Adaptation and Evolution in Embedded Information Systems

November 02 - 06, 1998
Schloss Dagstuhl, Wadern, Germany

Among the most significant technological developments of the last two decades has been the proliferation of embedded information systems (EISs). In EISs, functional, performance and reliability requirements mandate a tight integration of information processing and physical processes. EISs include a wide range of applications, such as computer-integrated manufacturing (CIM) systems, aerospace systems, computerized vehicles, appliances, consumer electronics equipment, and a wide variety of systems in health care, transportation, defense, communication, power generation and distribution. The rapid evolution of EISs has triggered paradigm shifts in industry and exerted a profound impact on engineering processes throughout the system life cycle, from design through manufacturing, operation and maintenance. This trend is clearly demonstrated by the dramatic increase of the role and size of software in products. For example, currently, over 60% of the development cost of aerospace systems is software, but even an electric shaver has over 16K bytes software.
The ultimate driver of this trend is the fact that incorporation of information processing as an integral part of physical systems increases the potential interactions among physical components and processes, generates complex dynamics, and establishes component interdependencies unknown in previous-generation systems. The tight integration of "physical" and "information" processes represents major challenges for the software technology. First of all, using Brook's terminology, the "conceptual construct" of the software which represents its "essential complexity" is inextricably combined with the conceptual construct of its "external environment", i.e. with the structure of physical processes. Consequently, the overall system behavior can only be understood if information, material and energy transfer processes are modeled and analyzed together. It means that software artifacts need to be modeled together "with their context", using a modeling language - or modeling paradigm - which is meaningful for the design, analysis and operation of the whole system. Another well-known challenge in the design and implementation of EIS software stems from the fact that it serves as a component in a larger, changing, heterogeneous system. As a result of this, the EIS software can not be static, it must change, evolve together with its embedding environment. An additional challenge of EIS software technology is criticality. EIS software directly impacts the operation of physical processes and failure may cause unacceptable social or economic damage. Thus the software technology must offer methods and tools for the formal verification and validation of system level requirements, such as dependability, safety and reliability.
This seminar intends to bring together leading researchers from academia and industry in the areas of embedded information systems, adaptive software architectures and evolutionary design environments. The goal is to review state-of-the-art and map future research directions that help to answer challenges in EIS and to

investigate the potential of applying recent advances in the aforementioned fields. The seminar will include the following topics:

1. Definition and characterization of adaptive and evolutionary embedded systems:
    - needs
    - characteristics
    - case studies

2. Critical technology components
    - self-adaptive software architecture
    - architecture-based evolution
    - incremental, embeddable generators
    - transient management

3. Design environments
    - design abstractions, hybrid/heterogeneous modeling
    - verification, correctness by construction
    - incremental design
    - design optimization

Further information and presentation material are made available under
http://www.isis.vanderbilt.edu/dagstuhl98

Hubertus Franke, Bernd Kleinjohann, Janos Sztipanovits

Formal Verification of Hybrid Systems

George J. Pappas, U.C. Berkeley

One of the most important analysis problems of hybrid systems is the reachability problem. State of the art computational tools perform reachability computation for timed automata, multirate automata, and rectangular automata. In this talk, we present a new decidable class of linear hybrid systems, which are introduced as hybrid systems with linear vector fields in each discret location. This extension is important given the applicability of linear vector fields in control systems. This result is achieved by showing that any such hybrid system admits a finite bisimulation, and by providing an algorithm that computes it using decision methods from mathematical logic. This is joint work with Gerardo Lafferiere, Shankar Sastry, and Sergio Yovine

Complex embedded systems have meanwhile migrated into a wide range of application domains. They have a profound impact on the performance of the systems which they are part of because embedded systems control the information flow and the manipulation of physical processes. The fact that information processing is now an integral part of material systems rises fundamental problems related to the interaction of physical components and of software-based system functionality.

My main points of interest in this seminar are therefore:

1. Specification and validation methods of embedded system functionality
2. Interaction between system functions implemented in HW and SW
3. Engineering processes for embedded systems under consideration of standard components, multiprocessor architectures and operating systems.

Model-Integrated Computing

Hubertus Franke, IBM T.J. Watson Research Center

One of the most significant developments in the last decade has been the proliferation of large-scale, complex computer integrated systems. In these systems, functional, performance and reliability requirements mandate a tight integration of physical and other processes with information processing. Important examples for systems where embedded information technology is critical for the overall system performance are: manufacturing systems, vehicles from cars to aerospace systems, patient management systems, transportation systems, weapon defense systems, and power generation and distribution systems.

The increasing role of information technology in complex systems necessitates a substantial change in the engineering approach characterized by a shift from the conventional “discipline” and “life-cycle” orientation to an integrated “product/domain orientation”. This shift needs to be facilitated with new theories, methods, and tools to accelerate the progress in this important system category.

The practice of using models in the full lifecycle of computer-based systems has been increasingly accepted. Multiple-aspect models are extensively used in requirement specification. Models are created and refined during design, and they are used in the verification of the design. Systems engineering tools use models for performance, reliability and safety analysis. It is a general trend that design-time models are increasingly used during system operation for model-based monitoring, control and diagnostics.

The tight integration of "physical" and "information" processes makes the application of a common description of these processes not only practical but mandatory. The common description means that software components are modeled as parts of the overall system, using concepts, relations and model structuring principles that are meaningful for the design and analysis of the whole system. Since computer-based systems are very multifarious, and software components play a rapidly increasing role in their operation, the modeling paradigms offered by conventional programming environments are not satisfactory. Typical programming environments support hierarchical structure and homogeneous decomposition which is far from the heterogeneity and semantic richness of representations routinely used in many engineering domains. *The challenge is to adopt domain specific, established modeling paradigms for representing software components, while preserving the capability of translating these models into executable code.*

The long term goal of our research has been the development of a broadly applicable software technology for the design and implementation of complex, computer-integrated systems. The specific applications driving our research during the past decade have been: (a) on-line problem-solving environments for chemical plants, (b) fault detection, isolation and recovery (FDIR) systems for aerospace systems, (c) real-time facility monitoring and signal analysis for propulsion system testing, and (d) information systems for discrete manufacturing. Based on our experience, the recurring requirements in all these systems have been the tight conceptual relationships between the computer applications and their environment, the need for adapting the application/system to changing end-user requirements and operating conditions, cost sensitivity, and the stringent reliability and dependability criteria of military and industrial applications. Often these system consist of various subsystems that have to be integrated with each other and with external interfaces (e.g. data acquisition, real-time databases, operator interfaces, etc.). This integration process is often expensive and error prone.

