KobrA-Team

Affiliation

University Mannheim

Team Leader

Daniel Brenner, dbrenner@uni-mannheim.de

Team Members

Colin Atkinson, Matthias Gutheil, Dietmar Stoll, Oliver Hummel, Giovanni Falcone, Philipp Bostan

Component Model

We intend to use the KobrA method which supports a model-driven, UML-based representation of components. This enables the benefits of component-based development to be realized throughout the software life-cycle and allows the reusability of components to be significantly enhanced. It provides techniques to:

- develop a model-driven architecture in which the key features of a system are described independently of specific implementation platforms
- systematically reuse COTS components in new applications
- improve the quality of components and the systems assembled from them

---

SOFA-Team

Affiliation

Charles University, Prague, Czech Rep.

Team Leader

Frantisek Plasil, plasil@nenya.ms.mff.cuni.cz

Team Members

Frantisek Plasil, Jiri Adamek, Jan Kofron, Petr Hnetynka, Pavel Jezek, Tomas Bures, Pavel Parizek

Component Model

We intend to use the SOFA 2.0 component platform [1,2]. SOFA 2.0 uses a hierarchical component model formally defined using its meta-model [1].

DisCComp-Team

Affiliation

TU Kaiserslautern, AG Softwarearchitektur

Team Leader

Sebastian Herold, herold@informatik.uni-kl.de

Team Members

Sebastian Herold, Holger Klus, Andreas Rausch, Yannick Welsch

Component Model

DisCComp - A Formal Model for Distributed Concurrent Components

Our formal model is based on set-theoretic formalizations of distributed concurrent systems. It allows modelling of dynamically changing structures, a shared global state and asynchronous message communication as well as synchronous and concurrent message calls.

Ain Shams University Team

Affiliation

Ain Shams University, Cairo, Egypt

Team Leader

Islam El-Maddah, islam_elmaddah@yahoo.co.uk

Team Members

Islam A. M. El-Maddah

Component Model

The model combines formal and in formal description

Formal parts

1. A set of variables describing the internal state of the component
2. A set of agents (software/hardware) intended to control these variables
3. A hierarchy of and-or tree to represent component basic responsibilities. Each node of this tree has formal pre- and post- conditions based on the variables

Each variable and agent and and-or tree node has formal and informal description. Different analysis and checks can be applied on each component in order to ensure the basic responsibilites will be correctly fulfilled. These checks include: consistency, completeness, reachability, symbolic execusion.

TU/e Eindhoven-Team

Affiliation

TU/e – Technische Universiteit Eindhoven

Team Leader

Egor Bondarev, e.bondarev@tue.nl

Team Members

Michel Chaudron

Component Model

Our compositional analysis technique introduces (a) composable software and hardware component models representing abstract specification of the component behaviour and corresponding resources, (b) operational semantics enabling composition of the models into an executable system model, and (c) simulation-based analysis of the obtained executable model resulting in predicted performance attributes. Example attributes are response time, throughput, utilization of processors, memory and communication lines. Special attention is paid to modeling of both passive and active components exploiting synchronous method invocation and asynchronous message passing interaction.
The component models itself are: behaviour, resource and process models.
For software components, our technique introduces three types of models: resource, behaviour and process models. Typical models for hardware IP blocks are memory, communication and processor performance models.
The resource model specifies resource requirements (e.g. number of claimed CPU instructions) of each accessible individual operation of a component. The resource requirements are obtained by profiling of each individual component on a reference processor.
The behaviour model describes the operation's underlying calls to operations of other interfaces and various synchronization constraints (mutexing, critical sections, etc).
The process model specifies the processes activated and running within an active component. For every process, the model describes process creation and release conditions, periodicity and a sequence of underlying operation calls to other interfaces. The data for the behaviour and process models is obtained by the source-code analysis.

