Annual Report of the Chair of Computer Science 2 (Programming Systems)

Address: Martensstr. 3, 91058 Erlangen
Phone: +49-9131-85-27621
Fax: +49-9131-85-28809
E-Mail: info@i2.informatik.uni-erlangen.de

Ordinarius:
Prof. Dr. Michael Philippsen
Honorary Professor:
Hon.-Prof. Dr.-Ing. Bernd Hindel
Hon.-Prof. Dr.-Ing. Detlef Kips
Professor Emeritus:
Prof. em. Dr. Hans Jürgen Schneider
Secretary:
Agnes Brütting
Scientific Staff:
Dipl.-Inf. Thorsten Blaß
Dipl.-Inf. Daniel Brinkers
Dipl.-Inf. Georg Dotzler
Dipl.-Inf. Stefan Kempf
Dipl.-Math. Jakob Krainz
Andreas Kumlehn, M. Sc.
Dipl.-Inf. Christopher Mutschler
Dr.-Ing. Norbert Oster
Norbert Tausch, M. Eng.
PD Dr. Ronald Veldema
Dipl.-Inf. Tobias Werth
Guest:
Dr.-Ing. Josef Adersberger
Dipl.-Inf. Samir Al-Hilank
Sören Braunstein, M. Sc.
Dipl.-Inf. Ralf Ellner
Dr.-Ing. Martin Jung
External Teaching Staff:
Dr.-Ing. Klaudia Dussa-Zieger
Dr.-Ing. Stephan Otto

The Chair of Computer Science 2 (programming systems) was founded in 1972 and is headed by Prof. Michael Philippsen (as the successor of Prof. H.-J. Schneider) since April 2002. Closely associated with the programming systems group are the professorship for Didactics of Computer Science and the professorship for Open Source Software.

1 Focus of research

The main research topics in the programming systems group are programming of parallel or distributed systems and programming of embedded or mobile systems. Software (and its development) for such systems should ideally be as complex, portable, maintainable and robust as existing software for single core systems and workstations. It is our long-term goal to allow applications to take full advantage of the available computing and network power. A particular focus lies on programming systems for multi-cores because more and more cheap multi-core high-performance parallel hardware (for example graphics cards or FPA-Hardware) is available. This will have an unpredictable impact on the future of the software landscape. Research results of the group are always evaluated by means of prototypes and demonstrators.

Important Research Areas

- **Exploit the available parallelization potential.** In the future the clock rate of multi-core systems will grow only slowly whereas the number of cores will grow. This makes it necessary to exploit the parallelization potential of already older, existing software to allow it to benefit from the new hardware. As a consequence, in most application areas a change to parallel computing is unavoidable. Therefore, the programming systems group develops tools to support the programmer interactively in reengineering existing sequential applications. It also develops architectural patterns for new software projects that scale automatically to support a growing number of cores.

- **Achieve portability in high-performance applications.** Up to the present, application programmers achieve the best possible performance results only if they handle latency issues and communications between different components of the system manually, optimize their code with hardware specific ”tricks” and split their application into multiple sections to outsource them to other hardware (for example graphics cards). To change this situation, the programming systems group researches the performance impact of higher programming abstraction layers that would improve programming productivity and software portability. The improvements are caused by generated code that allows the distribution of the program onto multiple heterogeneous system components to permit parallel execution. The higher abstraction layer makes the communication between the components transparent for the developer. To increase the efficiency of this approach it is necessary to give the programmer the possibility to express available domain knowledge in the programming language. For the higher abstraction layer, the details of the hardware architecture are hidden from the developer (for example by library functions or programming language extensions).

- **Adapt the degree of parallelism dynamically**. High-performance applications are often developed for a fixed number of cores. As requested cluster nodes of a batch system are statically assigned for a fixed time period, inefficient reservation gaps are unavoidable. Similar problems appear in multi-threaded applications on multi-core systems. The programming systems group works on the dynamic adaptation of the extent of parallelism by the means of code transformations (under consideration of the resulting data redistribution) and operating system interactions. As control flow based synchronization measures interfere with the necessary analyzes, the programming systems group researches new programming constructs that can replace the existing ones and allow to specify the synchronization in a data-centric way.