Over the last 14 years Vanderbilt University has developed a framework based on the model-integrated computing paradigm, called Multigraph Architecture. In model-integrated computing, domain specific models are constructed for all aspects of a computer-based system, e.g. information processing architecture, physical architecture, operating environment, as well as their interactions. The models serve several purposes: (a) they formulate the entire knowledge of the system, (b) they are used for system analysis, (c) they are used for generating executables and their integration with the physical system. System evolution is such limited to model evolution. Imperative to this approach is the ability to define a domain specific modeling paradigm facilitated by domain specific tools, such as model editors, model databases and system builders. The Multigraph Architecture provides a general framework to specify modeling paradigms and to generate the associated tools automatically. The Multigraph Architecture has a proven success record in the areas of discrete and continuous manufacturing and real-time fault diagnosis. Examples of domain specific systems installed are Monitoring and Control System at the DuPont Chemical Plant in Nashville and the Saturn car manufacture’s site production flow system.

Synthesis of Embedded Generators

Gabor Karsai, Vanderbilt University

The evolution of embedded information processing systems can be modeled as two, inter-linked "process" loops. The "outer loop" involves the design process, system synthesis, evaluation, and redesign/resynthesis. The design to implementation mapping, the synthesis can be performed using a model-integrated approach, i.e. design can be expressed in terms of domain-specific models, and the system automatically synthesized from them. The outer loop has time-constants commensurate with the product's life cycle. In sophisticated systems there is an "inner loop" as well, which proceeds along the steps of synthesis, execution, evaluation, and re-synthesis. This loop is responsible for the run-time adaptation of the system, and just like the outer loop it can be implemented using model-integrated techniques. The time constants of the inner loop are commensurate with the dynamic environment where the system is placed in: we would expect a re-synthesis to take place promptly but without excess transients in the system.

In both of the evolutionary/adaptive loops mentioned above a central component is the "generator" that transforms domain-specific models into executable entities. In the "inner" or adaptive loop the generator has properties that are specific to embedded systems. On one hand, it has to access the embedded models (which were generated by the "outer" loop), and has to interpret them, i.e. assign meaning to them in terms of a generic execution model. Thus the generator acts as a "model interpreter". On the other hand, it has to interface with an observer/monitor component that evaluates the embedded system, and triggers a re-interpretation if circumstances mandate that. The re-interpretation has to be performed within a required time period, and transients should be managed. Note that the model interpreter is defined with respect to the modeling language (the “paradigm”) used, and the execution environment, and they change rather infrequently.

The hand coding of model interpreters is a rather tedious task, involving mundane programming activities. However, conceptually, the model interpretation process is rather simple: it involves traversal of data structures (the models) and taking actions at specific points in the traversal. This observation enables us to "model" the interpreter in terms of "traversals" and "visitations": the first capturing the way a model graph should be walked, the second capturing what to do at specific nodes in a graph. The traversal/visitor approach has its background in the Visitor design pattern, Adaptive programming, and attribute grammars. From a concise specification of traversals and visits we can easily synthesize the "glue-code" the surrounds the user-supplied, domain-specific code containing the actions to be taken. The model interpreter can be built in the form of two objects (Traverser and Visitor) which are connected to the embedded models, and can be triggered/activated by the monitor component. The synthesized generator can be quite compact and efficient, and can handle incremental interpretation by encapsulating incremental changes in visitor actions.

Computer-Aided Evolutionary Process Development

Andreas A. Linninger, University of Illinois, Chicago

Phenomenalogical process models play an important role in development, planning and design of chemical production processes. The ongoing challenge for systems research is to invent a new breed of computer-based methodologies for assistance and/or partial automation of the creative modeling activity.

In systems research, we develop new problem formulations and methodological approaches for the advancement of central chemical engineering problems, such as

- Design of chemicals with desired properties,
- Synthesis and assessment of chemical processes with ecological considerations,
- Analysis and optimization of the steady state and dynamic behavior of existing processes.

Recent advances in mathematical techniques, computer-science and artificial intelligence open new avenues for computer-aided engineering activities in a wide range of significant disciplines including,

- Integrated process development embedded in a unified framework of process knowledge (reasoning, simulation, and optimization).
- Systematic synthesis of chemical processing schemes for pollution prevention,
- Intelligent process supervision and control including fuzzy, logic-based and model-predictive approaches.

The current research interests focus on:

- Novel modeling languages for phenomena-driven generation of process models,
- Development and promotion of symbolic/numerical algorithms for the robust solution of large-scale non-linear process models,
- Deployment of a computer-aided product and process synthesis methodology for the manufacturing of pharmaceuticals, agricultural and specialty chemicals,
- Design of a systematic framework for the assessment and improvement of health, safety and environmental performance of chemical production processes.

We will continue to demonstrate progress through fully-functional software prototypes in close contact with the industry.

The presentation investigates problems in computer assisted process engineering from a system theoretical point of view. A solution approach for models is demonstrated. It’s foundation rests upon a novel problem organization structure into three levels of abstractions. At level1, a new class relationship composed of Models-Problems interfaces addresses issues of model refinement, model aggregation as well as problems related to information binding required for the construction of large reusable model libraries. At level-2, a phenomena oriented modeling language (PML) enables domain experts to construct continuous/discrete models by specifying concepts pertaining to physical and chemical phenomena as used in their “natural” languages. This high-level language promises rapid development and scalability of model artifacts. Finally at level-3, meta modelling will allow to dynamically evolve the paradigms of level-2 languages, e.g. PML.