KLAPER-Team

Affiliation

Politecnico di Milano

Team Leader

Raffaela Mirandola, mirandola@elet.polimi.it

Team Members

Vincenzo Grassi, Antonino Sabetta: Universita' Roma TorVergata, Italy
Moreno Marzolla: INFN Padova, Italy
Antinisca Di Marco, Vittorio Cortellessa: Universita' de L'Aquila

Component Model

KLAPER is a modeling language aiming to capture the relevant information for the analysis of non-functional attributes of component-based systems, with a focus on performance and reliability. We point out that KLAPER is not actually a "component model", but it is rather meant as an itermediate model to be used in a transformation path from design-oriented models to analysis-oriented models. For this purpose, its goal is to help in distilling relevant information for analysis purposes from the original design-oriented model.
The idea underlying KLAPER is to provide support for the modeling of a system as an assembly of interacting resources, where a resource denotes an entity that may offer services (and possibly requires others). Each offered service is characterized by a list of formal parameters that can be instantiated with actual values by other resources requiring that service. The formal parameters and their corresponding actual parameters are meant to represent a suitable analysis-oriented abstraction of the “real” service parameters. To support performance or reliability analysis, each KLAPER service provides information about its execution time or failure characteristics. A service behavior is characterized by a set of used services and consists of a set of execution steps. Steps can model either an internal activity, i.e. an activity performed by the same resource offering the service, or a set of service calls, i.e. a set of requests for services offered by external resources.

CompoNets

Affiliation

LIIHS - IRIT - Université Toulouse 1

Team Leader

Rémi Bastide, Remi.Bastide@irit.fr

Team Members

Eric Barboni, Ph.D. student in Toulouse

Component Model

The proposed specification technique (called CompoNets as a contraction of 'Component' and 'Petri net') is based on a component model that complies with the current practice of software engineering, in particular inspired by the CORBA Component Model. The originality of the approach is to complement this component model with a behavioural specification notation based on high-level Petri nets. This notation is used to specify the internal behaviour of the components, that can exhibit internal concurrency and time- or event-based behaviour. In our approach, the inter-components communication is also specified in terms of Petri nets. Unicast synchronous communications can be modelled, as well as multicast, and publish-subscribe communication styles. The approach is supported by a software tool called PetShop, which supports the editing and execution of the models, as well as several forms of verification based on Petri nets theory.

SPEEDS-Team

Affiliation

OFFIS e. V. (Oldenburger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge und –Systeme)

Team Leader

Eike Thaden, Eike.Thaden@offis.de

Team Members

Qin Ma, Alexander Metzner, Eike Thaden

Component Model

The SPEEDS project is a concerted effort to define the new generation of end-to-end methodologies, processes and supporting tools for safety-critical embedded system design.
The bases of the SPEEDS project are so-called Heterogeneous Rich Components (HRC). HRC components describe the structure and the behaviour of functional entities of embedded systems. The structural specification part of a component uses blocks and ports as defined in SYSML connectors for connecting components and for delegating method calls, signals, etc. to subcomponents. The behavioural specification part of a component consists of viewpoints, which focus on different aspects of the component, for example functional behaviour, safety or realtime. Each viewpoint consists of a set of assertions which combine a set of assumptions with a set of promises. The promises are guaranteed only if the assumptions are fulfilled. The language used to formulate assumptions and promises is based on so-called extended state machines, which allow the specification of discrete and continuous behaviour. Extended state machines are formally defined in the semantics foundation of the HRC meta-model. Structural and behavioural specifications will be used to check compatibility of HRC components. For example two ports can be connected only if their protocol specifications consisting of compatible assumptions and promises.

Abstraction is used heavily to reduce the complexity of system specifications. Between HRC components there can exist a refine relationship, e. g. while the first component may talk only about external visible elements without defining subcomponents, a refined one can also define

subcomponents and delegation. In embedded systems, beside a functional layer, where components communicate via abstract message passing, also more concrete layers like the ECU layer or the hardware layer are of interest. In the ECU layer, functional units are seen as tasks, deployed to processors and communication is done using messages passed on busses. On the hardware layer also internal specifications of processors and busses are visible. Components of higher layers have more detailed representations in lower layers and are "connected" to them with a mapping relation involving assumptions and promises.