- **Develop Testing for Parallelism**. In software engineering, testing has always assumed an important role. Code coverage, test data generation, reliability assessment etc. are tools of the trade. Unfortunately, current research insufficiently covers the indeterminism caused by concurrency. To deal with that issue, the programming systems group develops tools that consider (based on the coverage criteria) interleavings of parallel threads in their test data generation. This topic also includes research on operating systems and schedulers. As concurrency considerably increases the search space of the test generation it is necessary to develop infrastructures that allow the test generation and execution on a cluster.

- **Improve of Software Development Processes**. The current development practice of complex, business or security critical software in global distributed teams (commonly found in the software industry) demands compliance with well-defined software development processes. To support the enforcement of this requirement, appropriate development tools are used. The Practical Software Engineering research group that is lead by the honorary professors Dr. Bernd Hindel and Dr. Detlef Kips cover the corresponding research area. Both possess long term experience in industrial software projects as managers of medium sized software companies. The goal of the Practical Software Engineering group is the development of a machine executable notation for modeling of software development processes. For that purpose the research group examines the semi-automatic retrieval of traceability information from the artefacts of different tools and notations as well as the model based development, integration and configurations of software components, used in the design of automotive embedded systems.

2 Research projects

2.1 ErLaDeF - Embedded Realtime Language Development Framework

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
PD Dr. Ronald Veldema
Andreas Kumlehn, M. Sc.
Sören Braunstein, M. Sc.
Dipl.-Inf. Thorsten Blaß
Dipl.-Math. Jakob Krainz
Dipl.-Inf. Daniel Brinkers
Start: 1.1.2012
Contact:
PD Dr. Ronald Veldema
Phone: +49-9131-85-27622
Fax: +49-9131-85-28809
E-Mail: ronald.veldema@fau.de

ErLaDeF is our test-bed for new programming language and compiler techniques. Our main focus is on building infrastructure for easier (hard + soft) real-time embedded parallel systems programming.

Real-time and embedded software have hard constraints on resource usage. For example, a task should complete in a fixed amount of time, have guaranteed upper-limits on the amount of memory used, etc. Systems software is software that is critical to the functioning of a device. For example, a device driver is such a critical piece of software. Current and future embedded hardware is, however, multi-core so that the embedded, real-time systems programs mentioned above need to be parallelized while still observing the constraints that the real-time, embedded software layers impose.

We are developing different ways to manage this concurrency using a combination of strategies: simpler language features, automatic parallelization, libraries of parallel programming patterns, deep compiler analysis, model checking, and making compiler analysis fast enough for interactive use.

- Using simpler language features (compared to, for example, Java or C#) ensures that the compiler analysis finds the existing concurrency and real-time violation bugs. This forms a basis of the other techniques we pursue in this project. In 2012 we have explored alternatives to polymorphism and inheritance that may be easier to analyze. We have also examined alternative thread-synchronization methods, for example transactional memory, implicit synchronization, remote procedure calls, etc.

- Our automatic parallelization efforts are currently focused on dynamic parallelization. While a program is running, it is analyzed to find loops where parallelization can help performance. Our current idea is to run long-running loops three times. The first two runs analyze the memory accesses of the loop and can both run in parallel. The first run saves for every memory address, in which loop iteration a write access happens in a shared data structure. We do not need any synchronization for this data structure, only the guarantee that one value is written to memory, when two concurrent writes happen. In the second pass we check for every memory access, if it has a dependency to one of the saved write accesses. A write access is part of any data-dependency, so we can find all types of data-dependencies. If we don't find any, the loop is actually run in parallel. If we find dependencies, the loop is executed sequentially. We have already seen performance gains using this method. In 2012 we have made progress on our parallelization framework. We have started work on techniques to allow the first few iterations of a loop to run sequentially while we concurrently determine if the rest of the loop can be run in parallel. This saves the cost of waiting for the inspection passes to finish in case the loop turns out to be not parallelizable.

- A well-debugged library of parallel programming patterns allows a programmer to select well-known parallelization and inter-core communication strategies. We are performing research into what (communication) patterns actually exist and where they can be applied. In 2012 we have examined different ways to let cores communicate values over communication channels and made a preliminary overview of higher-level communication patterns. Already some 30+ different ways of letting cores communicate have been discovered. For each of the 30+ patterns we have created a list of different properties and usage constraints. The main usage for our study is for determining when a communication pattern applies in a hard real-time environment. A pattern that is blocking for both reads and writes is, for example, less suitable than a pattern that never blocks. We will next try to automatically select one of these patterns for a given use case.

