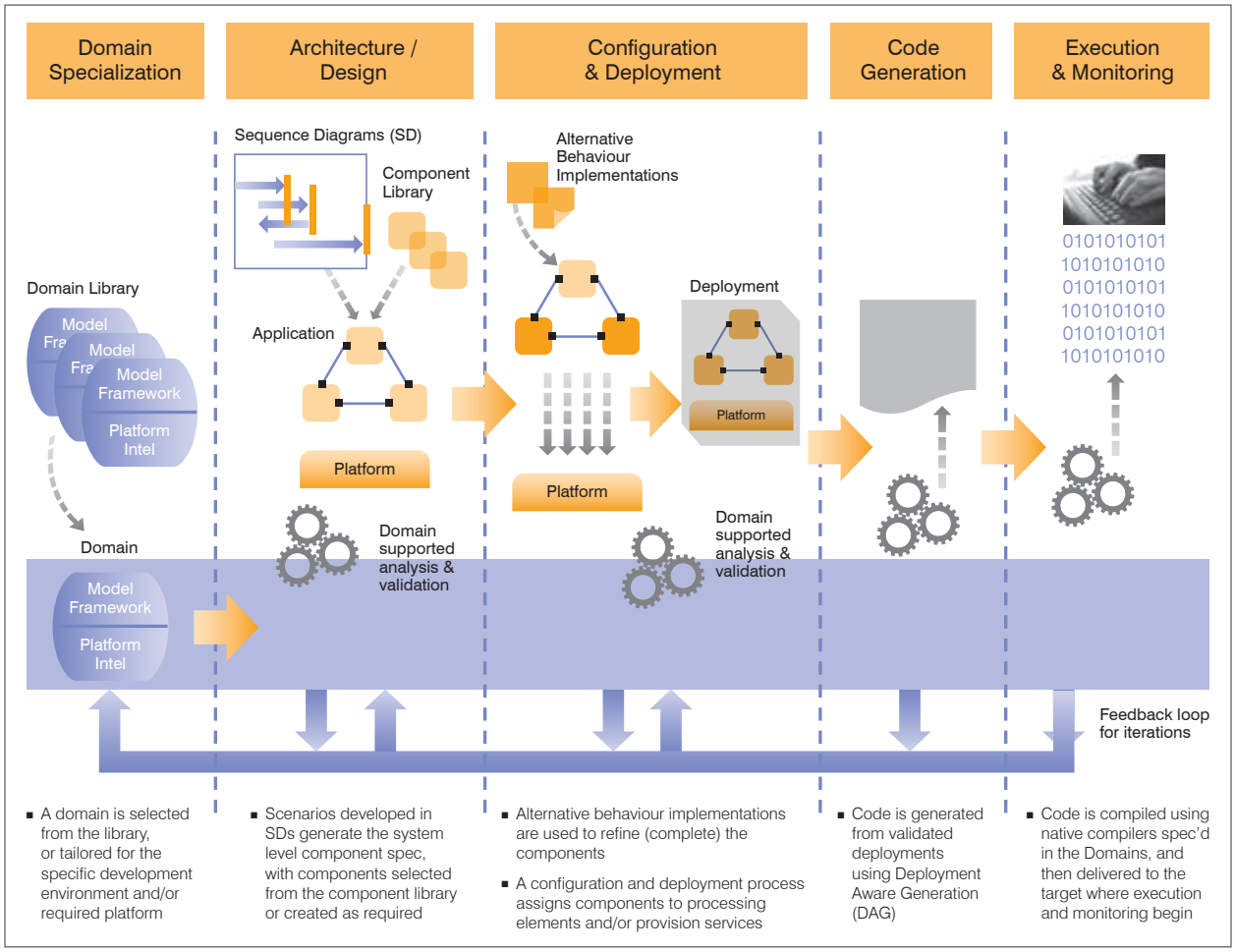


Figure 1: Component-Oriented Methodology overview

2.1 Platform Concept

Component-Oriented Engineering is unique in its formal recognition of platforms as first class objects within the methodology. Other methodologies may implicitly acknowledge platforms through the use of concepts, such as Platform Independent Models (PIM) in MDA, but only COE requires that developers recognize a platform entity and manage relationships between applications and platforms via another important concept — that of a **deployment**. Within COE, the deployment process is an integral part of the overall development process and is no longer simply a post-development activity[3]. The integration of application, platform and deployment enables advanced model analysis and validation. In particular, this facilitates rapid "what-if" analysis of alternative target configurations. In this way the developer can easily tune the application early, rather than suffering the costly implications of performing these changes late in the product life cycle. These analysis and validation capabilities are critical to ensuring a degree of correctness that is required in mission critical applications.