Multi-Language Design

Bernd Kleinjohann, University of Paderborn

Typically the market requires IT systems that realize a set of specific features for the end user in a given environment, so called embedded systems. For different application domains like mobile phones, set top boxes for TV or control modules in cars, airplanes or buildings, a variety of modeling languages with domain specific features was developed in the past. The talk presents how different languages can be used together during design of embedded systems and which problems have to be investigated during such multi-language design processes.
A first point will be the investigation of semantic problems in multi-language design, i.e. the composition of sub-modules in different languages and the consideration of the environment and its behavior. A concept of language coupling and language integration will be introduced. Language coupling mechanisms for the connection of inputs and outputs of subsystems modeled in different languages have to be provided without changing the original subsystem models. Language integration models specified in different languages have to be transformed into a unified model.
The second part of the talk will introduce extended predicate transition nets on an integration model. By small examples it will be demonstrated how Statecharts, continuous system models, synchronous models and asynchronous models can be integrated. If a subset of a predicate transition net fulfill some restrictions, the design methods provided for the original models can be adopted for the unified integration model.

Trends in the Domain of Systems Design

Michael Mrva, Siemens AG

Trends to be observed in the domain of Systems Design are:

Trend 1. A significant increase in the call for a solid methodical basis for system description languages. Rational's Unified Modeling Language (UML) for SW design is considered to be going in the right direction and is often cited, but has also come in for criticism:
(a) no method yet, but "only" a language; (b) rather large number of heterogeneous description components, so that the required methodical stringency can not be guaranteed.

At Siemens we are investigating the **usability of the UML for HW design**.

Trend 2. A new openness toward not only technical aspects, but also increasingly toward people-technology interaction aspects of system development. Here not so much the user-interface topic is meant, but rather questions such as:

- What are the driving factors in collaboration and design reuse (in both SW and HW)?
- How can we make systems capable of evolution (changeability-windows, habitability)?
- What can we learn from non-technical disciplines such as psychology or social sciences, about the procedures involved in the development of complex technical systems?

These questions are under research at Siemens. First partial results comprise:

- Besides all necessary competition, the **presence of caring** is essential which means that team members must learn to care for each other.
- We are working on **methods** which **don't hinder** the creativity of people, e.g., through application of "alien" thinking patterns. Those alien patterns often don't fit and are mostly used because one wants to make management happy or because there is a success story elsewhere about that method.
- A further important goal for good design methods is finding the ideal **level of abstraction** as well as the ideal **degree of independence** between components.

Trend 3. An accelerated growth of hybrid technologies, with which systems, consisting of HW and SW, can be described, modeled, verified, and implemented end-to-end.

One approach, **model-based co-design**, is supported by Siemens within the framework of research cooperation with the University of Arizona in Tucson, Arizona.

Integrating a Design Methodology and Tool Sets for Embedded Systems Development

Jerzy Rozenblit and Stephen Cunning, University of Arizona

Simulation modeling is increasingly recognized as a useful tool in assessing the quality of sub-optimal design choices and arriving at acceptable trade-offs. This approach is often called “simulation-based design.” However, our working hypothesis is that computer simulation and other advanced computational tools are of limited effectiveness without a methodology to induce a systematic handling of the multitude of goals and constraints impinging on a design process. Therefore, our work focuses on the development of techniques in which design models can be synthesized and tested within a number of objectives, taken individually or in trade-off combinations. Mode specifically, we have developed a methodology called model-based codesign that lets developers create models of embedded systems independently of the hardware and software implementation. In this framework, designers use simulation modeling-based techniques to explore the feasibility of virtual prototypes and then interactively map the specification onto a software-hardware architecture. In several publications, we have elaborated on the fundamental concepts supporting model-based codesign. Here, we postulate the need for realizing the underlying co-design methodology with integrated design tools sets that meet the following desiderata:

a) provide design flow control and management,
b) provide models for creation, storage, retrieval, and modification of design data,
c) provide facilities for integrating tools independent of the physical, logical, project, object, and tool specific data formats and structures, and
d) facilitate access to data bases in the form of procedural or command interface and interprocess communication.

We are developing an environment that would meet the above requirements. This environment, called SONORA, will realize the model-based codesign methodology. The Functional and Behavioral Requirements Specification and Modeling block embodies requirement solicitation and documentation and development of an executable model. The Behavioral Simulation and Model Refinement Loop is used to iteratively refine the design model until it is functionally correct. Structural Requirements Specification and Modeling relates physical design constraints to a proposed physical architecture. In the Performance Simulation and Model Refinement Loop the model is enhanced with performance estimates for computation and communication based upon the proposed physical architecture. Synthesis and Implementation involves extracting design information from the models in order to produce a physical prototype. Experimental Frame Development and Testing involves the creation of a set of test cases based upon the system requirements that are used to asses the current design at all stages of the design process.

Transients in Adaptive and Reconfigurable Measuring Channels

Gabor Peceli, Technical University of Budapest

The study of reconfigurable measuring channels, adaptive and/or reconfigurable DSP systems is a very important area of research related mainly to large scale, distributed intelligent monitoring and control systems. To use reconfiguration techniques in such computer-based applications has real meaning if drastic changes may occur in the physical system. Changes due to faults evolving into system degradation are typical examples. In such cases, the supervising computer program should observe the changes and turn to another operation or program. In other words, the models applied within the computer program are also to be changed to correctly represent the physical system. Model changes can be performed using different techniques. For conventional system models the typical solution is the adaptation or direct change of the coefficients and/or the (signal processing) structure. These changes, however, can cause large transients, since there is a real difference between the stationary behavior of the system before and after the change. From our investigations it turned out that reconfigurations transients depend significantly on the DSP structure applied. This structure dependency is strongly related to the energy distribution within the processing structure, therefore the famous orthogonal structures, which try to distribute the energy uniformly, provide good performance. The behavior of the widely used direct structure is rather poor.
The prediction-correction type processing structures, which incorporate the model of the system generating the input signal, provide relatively good transient behavior. If during reconfiguration the system order is also modified, it is important to assign the new state-variables to the previous ones in a systematic way. The proper strategy is not available yet, further investigations are required. The signal processing structures incorporating the model of the input signal can be suggested to serve the so-called “any-time” algorithms, which are expected produce acceptable output even if there is a temporal shortage of input data and/or computational power.