- Deep compiler analysis not only covers single functions, but it considers the whole program to find bugs. An example of a deep analysis that we are currently pursuing is heap analysis. Heap analysis can answer questions such as: Which variables have a reference to this object? Which objects have references to this other object? In 2012 we have made good progress into creation of a program analysis framework that allows partial lazy evaluation of programs and that already allows partial incremental analysis. More work is needed to make the analysis fully incremental.

- Model checking examines all interleavings of threads to determine if a concurrency bug can occur. Bugs that a model checker can find are race conditions, deadlocks, etc. A model checker can guarantee that a bug is found at the cost of long analysis times. In 2012 we have made progress in coding a general, language independent model checking framework. It is future work to optimize our new model checker by for example parallelization and algorithmic changes.

- To ensure that program design errors are caught early in the development cycle, it is necessary to find bugs while editing. This requires that any program analysis works at interactive speeds. We are following two approaches to this. The first approach addresses algorithmic changes to program analysis problems. Making analysis problems lazy means that only those parts of a program should be examined that are pertinent to the question that is currently asked by the compiler. For example, if the compiler needs to know which functions access a certain object, it should not examine unrelated functions, classes, or packages. Making program analysis incremental means that a small change in the program should only require small work for the (re-)analysis. In 2012 our new program analysis framework has been extended to find race conditions at compile-time and does so by performing partial lazy and incremental analysis. Our second approach to bring compiler analysis to interactive speed is to make the analysis itself parallel. In 2012 we have started to develop data-parallel formulations of the basic compiler analyses. The data-parallel analyses are then portably executable on different multi-core architectures. As a building block, we have started to implement a generic data-parallel predicate propagation framework that works on both multi-core CPUs and GPUs.

2.2 InThreaT - Inter-Thread Testing

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Norbert Oster
Start: 1.1.2012

Contact:
Dr.-Ing. Norbert Oster
Phone: +49-9131-85-28995
Fax: +49-9131-85-28809
E-Mail: norbert.oster@fau.de

In order to achieve higher computing performance, microprocessor manufacturers do not try to achieve faster clock speed anymore - on the contrary: the absolute number of cycles has even decreased, while the number of independent processing units (cores) per processor is continually increased. Due to this evolution, developers must learn to think outside the box: The only way to make their applications faster (in terms of efficiency) is to modularize their programs such that independent sections of code execute concurrently. Unfortunately, present-day systems have reached a level of functional complexity, such that even software for sequential execution is significantly error-prone - the parallelization for multiple cores adds yet another dimension to the non-functional complexity. Although research in the field of software engineering emerged several different quality assurance measures, there are still very few effective methods for testing concurrent applications, as the broad emergence of multi-core systems is relatively young.

This project aims to fill that gap by providing an automated test system. First of all, a testing criteria hierarchy is needed, which provides different coverage metrics tightly tailored to the concept of concurrency. Whilst for example branch coverage for sequential programs requires the execution of each program branch during the test (i.e. making the condition of an if-statements both true and false - even if there is no explicit else branch), a thorough test completion criterion for concurrent applications must demand for the systematic execution of all relevant thread interleavings (i.e. all possibly occurring orderings of statements, where two threads may modify a shared memory area). A testing criterion defines the properties of the ”final” test set only, but does not provide any support for identifying individual test cases. In contrast to testing sequentially executed code, test scenarios for parallel applications must also comprise control information for deterministically steering the execution of the TUT (Threads Under Test).

In 2012, a framework for Java has been developed, which automatically generates such control structures for TUT. The tester must provide the bytecode of his application only; further details such as source code or restrictions of the test scenario selection are optional. The approach uses aspect-oriented programming techniques to enclose memory access statements (reads or writes of variables, responsible for typical race conditions) with automatically generated advices. After weaving the aspects into the SUT (System Under Test), variable accesses are intercepted at runtime, the execution of the corresponding thread is halted until the desired test scenario is reached, and the conflicting threads are reactivated in the order imposed by the given test scenario. In order to demonstrate the functionality, some naive sequence control strategies were implemented, e.g. alternately granting access to shared variables from different threads.