Moreover, detailed platform and deployment information is necessary to drive high-performance automated model transformations (such as code generation) that are key to meeting the performance requirements demanded in many embedded applications, while still achieving the productivity gains expected from higher level domain representations.

3. In reality most methodologies require the developer to make deployment related decisions early in the development cycle. These decisions are reflected in the architecture/design that then make it costly to evolve implementations to accommodate new platforms.

Application

Applications in COE are created by assembling a set of components and providing connections between compatible ports (formal interfaces) on adjacent components. Using MDD techniques, individual components and the overall application model can be validated for correctness at any point in the development process.

Platform

Complex embedded systems are built on heterogeneous platforms that are composed of multiple layers, each layer addressing a different aspect of the overall system. Typical platform layers include a hardware layer composed of different types of processors (GPP, DSP, FPGA), memory buses and interconnections, and the RTOS layer and middleware layers[4] that provide services to application components. Similarly to applications, platform layers can be individually defined in a component-based manner. The component model can also be used to analyze and validate the configuration and generate code and configuration files.

Deployment

One of the core innovations — and benefits — introduced by COE is the ability to graphically manage deployments as part of the development process. This supports a unique exploratory approach to refining target/deployment choices in order to optimize the product according to specified engineering goals. A deployment is an allocation of application components to platform (software or hardware) level components. Similarly to applications and platforms, deployments can be modeled and validated. Resources required by the application components can be assessed against those provided by the platform components and a determination made about the correctness of the deployment. Further, the deployment is the first design element from which code and other related artifacts can be generated for the application and optimized for the allocation decisions.

Domain

Customization of the COE development enviroment is provided by domains, which tailor the COE framework for platform intelligence and component model frameworks. Platform intelligence contains rules and parameters to support static analysis and validation that is essential to achieving the QoS objectives of many embedded applications. Component model frameworks further refine the modeling rules and basic constructs. Domains also include model transformation engines that correspond to the selected component framework. Some examples of component frameworks are the Software Communications Architecture (SCA) for software defined radios, AUTOSAR for automotive and the CORBA Component Model (CCM).

4. A complex embedded system typically uses many different middleware layers.

2.2 Scenario Driven

COE's scenario-driven approach plays a central role in facilitating a more productive and "correctness-focused" approach to development. Two distinct types of scenarios are supported:

- Test scenarios
- Deployment and configuration scenarios

Test Scenarios: Test scenarios supported by sequence diagrams may be used to build the specification for components as well as their interactions with other components. These may then be used to drive the creation of automated test suites which may also be used to provide automated regression suites, component test suites and/or system test suites. As in Agile Software Development (ASD) this approach to early specification and use of test cases tends to be more effective and less expensive than traditional test approaches.

Deployment and configuration scenarios: Perhaps the most novel aspect of COE — the platform deployment process — enables the developer to rapidly explore the implications of different configurations early in the life cycle. This helps avoid costly mistakes and save weeks of development and re-design effort. In a world of increasingly distributed architectures (e.g. multicore processors), exploring alternative deployment scenarios in this way offers a powerful mechanism for dealing with the added design complexity associated with these complex distributed systems. Moreover, this process allows the designer to quickly determine the performance implications of different allocations.

In addition to the usual configuration options, COE also allows different behavior models/implementations to be plugged into a component over the product's development life cycle — as long as the behavior complies with the component interfaces and constraints. The software engineer may start with a high-level prototype (or stub) to be followed by a more complete and/or efficient implementation later in the cycle.

Deployment and configuration scenarios provide the developer with powerful mechanisms for early and tangible feedback on design effectiveness. Together with test scenarios, this approach supports the iterative and "test early; test often" principle of Agile Software Development.

2.3 Domain Specialization

One challenge for any development methodology and tool is that in the real world there will always be numerous component model frameworks as well as company specific variations. How can one develop a methodology and tool suite to practically support this variety? This challenge is further exacerbated by the multitude of platforms that must be understood by the tool in order to produce efficient implementations that are required by the embedded market.

In order to tackle this challenge, COE incorporates the concept of domains, where a domain is an entity that encapsulates specific platform knowledge, component framework information and associated transformation engines. Domains can then be specialized to accommodate a specific target platform as well as a component framework such as AUTOSAR, SCA, or CCM. Moreover, domains may be further specialized to accommodate any industry standard component framework.