Traceability among Software Artifacts Based on Meta-Modelling

Antje von Knethen, University of Kaiserslautern

Technical systems, like automobiles, washing machines, or building automation systems are widely distributed and depend more and more on software. The embedded software in these systems has to be easily adaptable to requirement changes that are unavoidable considering the longevity of technical systems. Today, various notations are used to describe different abstraction levels (e.g., system requirements, software requirements, and software design) and different views on one level (e.g., static and dynamic view). The relationships among different abstraction levels (vertical traceability) and among different views (horizontal traceability) are often not explicitly documented. Therefore, it is difficult to analyse the impact of a change (i.e., the definition of the elements that have to be changed to get a consistent software documentation at the end) and to implement a change consistently. An explicit documentation of vertical and horizontal traceability can support impact analysis, implementation of changes and consistency checking. One typical approach to support vertical traceability is to apply a requirements traceability tool, like ARTS, DOORS, or RTM. Such tools focus on traceability among requirements (mostly described informally) and documents on subsequent abstraction levels. Moreover, they support horizontal traceability on requirements level. The relationships among the components have to be set manually. The tools manage and visualize the relationships by generating matrices, cross references, or entity-relationship models. Disadvantages of using such tools are that the effort for establishing traceability is expensive and the granularity of the components that can be traced is typically too coarse for accurate impact analysis. One typical approach to support horizontal traceability is to apply meta-modelling (the main goal is to guarantee consistency between different views). Each notation applied to describe a view is defined on a meta-level (meta-model). On this meta-level, consistency rules among the notations can be defined (i.e., to define semantic relationships among views). The defined rules can be used to analyse the impact of a change in one view on other views and to check consistency among the different views automatically (if the description language used on the meta-level is formal). It is possible to transform one model into another with transformation rules developed on the basis of the consistency rules. For a transformation, the information represented by the models has to be the same. The advantage of the approach is that the effort for establishing traceability is low and the granularity of the components that can be traced is fine. My approach is to apply meta-modelling to vertical traceability among system requirements, software requirements and software design. This means that the notations applied to describe the abstraction levels have to be defined on a meta-level. Then consistency and transformation rules among components on the different levels have to be developed. Therefore, information about design decisions is required. The information can be taken from design guidelines (i.e., guidelines described by methods, like OCTOPUS or FUSION), design patterns, and architectures. Based on the meta-information (i.e., meta-models and rules) a tool should be developed that should generate some artifacts on the subsequent abstraction level. A complete definition of consistency and transformation rules (i.e., a full semantic description) for the vertical traceability among the abstraction levels seems to be impossible (e.g., if the semantic relationship between components depends on a creative process). Therefore, the manual setting of relationships should also be supported by the tool to be developed. With the tool, the impact analysis of a system requirement change, the implementation of changes and the checking of consistency should be supported.

Adapters and Adaptive Computing Systems Benchmarking

Sanjaya Kumar, Honeywell Technology Center

FPGAs (Field-Programmable Arrays) have attracted quite a bit of interest as an implementation technology. Recently, work has focused on the development of techniques to support dynamic reconfiguration of FPGAS, providing another element of adaptation in systems.
This presentation discusses two aspects of reconfigurable technology being supported by DARPA (Defense Advanced Research Projects Agency):

1) Evaluation Technology (ACS (Adaptive Computing Systems) Benchmarking),
2) Programming Development Environments (Adapters).

The ACS Bnechmarking Program developing a suite of benchmarks for evaluating configurable computing systems. 6 Benchmarks have been developed (5 stressmarks and 1 CAD benchmark). A stressmark is a benchmark that focuses on a specific characteristic or property of a reconfigurable system. For example, the versatility stressmark focuses on the ability of an infrastructure (both tools and architecture) to implement a variety of functions using a specified sequence of steps. The Adapters program is developing three core technologies: (1) programming environments, (2) modeling and analysis techniques, (3) dynamic reconfiguration techniques for FPGAs. The focus of the program is to develop technologies that can be used to map a complex application onto a heterogeneous collection of resources: general purpose processors, application-specific processors, and FPGAs. One class of applications being explored is mode-based systems. In addition, partial reconfiguration ideas are being investigated. The following platforms are being used to illustrate the technologies being developed; Alacron board, VCC Hot Works board and Annapolis Micro Systems Wildforce board.

Composition of Software Hardware Systems Gesture-based Interaction/Programming

Pradeep Koshla, Carnegie Mellon University

We are developing technologies that allow users to:

- Create robots that are customized for tasks from modular and reconfigurable elements
- Create automatically real-time software (from software modules that are intelligent) for robot control
- Interact with robots using gestures (based on apent oriented programming).

Our goal is to create the hardware and software infrastructure for next generation intelligent systems. Our vision is that such systems will be able to interact with humans and interpret their intent. Based on this interpretation, the system will automatically create its control programs. We are applying these futuristic ideas to robot arms and distributed mobile robots. The distribute robots range in size from 2m (all terrain vehicle) to 5 cm on the side (millibots).

Architectural Refinement Calculi

Jan Philipps, Technische Universität München

It has long been recognized that there are high demands on the correctness of embedded software systems. Consequently, in the last years an impressive amount of work has been produced on the formal verification of embedded systems, partly using deductive techniques such as theorem provers, partly using model checking.