Palladio-Team

Affiliation

Universität Karlsruhe (TH)

Team Leader

Ralf Reussner, reussner@ipd.uka.de

Team Members

Ralf Reussner, Steffen Becker, Heiko Koziolek, Klaus Krogmann, Michael Kuperberg, Thomas Goldschmidt, Jens Happe

Component Model

The Palladio Component Model (http://sdqweb.ipd.uka.de/wiki/Palladio_Component_Model). The Palladio Component Model (PCM) is a metamodel for a domain specific language to describe systems build using components. It is designed to support a component based software development process. Its special capabilities allow the system designer to do early design time predictions of Quality of Service (QoS) attributes. The PCM is implemented using the Eclipse Modeling Framework (EMF).

ADL-Team

Affiliation

University of California, San Diego

Team Leader

Ingolf Krueger, ikrueger@cs.ucsd.edu

Team Members

Massimiliano Menarini, mmenarini@ucsd.edu
Vina Ermagan, vermagan@cs.ucsd.edu

Component Model

The model we intend to use is a part of our Service-Oriented specification ADL. The model is based on message passing between logical entities in the system. Our Service ADL (SADL) uses two layers: a logical one that captures the interaction pattern among logical entities (Roles), and a deployment layer that assign roles to physical components and logical communication paths to physical connections.
The interaction patterns are described using a dialect of message sequence charts (MSC) that contains various operators for service composition. The deployment part of the system is described by a component diagram. Our SADL uses data flows diagrams, state charts, and calls to external functions (written in some programming language suitable for the deployment platform) to specify computations local to each Role, aka local actions, and triggered by the interaction patterns captured in the MSCs.
Our SADL captures the concept of failure in a failure hypothesis model. This supports catering to different types of failures both of logical and deployment elements of our model.
The formal semantics of our SADL is defined for the logical models over the set of infinite streams capturing: the messages sent trough logical communication channels and the internal state changes of each Role. A mapping between logical and deployment models establishes a refinement relation between the logical specification and the implementation.

Grid Component Model Team

Affiliation

INRIA Sophia-Antipolis

Team Leader

Eric Madelaine, eric.madelaine@sophia.inria.fr

Team Members

Eric Madelaine, Ludovic Henrio, Antonio Cansado, Marcela Rivera, Emil Salageanu

Component Model

We work on the Grid Component Model that is a hierarchical, distributed, component model for computationnal grids, defined in the CoreGrid NoE.
This is a component model inspired and extended from a distributed implementation of the Fractal component model based on the ProActive library.

Our stress is on specification and validation of behavioural models of the components, based a model called parameterized Networks of synchronized transition systems (Forte'04). We have defined model-generation methods and a set of supporting tools, for checking safety properties of components and validating their correct composition (Spin'05, Facs'05, Facs'06).
This work at the formal model level is backed-up by the ProActive implementation of the component model, that is a wide-spread technology for developing component-based applications (see e.g. the Grid@works plugtest series, http://www.etsi.org/plugtests/Upcoming/GRID2006/GRID2006.html).

rCOS-Team

Affiliation

UNU-IIST

Team Leader

Zhiming Liu, lzm@iist.unu.edu

Team Members

Xin Chen, Dang Van Hung, Xiaoshan Li, Zhiming Liu, Vladimir Mencl, Joseph Okika, Volker Stolz, Anders P. Ravn, Lu Yang, Naijun Zhan

Component Model

rCOS – A relational Calculus for Object Systems

Box-Model-Team

Affiliation

TU Kaiserslautern

Team Leader

Arnd Poetzsch-Heffter, poetzsch@informatik.uni-kl.de

Team Members

Jean-Marie Gaillourdet, Kathrin Geilmann, Arnd Poetzsch-Heffter, Markus Reitz, Jan Schäfer

Component Model

We intend to use our box-model (see: Formal Methods for Components and Objects, FMCO 2005. LNCS, volume 4111, Springer, p. 313--341, 2006). The box-model is a component model based on object-oriented programming abstractions. The box-model is still under development. It is meant to fill the gap between high-level descriptions and implementations of component-based systems.