Domains are used to complete the COE framework in order to provide a modeling and design solution.

3 PROCESS INTEGRATION AND LIFE CYCLE SUPPORT

To maximize productivity improvement and facilitate adoption of the methodology, COE was designed to integrate smoothly with existing development processes. COE incrementally adds a component-driven aspect to the existing system/software development methodologies and processes. Moreover, it is agnostic with respect to the approach used to implement the components (code). This means that component developers can choose to implement

components using an MDD approach or a code-centric approach, and they can select different programming languages, like C, C++ or Java, depending of the type of component.

From a life cycle perspective, COE focuses on the component-based aspect of the entire product development life cycle, including component definition, system integration, and testing from individual components to integrated systems. It integrates with existing methods and tools to provide support for the other development aspects such as requirements management, component implementation and version control.

4 COE ADDRESSES TYPICAL MD/CBD SHORTCOMINGS

Three commonly cited reasons for not adopting MDD or CBD technologies, despite their substantial promise of improved productivity, are:

1. Scalability issues
2. Inefficient code (implementations)
3. Awkward and/or restrictive programming models that compromise productivity

Component-Oriented Engineering embraces the key elements of MDD and CBD and further extends them with novel capabilities that significantly leverage the productivity benefits of these approaches. In addition, COE and the Zeligsoft toolset effectively address these three concerns most often raised over adopting any of these methods.