Most of this work, however, focuses on the verification of single components or of black-box-views of embedded systems. Architectural and structural aspects of systems have largely been ignored, and there are no satisfying techniques to reason about the evolution of system structures caused by changes in technology, additional requirements, or new product variants.

To address these questions, we believe that design processes for verifiably correct systems need rules to formally describe structural changes of the glass-box description of a system. Such rules could appear in various forms: as simple deductive rules, similar to first-order formulas, or as graphical manipulations dealing with the addition and removal of components and their connections. Of course, there will be side-conditions for a well-formed rule application. It is important that these side-conditions can be kept local, so that they can be discharged using established verification techniques such as (component) invariance proofs or model checking.

Some initial work on structural refinement calculi, which, however, is not specifically targeted at embedded systems, is presented in the references below.

Top Down Design of Mixed Signal

Klaus Waldschmidt, J. W. Goethe Universität Frankfurt

The design of mixed-signal systems is essential in the domain of embedded systems as for example in telecommunication, avionic and automotive applications.
Mixed-signal systems consist basically of software, off-the-shelf digital hardware and customized digital hardware. For the digital processing of analog signals, incoming signals from the analog environment must be converted to digital signals. Besides sensors and actuators this requires analog components for the analog signal preprocessing, such as filters, nonlinear operators or amplifiers. These analog components are often integrated together with the digital components to build a mixed-signal system ``on a chip".
Today, the design of the digital parts of such systems can be done in a very systematic „top-down" design flow, which starts with an algorithmic specification and is supported by tools for hardware/software-codesign or high-level synthesis. However, this designflow neglects some important aspects of mixed-signal system design:

- Mixed-signal systems often perform signal-processing functions. The choice of important parameters such as bit-widths, sampling-rates or methods for filtering and conversion are determined intuitively.

- The design of analog components goes rather bottom-up, takes much time and requires a lot of expert-knowledge and experience.
- An approach which is as general as the register-transfer synthesis is still missing in the analog domain.

These two aspects of the design of mixed-signal systems are the main focus of this presentation.
In this presentation, a top-down methodology for the design of mixed-signal systems is proposed. The proposed methodology structures the design of mixed-signal systems into a sequence of transformations on a graph-based model.
The benefits of such a design methodology are obvious:

- The top-down design-process leads to a more systematic exploration of the design space.
- The determination of system-parameters and the global management of resources as part of the design-process makes these parameters available for an optimization regarding implementation-costs.
- The methodology can be automated easily.

The proposed methodology is based on the formal model of hybrid data-flow graphs (HDFG). HDFG permit the homogeneous and graph-based representation of arbitrary hybrid systems. The specified behavior of a mixed-signal system can be represented by HDFG as well as the chosen architecture (structure). The design-process can then be represented by a sequence of transformations that start with the HDFG-representation of the specified behavior, and that ends with a HDFG-representation of a chosen structure.

In order to support the design of mixed-signal systems, a VHDL-AMS or block-diagrams can be translated to HDFG. Well-known optimization techniques for data-flow graphs can be applied on HDFG. Furthermore, transformations can be performed between different models of computation. Finally, the HDFG can be mapped onto discrete, functional blocks by a graph-covering algorithm. Outputs of all functional blocks are buffered with registers in the digital and with operational amplifiers in the analog domain. Each functional block can then be designed separately, either automatically or in an interactive way.

The proposed top down design methodology for mixed-signal systems is now under evaluation within concrete applications.

Representation Issues in Self-Adaptive Computing

Janos Sztipanovits, Vanderbilt University

Embedded information systems provide ample evidence for the need of self-adaptive behavior. A common challenge in these applications is the unpredictable number and kind of events emerging from the physical environment that impact fundamentally the required software architecture. For instance, in position control of manipulators the controller receives the measured position and speed of the manipulator, and calculates a control signal. If one of the sensors breaks down, control can still be maintained but the architecture of the controller must be changed. This change impacts the signal flow and complexity of the computations which in turn requires change in the software architecture of the controller.

The model-integrated approach to self-adaptive software decomposes the problem into two major issues: (1) representation and (2) the reconfiguration mechanism. The representation issue deals with modeling of self-adaptive systems, including models of architectures and adaptation processes. The objective of the research on representation is finding the appropriate level of abstraction, modeling constructs and modeling paradigms that facilitate a manageable design process for self-adaptive systems. The reconfiguration mechanism focuses on methods for mapping the models into executable systems, and changing the dataflow and control structure of the application in a safe, consistent manner. Summarized below are considerations for representation strategies in self- adaptive software.

Representation in self-adaptive software faces two primary challenges:

- separation of the time-variant and time-invariant elements of the software, and
- formalism for the representation of the time-variant components.

The justification of decomposing self-adaptive software into "time-variant" and "time -invariant" components deserves some consideration. Since software, self-adaptive or not, defines behaviors/trajectories in an infinite state-space; why not to use existing technology? The argument is similar to that of used in the theory of adaptive dynamic systems. Adaptive dynamic systems are time-variant, non-linear systems. However, they are conceptualized as an "adapted system" (time variant component) and an "adaptation algorithm" (time invariant component) in order to make their design manageable. The design of self-adaptive software can be formulated in this conceptual framework which offers similar theoretical and practical advantages.

Selection of time variant characteristics of an adapted system is another fundamental issue. The most frequently used method in building adaptive signal processing or control systems is to adapt carefully selected parameters of the adapted system. The goal in self-adaptive software is to change system behavior through adapting the composition of a running system. Accordingly, the representation in self-adaptive software must include formalism describing the time variant composition of the adapted system and must provide constructs for expressing the adaptation process in terms of composition changes. Time variant system composition can be modeled by means of trajectories in a "Design Space". Transitions in the Design Space are driven by state transitions in the computing system, in its environment, or both. Points in the Design Space represent architectures that can be composed form the available resources. A representation theory in this framework requires solution for the following problems:

- State-Space representation: modeling the discrete state space describing the behavior of the computing system and/or its environment for the adaptation process.
- Design Space construction: modeling the possibly very large number of meaningful architectures that can be created from the available resources.
- State-Space and Design Space relationship: modeling the mapping between the State Space and the Design Space.
- Design Constraints: modeling the functional, compatibility, resource, performance and other constraints that define the set of meaningful architectures in the Design Space.
- We will discuss a formal representation approach that enables the design and analysis of self-adaptive systems over finite (but possibly very large) state and design spaces.