2.3 OpenMP/Java

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
PD Dr. Ronald Veldema
Dipl.-Inf. Georg Dotzler
Dipl.-Inf. Thorsten Blaß
Duration: 1.10.2009–1.10.2015

JaMP is an implementation of the well-known OpenMP standard adapted for Java. JaMP allows one to program, for example, a parallel for loop or a barrier without resorting to low-level thread programming. For example:

```
class Test {
...void foo(){
......//#omp parallel for
......for (int i=0;i<N;i++) {
.........a[i]= b[i]+ c[i]
......}
...}
}
```

is valid JaMP code. JaMP currently supports all of OpenMP 2.0 with partial support for 3.0 features, e.g., the collapse clause. JaMP generates pure Java 1.5 code that runs on every JVM. It also translates parallel for loops to CUDA-enabled graphics cards for extra speed gains. If a particular loop is not CUDA-able, it is translated to a threaded version that uses the cores of a typical multi-core machine. JaMP also supports the use of multiple machines and compute accelerators to solve a single problem. This is achieved by means of two abstraction layers. The lower layer provides abstract compute devices that wrap around the actual CUDA GPUs, OpenCL GPUs, or multicore CPUs, wherever they might be in a cluster. The upper layer provides partitioned and replicated arrays. A partitioned array automatically partitions itself over the abstract compute devices and takes the individual accelerator speeds into account to achieve an equitable distribution. The JaMP compiler applies code-analysis to decide which type of abstract array to use for a specific Java array in the user's program.

In 2012, we extended the JaMP framework to also handle Java objects on multiple machines and accelerators (and not just arrays of primitive types). We added two different ways to handle objects. Standard shared objects are replicated on all compute devices. Arrays of objects are now also replicated or partitioned over the different devices. To increase the performance of the program, the framework has to break with Java's semantics. Java's object structure is mapped to a flat memory structure for the execution on the different devices.

2.4 PATESIA - Parallelization techniques for embedded systems in automation

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Stefan Kempf
Dipl.-Inf. Georg Dotzler
Dipl.-Inf. Thorsten Blaß
Start: 1.6.2009
Sponsored by:
ESI-Anwendungszentrum
Participating institutions:
ESI-Anwendungszentrum
Contact:
Dipl.-Inf. Stefan Kempf
Phone: +49-9131-85-27624
Fax: +49-9131-85-28809
E-Mail: stefan.kempf@fau.de

This project was launched in 2009 to address the refactorization and parallelization of applications used in the field of automation. The programs are executed on specially designed embedded systems. This hardware forms an industry standard and is used worldwide. As multicore-architectures are increasingly used in embedded systems, existing sequential software must be parallelized for these new architectures in order to gain an improvement of performance. As these programs are typically used in the industrial domain for the control of processes and factory automation they have a long life cycle. Because of this, the programs are often not being maintained by their original developers any more. Besides that, a lot of effort was spent to guarantee that the programs work reliably. For these reasons, the software is only extended in a very reluctant way. Therefore, a migration of these legacy applications to new hardware and a parallelization cannot be done manually, as it is too error prone. Thus, we need tools that perform these tasks automatically or aid the developer with the migration and parallelization.

To achieve this, this project works in three areas. The first part examines compiler analysis and programming language features needed so that in total, a novice programmer can effectively create solutions for real-time systems. For example, we have examined techniques to absolve the programmer from synchronization tasks and still allow low-level memory manipulations. To speedup compiler analysis, we have started investigations into accelerator use (GPUs) and algorithmic changes to get advanced compiler analysis fast enough for interactive use.