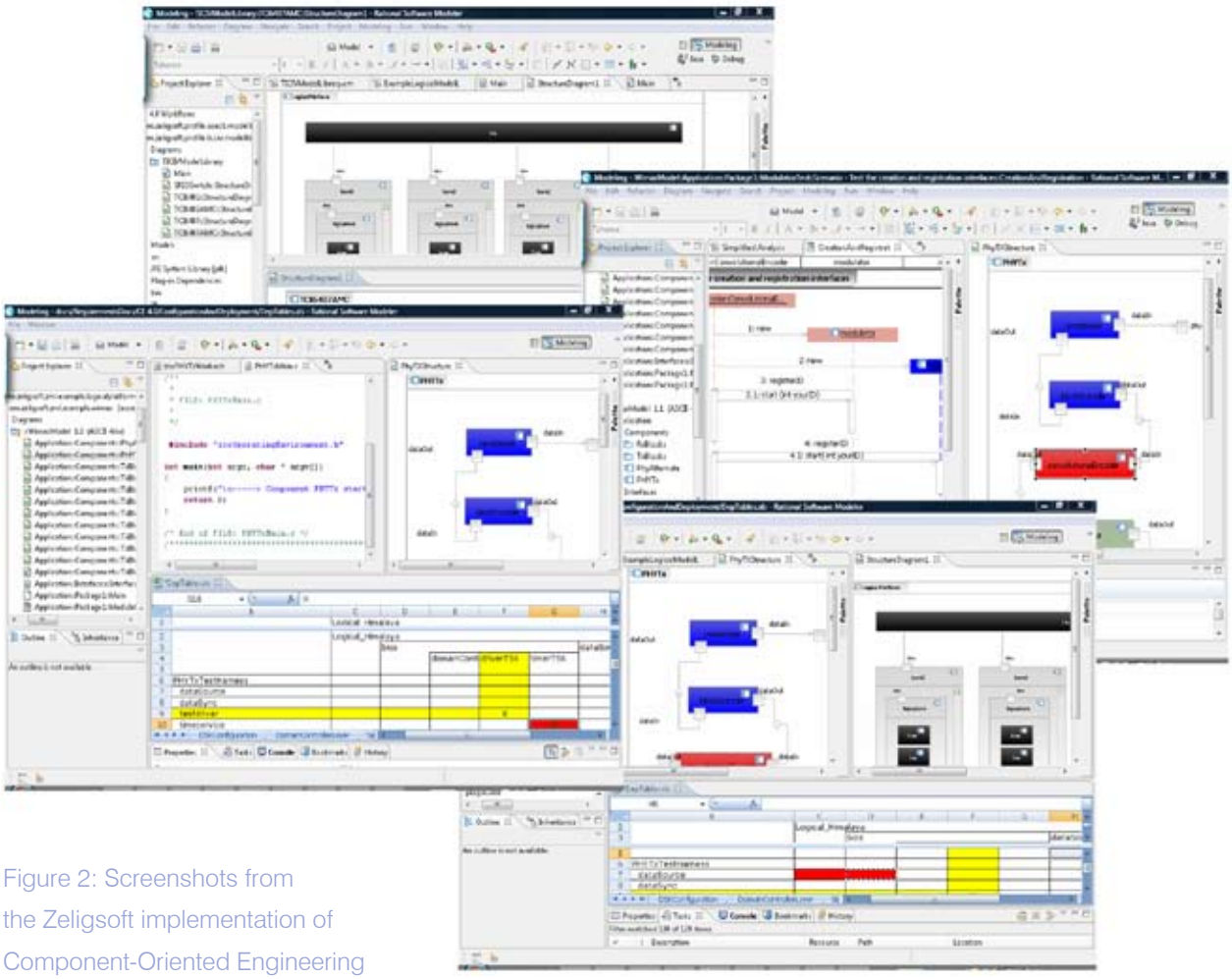


Figure 2: Screenshots from the Zeligsoft implementation of Component-Oriented Engineering

Scalability: Veterans of the MDD industry have developed the Zeligsoft toolset. They have designed the Zeligsoft development environment from day one to ensure scalability. This is important because it is very difficult to re-engineer a tool suite for performance. Innovative UI models have been adopted to help effectively manage the browsing of very large designs.

Inefficient code: Platform intelligence that is embedded in the domain specializations, combined with the COE deployment process, provide the necessary information from which very efficient code can be produced. The Zeligsoft implementation of COE incorporates a patented, Deployment-Aware Generation (DAG) capability that has proven its ability to generate very efficient code in demanding customer environments.

Restrictive programming models: COE is open to any programming model and tool that the developer wishes to use to specify component behavior. In fact, it is even possible to define and implement a proprietary programming model using domain specializations. Moreover, COE can be easily incorporated into virtually any life cycle development methodology that is in use in the organization, thus helping to improve its acceptance in organizations with well-established methodologies and processes.

5 THE FUTURE OF COE

Over 20 years ago, object-oriented programming offered the promise of greater development efficiency via increased software reuse. Programs would be composed from a collection of reusable software objects and classes. Decoupling of data from procedures, along with proper encapsulation, intended to achieve massive reuse and "the industrialization of software development". These changes alone, however, proved insufficient to launch the world of massive reuse that many had envisioned[5]. COE progresses the state of the art, by focusing on the structural aspect of software that is fundamental to achieving effective reuse.

It is essentially agnostic with respect to the type of behavior modeling that is adopted, and it treats platforms as first class entities within its modeling framework in order to ensure the performance and correctness that is required in embedded systems. COE leverages the abstract modeling benefits of MDD and formal model transformations in order to support static analysis, validation and automated generation of tests and code.

With support from the Object Management Group (OMG) and some very influential industrial partners, acceptance of Model Driven Development has increased tremendously over the past several years. MDD as well as Component-Based Development have taken root across the entire embedded software development industry as evidenced, in part, by the increase of industry standard component frameworks and modeling languages such as AUTOSAR (automobile industry), SCA (software defined radios), MARTE (real-time UML) and others.

COE has the potential to dramatically transform embedded software development from an exercise of creation from first principles, to one of assembling solutions from a set of reusable components, inventing new components to fill specific needs. The potential to more effectively leverage the creative powers of talented developers, reduce development costs, reduce product defects and gain time-to-market is truly exciting.

5. Brad J. Cox, November 1990, *Planning the Software Industrial Revolution*

Information about Zeligsoft can be found at **www.zeligsoft.com** or contact **sales@zeligsoft.com**.