Adaptive Computing Runtime Environments

Ted Bapty, Vanderbilt University

High-performance, embedded applications must function efficiently in rapidly changing environments. Power and volume constraints limit hardware resources, while extreme performance requirements demand algorithm-specific architectures. Reconfigurable computing devices address these problems by allowing the architecture to change in response to the changing environment and changing algorithms.

With the advent of Field Programmable Gate Array (FPGA) chips, the task of designing reconfigurable hardware is relatively straightforward. FPGA's are typically used for computations and bus connections, along with other technologies such as RISC processors, DSP's, and fixed-function ASIC's.

Implementation of the application software is more difficult. The design challenge lies in the need to implement the many modes of system operation using a common set of hardware resources. The designer must implement a separate configuration for each operational mode, optimizing the system to function within the minimum overall hardware envelope.

Given the complexity of the operational modes and the heterogeneous, changing nature of the hardware target platform, significant design and runtime support is required. These can be divided into two categories: high-level design tools for requirements, algorithm, and system resource capture and manipulation; and mid/low-level runtime infrastructure. The concepts and implementation of the high-level design environment can be found in. The critical point of this division is the interface between high level design and the runtime infrastructure. This interface must present a high-level abstraction of the underlying hardware/software environment to simplify system synthesis.

The underlying runtime system must support the system synthesis tools, implementing the functional system. The functions for the runtime system include:

- Abstraction of Hardware/Software Interfaces: Hardware details must be hidden to support a high-level, uniform view of the underlying resources. A real-time dataflow paradigm is implemented.
- Configuration of resources: FPGA's must be programmed with the proper configurations for the system operating mode. Real-time schedules must be constructed and installed across parallel DSP and RISC processors. Communication topologies must be implemented in the messaging fabric.
- Control of dynamically changing system state: The infrastructure must support the dynamic reconfiguration of the executing system. This involves rapid transition from one computational architecture to another. The transitions include changing hardware topologies, processor schedules, and communication maps. System consistency must be maintained during transitions.
- Implementing timing constraints: The real-time behavior of the system must be maintained, during normal computation/communication, and during reconfiguration.

The semantics of the execution environment implement a large-grain dataflow architecture. Processes and Processors are equivalent, representing functions on data. Processes/Processors are connected via logical Streams/Signals which must buffer, communicate, and match data formats.

The execution environment spans software and reconfigurable hardware. The software environment consists of a simple, portable real-time kernel with a run-time-configurable process schedules, communication schedule, and memory management. Communications interfaces are supported within the kernel, making cross-processor connections invisible. Memory management is integrated with the scheduler and communication subsystems, enabling (but not solving) the problems associated with dynamic reconfiguration. The software kernel uses the communication infrastructure broadcast commands from the Reconfiguration Manager to receive configuration information.

The hardware execution environment is semantically similar, but the implementation is much different. The Virtual Hardware Kernel exists as a design paradigm only: no operating system executes on the FPGA hardware. Instead, the necessary communication and computational components implement the dataflow functionality. The MIC Generator synthesizes the processors, the signal buffering, and the necessary off-chip interfaces and data converters.

The Reconfiguration Manager manages resources and executes the system behavior state machine. Using results from computations, it evaluates state transition decisions. When a transition is indicated, the manager performs the orderly shutdown of existing components, reconfigures hardware components, and reinitializes the system with the new mode of operation.

Status: The environment is still preliminary. A set of intrinsic components are being constructed for the hardware communication interfaces. Dynamic reconfiguration approaches are being tested and refined. Applications are being built to evaluate and refine components and approaches.

The Design of Embedded Real-Time Systems using Extended Predicate/Transition-Nets

Jürgen Tacken, C-LAB

In recent years embedded systems have gained increasing importance. Due to the increasing functionality they have to be designed in teams with several specialists, each of them working on one single part of the whole system. But the focus in design is not only more functionality and higher performance but also safety and reliability criteria that have to be fulfilled by the designed components. This includes functional requirements as well as real-time constraints.
By now every single area of application has developed its own well understood techniques for modeling, combined with corresponding methods and tools for analysis and simulation. Already at the state of many individual models many predictions about the temporal and functional behavior of each subsystem can be made. But to validate the behavior of the whole system, the individual models have to be coupled.

Especially for the systems' reliability it is important to consider not only each single component on its own but its behavior within the whole context. Many functional errors only expose themselves when all individual components work together in the whole context, observed over time.

Often the models are given in different domain specific modeling languages, so coupling can only be realized either on the level of simulation or within a hybrid language, that usually does not offer any facilities for further analysis. Coupling of simulators allows a hybrid simulation in the sense of a combined simulation of all subsystems, while each subsystem may be formulated in a different language for a specific simulator. On the level of simulation temporal and functional behavior and performance can be studied and validated. But not all errors can be found by simulation because of the exhaustive number of possible simulation runs. Hence it is desirable to run a formal analysis of the static and dynamic properties for formal verification purposes.

A further problem in a separated design process of different subsystems is that each individual domain has proprietary methods of optimization, but for a global optimization of the joint system no facility exists. Especially when coupling different subsystems it may be useful and more cost-effective to export some functionality from one into another subsystem. An overall analysis of the joint system may reveal states that can never be reached and therefore can be eliminated from the design. The prerequisite for this kind of analysis is that all models are given in an uniform language.