To work on the second part, a special compiler for the parallelization of existing automation programs was developed. First, we examined applications for automation on their automatic parallelizability. We found that these programs need to be refactored first, in order to be able to efficiently parallelize them automatically. For this reason, and as a second step, we extended the compiler to search for critical programming constructs that hinder parallelization, and to make the developer aware of them. The patterns we try to find are integrated into the compiler. This part is the origin of the migration tool described above, where the patterns can be described in a special language. The library of the runtime environment dynamically distributes parallel tasks that are created at runtime over the available cores. This mechanism was made more efficient by an internal reorganization, in order to evenly balance the work over all cores. Besides that, an implementation of transactional memory and a model for interrupt handling was added to the library as well. In 2012, the compiler was extended to automatically detect critical sections in parallel code. The mutual exclusion of the critical sections is implemented with Software Transactional Memory (STM). Moreover, STM optimizations were implemented. Previously, STM systems used a fixed-size table of locks to implement mutual exclusion of atomic blocks. This setup has shown to be a bottleneck at runtime. The optimizations scale the size of the table according to the workload of the application.

The third aspect focuses on optimization, refactorization and migration of legacy applications in order to render them operational on next-generation embedded systems. In this project we also develop a self-learning expert system. The goal of the system is to support developers in optimizing and refactoring their programs without forcing developers to specify the search patterns and transformations by hand. Instead the system extracts code transformations by comparing unmodified and modified source code and is able to apply these transformations to other programs. In a first step, parsers for different programming languages, C and Java amongst others, were developed. The parsers built a language independent abstract syntax tree (AST) from the different source codes. After that, a prototype was developed that finds code patterns suitable for transformations in an AST. If a pattern in an AST is found, the system generates source code that contains the applied transformation. The changed source code is presented to the developers as recommendation. For the recommendation generation, patterns from a database, extracted from the comparison of original and transformed source code, are used.

Annotations allow the generalization of the patterns. The annotations can access different plugins to be used even more flexible. The usefulness of the generated recommendations was demonstrated with source code of the Apache http server, the STAMP benchmark suite and the Java Grande benchmark suite. In the tests, extracted patterns were found in the source codes and suitable transformations for the occurrences were presented as recommendations.

In 2012 a transformation tool was added to the system. The goal of the transformation tool is the generation of general patterns from different code versions found in open-source version control systems. With a classification system, the tool recognizes irrelevant patterns and discards them. Only suitable patterns are saved in a database for later use.

Another part that was already finished in 2011 examines the migration of legacy applications to new hardware platforms. The challenge here is that the development environments for the new platforms do not support all language constructs that that have been available on the old hardware. This means that the programs need to be refactorized. Second, we examine how the applications can be automatically parallelized after the migration is performed. To guarantee an efficient parallelization, the programs may need to be further restructured. This also required work on the requirements of a runtime environment for parallel programs. We also perform research in a third area that attempts to combine the first two aspects in an expert system that uses a learning approach. The goal is to create a general refactoring tool that can be used for both migration and parallelization. The development for the migration tool was started in the end of 2010. This subproject treats the problem of the automatic refactorization of code. The main element is a special description language that can be used to describe code patterns that must be refactored. This makes it easy to write rules for refactorings. A pattern is a sequence of statements that should be replaced by some other code sequence. Another possibility is to just inform the user about the occurrence of a pattern if the refactorization cannot be done automatically. The description language, which is still in its development phase, is designed to be easy to use and to be close to natural language. It is our goal to be able to describe complex patterns in a simple and easy to learn way. We started with the design of the language and the integration into a compiler for programming languages used in the field of automation. This compiler is being developed at our department.

Parts of the project are funded by the ”ESI-Anwendungszentrum”:
http://www.esi-anwendungszentrum.de/

2.5 Compiler-supported parallelization for multi-core architectures

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Tobias Werth
Start: 1.3.2007
Contact:
Dipl.-Inf. Tobias Werth
Phone: +49-9131-85-28865
Fax: +49-9131-85-28809
E-Mail: tobias.werth@fau.de

Several issues significantly retard the development of quicker and more efficient computer architectures. Traditional technologies can no longer contribute to offer more hardware speed. Basic problems are the divergent ratio of the latencies of memory access and CPU speeds as well as the heat and waste of energy caused by increasing clock rates.

Homogeneous and heterogeneous multi-core architectures were presented as a possible answer and offer enormous performance to the programmer. The decreasing clock rates help avoid most of the above problems, while the multiplied hardware can still deliver high performance since more arithmetic operations can be executed per time unit with less energy. Potentially, performance can increase even further by specialization of some hardware components. For example, often the latency problem is attacked with a multi-tiered memory hierarchy and lots of caches.