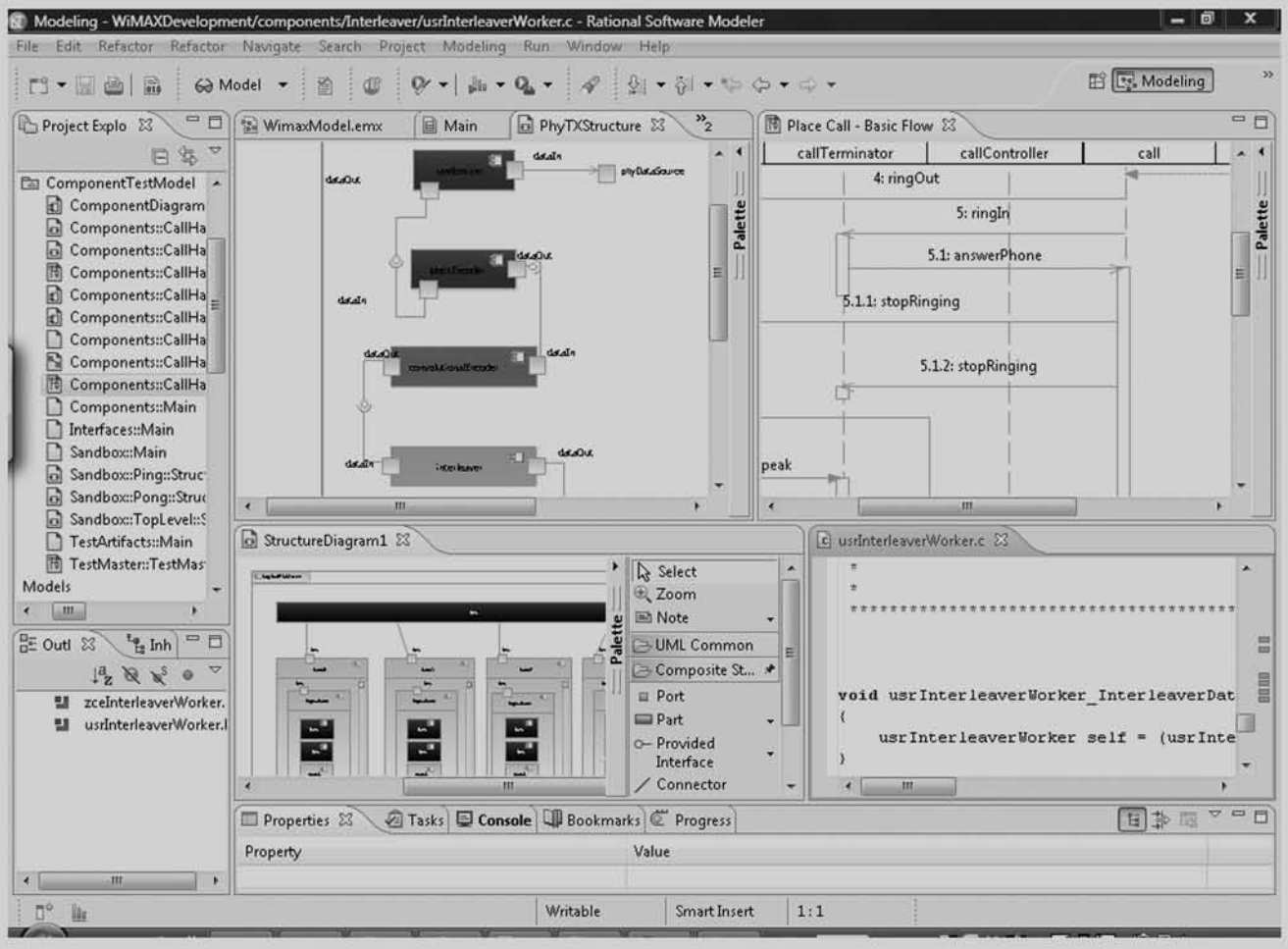
REFERENCES

1. Crnkovic, I., Larsson, S., and Chaudron, M., "Component-based Development Process and Component Lifecycle". *Journal of Computing and Information Technology* 13 (4), P: 321-327 (November 2005).

2. Schmidt, D.C.. "Model-Driven Engineering". *IEEE Computer* 39 (2), P: 25-31 (February 2006).

3. Agile Alliance. "Agile Software Development Manifesto." 13 Feb. 2001 www.agilemanifesto.org.

zeligsoft


Modeling - WiMAXDevelopment/components/Interleaver/usrInterleaverWorker.c - Rational Software Modeler
File Edit Refactor Refactor Navigate Search Project Modeling Run Window Help
Model
Modeling
Project Explo
WimaxModel.emx
Main
PhyTXStructure
Place Call - Basic Flow
callTerminator
callController
call
4: ringOut
5: ringIn
5.1: answerPhone
5.1.1: stopRinging
5.1.2: stopRinging
peak
ComponentTestModel
ComponentDiagram
Components::Main
Interfaces::Main
Sandbox::Main
TestArtifacts::Main
Models
StructureDiagram1
usrInterleaverWorker.c
Select
Zoom
Note
UML Common
Composite St...
Port
Part
Provided Interface
Connector
Palette
void usrInterleaverWorker_InterleaverDat
usrInterleaverWorker self = (usrInte
Properties
Tasks
Console
Bookmarks
Progress
Property
Value
Writable
Smart Insert
1 : 1

Contact Information

Website: www.zeligsoft.com

Email: info@zeligsoft.com

Toll-free (North America): 1-800-ZELIGSW (1-800-935-4479)

Direct dial: +1 819-684-9639