In the talk I will describe our method for the design of embedded real-time systems. During the specification and modeling phase it allows the use of several domain specific modeling languages. All these different languages are transformed into one common model using extended Predicate/Transition-Nets. Predicate/Transition-Nets are a high level form of Petri-Nets. This modeling language is very powerful since it must have at least the functionality of every single domain specific modeling language.

After combining all the different designs a global analysis and formal verification of the system may be performed on the Predicate/Transition-Net Model. To apply efficient and useful analysis and verification methods together with a corresponding software and Hardware synthesis the modeling language should be less powerful and very restrictive. This is a contradiction to the first claim of a powerful language. So how can this conflict be solved?

Since it is apparently more effective to reuse existing analysis methods than deducing new ones for a powerful modeling language, we decided to use less powerful modeling languages for the analysis phase of our design method. This less powerful modeling languages are reduced versions of the extended Predicate/Transition-Nets with properties like determinism and synchronism that support an effective analysis and synthesis of the specified models. In the talk I will describe strategies how extended Predicate/Transition-Nets can be transformed into the less powerful versions and show how analysis and formal verification can be performed for this model.

Adaptation Techniques in Embedded Real-time Communication Systems

Franz J. Rammig, University of Paderborn

Embedded Real Time Systems tend more and more to become distributed and parallel ones. This is true for both, the platform for the design process and the embedded system itself.
Reason for parallel and distributed platforms for the design of embedded systems are the needed computing power, the natural parallelism of the systems to be developed and the distributed design process itself.
Concerning the target system it makes sense to distribute the controllers to the objects to be controlled rather than concentrate all controlling activities in one centralised computer. This distribution results in very short connections of high communication bandwidth (between the dedicated controller and the controlled object) while relatively low bandwidth is needed for the communication between the controllers.
It depends on the computing power needed for such a dedicated controller, whether a parallel architecture is needed or not.
Traditionally distributed controlling systems have designed just as a collection of individually designed dedicated controllers. In such an approach only the local application supported by a local RTOS has to be considered. Any kind of a general purpose interconnection system with enough bandwidth then might serve to support the communication.
Obviously this approach leads to sub-optimal solutions. Therefore holistic approaches have to be considered. In such approaches the entire system to be designed as a distributed system is considered. This leads to distributed application programs, to distributed RTOS and to a Real Time Communication System (RCOS) which now becomes a component of its own.
In our contribution we want to discuss design support for such a scenario. We strongly belief that a design process for complex embedded systems has to cope with heterogeneous specifications using different description paradigms. These multiparadigmatic descriptions have to be mapped to a unified modelling platform. In our case we use extended predicate/transition nets for this purpose. Based on this unified internal format design techniques adapted from the design of digital hardware can be applied. As a result a distributed embedded RT application is obtained that runs on a network of microcontrollers, based on a distributed RTOS and supported by a distributed RCOS.

High-Level Embedded System Specifications Based on Process Activation Conditions

Wolfgang Bossung, University of Darmstadt

High-level specifications for the behavior of information processing systems consist of data and control flow descriptions as well as of timing requirements. These are to be met by feasible implementations. Using a functional partitioning of a system, a process net description with conditional process activation is proposed. The simulation of token flow leads to a schedule that makes investigations in the timing analysis of the proposed Codesign Model (CDM) possible. Predictions about the delay between any two nodes of the system are also possible, as well as the speed of processing external inputs and outputs, iteration times of determined periods and, hence, all derivable time criteria. A formal notation of process nets as cyclic graphs is given, which is useful for the description of complex digital embedded systems. n this context of embedded systems design and its specification by the proposed model, some terms and properties are to be introduced. A process P consists of an ordered set of atomic activities, which embody a basic function of the embedded system. A processor element PE denotes a set consisting of an allocated processor, the associated executable code, and the process shell. A processor is a piece of hardware, either with an own instruction set such as standard/DSP/ASIP processors or programmable in a more general sense such as ASIC/FPGA. The process code is an executable program code in the process core as part of a PE. And the process shell is an additional program code dedicated to the definition and execution of communication tasks of a process within the process net. To every process running on a determined hardware architecture, an activity time can be associated. A computing time for every process in the CDM graph is introduced to generate a schedule. In general, scheduling with resource allocation on a graph structure, as proposed in the talk, is NP-complete. However, the proposed simulation methodology delivers detailed information on conditional paths of data and control flow in a CDM. The simulation can be terminated at every point in time for evaluation of the results. Scheduling of a new specified CDM will lead to a guided refinement of the CDM. Internal state transitions for processes need not necessarily be specified to start with this kind of scheduling. With regard to the specified I/O relations, a simulation of the schedule makes all state sequences of processes obvious. A designer can eliminate non-appropriate sequences step by step and restrict alternate paths of data gained from simulation. So, the simulation time decreases for a specific number of simulated iteration cycles.

Composable Simulation for Design of Mechatronic Systems

Chris Paredis, Carnegie Mellon University

We are currently developing a framework for composable simulation in which simulation of mechatronic systems is tightly integrated with design. With composable simulation we mean the ability to automatically generate simulations from individual component models by manipulating the corresponding physical components in a CAD system. Associated with each physical component, there are multiple model fragments describing the component's behavior in multiple energy domains and at multiple levels of detail within a single energy domain. Based on system-level simulation requirements, the appropriate simulation models are selected for individual components and combined in a system graph (a linear graph representing the energy flow through the system). In a model composition phase, the system level dynamic equations are extracted from the system graph and are compiled and executed in a simulation kernel. The goal is to allow the designer to specify the complexity and fidelity of a simulation such that it supports the required analyses with minimal computational cost at each stage of the design process. This form of virtual prototyping will reduce the design cycle significantly by providing immediate feedback to the designer with minimal intervention of simulation and modeling specialists.