But there is no free lunch. It seems to be quite difficult to make multi-core architectures deliver their theoretically available performance into applications. Only with a lot of expertise in both the application domain and the specifics of the multi-core platform at hand and only with enough time to invest into tuning endeavors, one can make multi-core programs run fast.

From the point of view of a programming systems research group, there are - among others - the following open questions: What kind of support can a modern compiler offer to the programmer that develops applications for multi-core architectures? How much context knowledge is necessary in order to make reasonable decisions for parallelization? Which part of the available performance can be used by the programmer with a reasonable amount of effort without detailed knowledge about the features and quirks of the underlying architecture? Which tools are necessary for debugging and for finding bottlenecks in applications that run on multi-core architectures? How can they be designed?

It is the intention of this new research project to answer these questions for a restricted application domain. We have selected the Lattice-Boltzmann-Method (LBM) that is mostly used in computational fluid dynamics as our problem domain. Caused by its lattice structure and its manageable number of data dependencies between the single lattice points, it is comparatively straightforward how to parallelize it. Hence, our compiler research can focus on the above questions.

The heterogeneous CellBE architecture is selected as target architecture due its good performance on a single chip. It consists of a PowerPC core (PPU) and eight Synergistic Processing Units (SPUs), which can do computations in parallel. The programming model Cilk was further developed to allow a robust and efficient execution on the SPUs. We made it much easier to debug the execution while stealing functions from remote SPUs. Beside that, the source to source transformation was rewritten to produce code for both PPU and SPU in a simpler and more generic way.

In 2012, we focused on graphic cards (GPUs) as a second target architecture. GPUs offer a lot more performance than ordinary CPUs, however achieving peak performance may be difficult. For data parallel problems, the performance can be achieved using Cuda (NVidia) or OpenCL (AMD) relatively easy. However, it is much more difficult to port task parallel problems with reasonable performance to the GPU, which is one of the goals on our roadmap. Thus, we will design, implement and compare various load balancing algorithms. In 2012 we designed a first approach with hierarchical queues that is based on the idea of work donation.

2.6 Efficient Software Architectures for Distributed Event Processing Systems

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Christopher Mutschler
Duration: 15.11.2010–14.11.2013
Sponsored by:
Fraunhofer Institut für Integrierte Schaltungen

Localization Systems (also known as Realtime Location Systems, or RTLS) become more and more popular in industry sectors such as logistics, automation and many more. These systems provide valuable information about whereabouts of objects at runtime. Therefore, processes can be traced, analyzed and optimized. Besides the research activities at the core of localization systems (like resilience and interference-free location technologies or methods for highly accurate positioning), there emerge algorithms and techniques to identify meaningful information for further processing steps. We focus our research on automatic configuration methods for RTLSs as well as on the generation of dynamic moving models and techniques for event processing on position streams at runtime.

In 2011, we investigated whether events can be predicted after analyzing and learning event streams from the localization system at runtime. As a result, we are able to deduce models that represent the information buried in the event stream to predict future events.

In 2012 we developed several methods and techniques to process and detect events with low latency. Events (composite, complex) can be detected by a hierarchical aggregation of sub-events. Those sub-events can be detected by using several event detectors that only process a sub-information from the event stream. The complexity of the detection components is strongly reduced. Hence, event detection components are fully maintainable, and can use parallel or distributed cluster architectures more efficiently. It is now possible to detect important events within a few milliseconds.

The project is a contribution of the Programming Systems Group to the IZ ESI:
http://www.esi.uni-erlangen.de/.

2.7 Software Project Control Center - Prototypical implementation of a new tool for quality assurance during software development

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Josef Adersberger
Norbert Tausch, M. Eng.
Duration: 1.11.2009–31.12.2013
Sponsored by:
Bundesministerium für Wirtschaft und Technologie
Contact:
Prof. Dr. Michael Philippsen
Phone: +49-9131-85-27625
Fax: +49-9131-85-28809
E-Mail: michael.philippsen@fau.de

Modern software systems are getting increasingly complex with respect to functional, technical and organizational aspects. Thus, both the number of requirements per system and the degree of their interconnectivity constantly increase. Furthermore the technical parameters, e.g., for distribution and reliability are getting more complex and software is developed by teams that are not only spread around the globe and also suffer from increasing time pressure. Due to this, the functional, technical, and organizational control of software development projects is getting more difficult.