A box is a runtime entity with state and a semantics-based encapsulation boundary. The implementation of a box consists of a number of classes and interfaces. A box is created by instantiating one distinguished class of the implementation. A box can encapsulate other boxes, so-called inner boxes. The result is a hierarchy of boxes at runtime.

A client of a box B can hold references to the objects at the boundary of B. Invocations on boundary objects will be synchronized by the box, that is, from the outside a box essentially exhibits a sequential behavior whereas execution inside a box is in general concurrent.

Box specifications describe the interface behavior in terms of abstract state changes and the callbacks caused by method calls. The realization or architecture of a box B is described in terms of its inner boxes B_i, the connections and interactions between the B_i's, and the expression of B's abstract state in terms of its concrete state and the abstract states of the B_i's.

SysML-Team

Affiliation

LIUPPA - Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour

Team Leader

Jean-Michel Bruel, bruel@univ-pau.fr

Team Members

Belloir Nicolas, Bruel Jean-Michel, Dalmau Marc, Jobard Bruno, Laplace Sophie, Luthon Franck, Pham Congduc, Roose Philippe

Component Model

We intend to use SysML improved with a profil dedicated to composition and to QoS handeling.

Fractal-Team

Affiliation

Charles University in Prague

Team Leader

Frantisek Plasil, plasil@nenya.ms.mff.cuni.cz

Team Members

Frantisek Plasil, Thierry Coupaye, Nicolas Rivierre, Jiri Adamek, Tomas Bures, Jan Kofron, Pavel Jezek

Component Model

We intend to use the Fractal component model and its ADL (Architecture Description Language) for the architecture description and Behavior Protocols as a behavior specification language. Behavior protocols are a general concept; therefore they can be used for various component platforms.

LMU/DTU Team

Affiliation

Ludwig-Maximilians-Universität München

Team Leader

Alexander Knapp, knapp@pst.ifi.lmu.de

Team Members

Hubert Baumeister, Florian Hacklinger, Rolf Hennicker, Stephan Janisch, Alexander Knapp, Martin Wirsing

Component Model

In FACS'05, see http://www.pst.ifi.lmu.de/veroeffentlichungen/baumeister-et-al:facs:2005.pdf, we have described an abstract component model for components, ports, and assemblies. We use the interface automata approach to I/O-transition systems for describing the observable behaviour and the states-as-algebras approach for representing the internals of components and assemblies. In particular, component and port types are modelled as sorts in order to ease dynamic reconfiguration and dynamic creation and deletion. This abstract component model is used as a formal underpinning for our architectural programming language Java/A.

Focus/AutoFocus-Team

Affiliation

TU München

Team Leader

Michael Meisinger, meisinge@in.tum.de

Team Members

Michael Meisinger, Gerd Beneken, Florian Hölzl, Bernhard Schätz

Component Model

The Focus/AutoFocus approach uses the formalism of timed infinite streams to specify component behavior. A system is formed of components that are connected via directed typed channels. Components communicate via asynchronous message exchange. Component behavior is specified by giving relations of input and output streams. Component-based software architectures are formed by connecting components in a hierachical way with each other and the environment using channels. AutoFocus is a case tool that uses a simplified Focus semantics, which uses globally time synchronous component state machines to specify component behavior.
Recent extensions to Focus and AutoFocus such as for service-based software design will be used as well.

Component-Interaction Automata Team

Affiliation

Masaryk University

Team Leader

Ivana Cerná, cerna@fi.muni.cz

Team Members

Nikola Benes, Lubos Brim, Ivana Cerná, Jirí Sochor, Pavlína Vareková, Barbora Zimmerova

Component Model

Component-interaction automata (first presented in [1]) are a language for modelling of component interactions in hierarchical component-based software systems. The language aims to support formal specification of component interactions and hence provide a basis for formal analysis and verification of component-based systems. It is motivated by the need of formal analytical techniques (and thus also specification language) for interaction behaviour of components and inter-component communication in systems assembled from COTS components.

Hence it focuses not only on actions communicated in the system, but also on components that communicated on the actions, to enable detection of faulty components. Moreover, it allows us to verify the systems with respect to a set of temporal properties that can specify correct sequences of interactions among concrete components (by LTL model checking).