Introduction of Delay-Insensitivity into HW/SW Codesign Methodology

Wolfram Hardt, CLAB Paderborn

Today's computation power available for execution of design algorithms allow the handling of much more complex tasks than a few years ago. On the other hand there are well established and in praxis proven design environments. Such environments implement design methodology. New challenges make the introduction of new complex design tasks necessary. This can not be realized by a complete change of design methodology but such tasks have to be introduced into the overall design methodology. For demonstration we have picked the very old idea of delay-insensitive design and introduced it into the HW/SW Codesign methodology. It points out that for each step in the design methodology an equivalent step for the introduced aspect is needed. We have chosen a mixture of top down and bottom up introduction. On high level performance quantification concepts allow the application of analysis and early partitioning. On lower level implementation technology is needed as well as the implementation of functionalities. In our example –referred to by FLYSIG-bitserial implementation of dataflow oriented algorithms has been chosen. This is reflected by the architecture based on the mult-ring concept introduced by Staunstrup. The mapping from analysis results onto the register transfer level is performed by high level synthesis algorithms which may be taken form the existing design environment.

Active Software Composition

Robert Laddaga, MIT & DARPA

Active software composition (ASC) is a DARPA research program whose goal is to enable the development of programs that can modify their behavior in response to changing conditions and changing needs. This software self-adaptation is accomplished by having the software evaluate its progress to gaol, and select alternative approaches based on the evaluation. Thus, the software needs to have the following capabilities:

1) evaluate progress to goal
2) have access to alternative algorithms and implementations
3) select an appropriate alternative based on evaluation
4) reconfigure and optimize for newly selected alternatives.

Clearly, to have these capabilities, the code will need to contain and operate on explicit descriptions of intent, specifications, design, program structure and mapping of tasks to functions and modules. Also the code will need to contain a large number of alternative algorithms and implementations, or have access to such alternatives over network connections. The key problem to solve is how to evaluate progress to goal. Classification of problems into constructive and truth finding (with or without ground truth available) classes is discussed, with examples of evaluators for each class. A subsidiary problems is the need to avoid the potentially significant additional work of writing evaluators. The potential for generating evaluators (as well as principle code) from formal specifications, is discussed. Also presented is a broader context of active code and data, which includes mobile software, tolerant software and self-organizing data, in addition to ASC.

Large Scale Distributed Simulation

Zoltan Papp, TNO Institute of Applied Physics, Delft

A certain type of large-scale, mixed continuous/discrete event simulators belongs to a particular type of structurally adaptive dynamical systems. The problem domain can be characterized as follows: the behavior of relatively big number of highly autonomous dynamical entities has to be simulated. The entities can observe the embedding world (which also contains some representation of other entities, for) and they can receive transmissions from other entities. According to these inputs an entity changes its internal states, can broadcast messages and can make transformations on the world (including other entities).
The main topic of the research is the design of a runtime environment, which (1) enables distributed implementation on messaging hardware architectures, (2) provides scaleability, (3) maintains hardware/software platform independence. A modeling concept has been introduced, which by incorporating a sensor/actuator abstraction layer can separate clearly the simulation environment dependent components and the experiment independent runtime infrastructure and simplifies the entity interactions for quantifying entity relationships.
The formalization of the sensor/actuator abstraction makes the generation of the relation maintenance and database management components of the runtime system possible. This way the runtime system can be optimized with respect to communication overhead, which is the key for successful application of this framework in various application domains. Currently the research focused on traffic micro-simulation applications.

Rapid Prototyping of Embedded Systems

Klaus Buchenrieder, Siemens AG

Embedded systems are everywhere and many devices that appear to belong in the hardware domain are actually software products. Without embedded systems (ES) many appliances and devices would be more expensive, larger and harder to use. The right mix of embedded software and hardware differentiates price and decides over business success or failure.
The processing power of a traditional ES comes either from a core, an IP or from a reconfigurable element. The combination of embedded software and soft-configuration is the key to flexibility. Versability is achieved through system and core/IP parameterization whereby reprogrammable HW structures at the core-level are mainly responsible for system performance, flexibility and affordable price. The trend however, is to increase the flexibility and performance of ES to satisfy the needs of new challenging applications. Especially when subsystems guide users in accomplishing a task. This higher degree of flexibility is achievable only through adaptation or evolution methods. Changes of HW and SW capabilities, ensure that ES can properly evolve by development of new characteristics and properties to master new situations quickly. As a result, embedded systems can fulfill goals under changing environmental conditions, while maintaining the specified performance.

Our research focuses on prototyping and debugging of advanced E-HW/SW systems. Prototyping allows us to continuously measure the performance while a HW/SW prototype performs its task in a real-world environment. The clear advantage over simulation or emulation is that function checks and adaptation/evolution steps can be performed and observed under real circumstances.

Domain Specific Model Architecture for Complex Embedded Systems: A Building Automation Case Study

Gerhard Zimmermann, University of Kaiserslautern

There is a large clan of complex embedded systems that are both reactive and distributed with real-time and fault tolerance requirements. Digital control systems ot automobiles, buildings, and airplanes are typical examples. Software engineering has to provide techniques and methods to support the efficient and reliable development of such systems. It is our goal to provide domain specific methods for the development of well structured models for complex problems with a considerably smaller effort than universal methods can achieve. This paper concentrates on the analysis phase of the software engineering process for embedded control systems, based on an available modeling language (SDL). The contribution of our group is a new model architecture called “organizational architecture” because it is similar to hierarchies in organizations. This architecture can be supported by a small number of model patterns. A specific modeling method is shown that applies this architecture to control problems in an efficient way. The resulting models build system requirements specifications that can be automatically compiled into prototype software for validation and verifications using commercial tools. A case study with a complex building automation system with more that 400 controlled devices has shown, that this method can be easily applied, that the proposed model architecture is well suited to structure control systems in the chosen domain, and that the analysis phase was completed in a very reasonable time. It also helped the domain experts to express their overall solution of the control problem at an abstract level. Further case studies will be executed to demonstrate the advantage of reuse of generic artifacts during the analysis phase.