The ”Software Project Control Center” is a tool that helps the project leader, the software architect, the requirements engineer, or the head of development. Its purpose is to make all aspects of the development process transparent and thus to allow for better project control. To achieve transparence, the tool distills and gathers properties from all artifacts and correlations between them. It presents/visualizes this information in a way suitable for the individual needs of the users.

The Software Project Control Center unifies the access to relations between artifacts (traceability) and to their properties (metrics) within software development projects. Thus, their efficiency can be significantly increased. The artifacts, their relations, and related metrics are gathered and integrated in a central data store. This data can be analyzed and visualized, metrics can be computed, and rules can be checked.

For the Software Project Control Center project we cooperate with the QAware GmbH, Munich. The AIF ZIM program of the German Federal Ministry of Economics and Technology funded the first 30 months of the project.

The Software Project Control Center is divided into two subsystems: The integration pipeline that gathers traceability data and metrics from a variety of software engineering tools, and the analysis core, that allows to analyze the integrated data in a holistic way. Each subsystem is developed in a separate subproject.

The project partner QAware GmbH implements the integration pipeline. The first step was to define TraceML, a modeling language for traceability information in conjunction with metrics. The language contains a meta-model and a model library. TraceML allows to define customized traceability models in an efficient way. The integration pipeline is realized using TraceML as lingua franca in all processing steps: From the extraction of traceability information to its transformation and integrated representation. We used the Eclipse Modeling Framework to define the TraceML models on each meta-model level. Furthermore, we use the Modeling Workflow Engine for model transformations and Eclipse CDO as our model repository. A set of wide-spread tools for software engineering are connected to the integration pipeline including Subversion, Eclipse, Jira, Enterprise Architect and Maven.

The main research contribution of our group to this project is the analysis core, i.e., the design and realization of a domain-specific language for graph-based traceability analysis. Our Traceability Query Language (TracQL) significantly reduces the effort that is necessary to implement traceability analyses. This is crucial for both industry and the research community as lack of expressiveness and inefficient runtimes of other known approaches hinder the implementation of traceability analysis. TracQL eases not only the extraction, but also the analysis of traceability data using graph traversals that are denoted in a concise functional programming style. The language itself is built on top of Scala, a multi-paradigm programming language, and was successfully applied to several real-world industrial projects.

2.8 Integrated Tool Chain for Meta-model-based Process Modeling and Execution

Project manager:
Hon.-Prof. Dr.-Ing. Detlef Kips
Project participants:
Dipl.-Inf. Ralf Ellner
Prof. Dr. Michael Philippsen
Dr.-Ing. Martin Jung
Dipl.-Inf. Johannes Drexler
Dipl.-Inf. Samir Al-Hilank
Duration: 1.10.2008–31.12.2012
Sponsored by:
BMWi

As demands on the development of complex software systems are continuously increasing, compliance with well-defined software development processes becomes even more important. Especially large and globally distributed software development projects tend to require long-running and dynamically changeable processes spanning multiple organizations. In order to describe and support such processes, there is a strong need for suitable process modeling languages and for powerful support by tools.

The results of a preceding cooperation project show that today's tools markets lack integrated tool chains which actually support the fine-grained and precise modeling of software development processes as well as their computer-aided execution, controlling and monitoring. A cooperation project has bridged this gap. This cooperation project was carried out together with develop group as an industrial partner and was funded by BMWi. It started in October 2008 and has been scheduled for three researchers. The project was finished in September 2011.

The objective of this cooperation project was to prototype an integrated tool chain by using a rigorous, meta-model based approach that supports modeling, enactment, and execution of industrial software development processes. Bearing the applicability of such a tool in mind, this approach was mainly intended to provide a wide adaptability of process models to different industrial development scenarios, to define a user-friendly concept of process description and to establish an extensive computer-aided process execution support, contributing to the efficiency of development activities. These benefits were achieved by a high grade of formalism, by an integrated, generic concept of process modeling and process enactment and by using commonly accepted industrial standards (UML, SPEM).

The integrated tool chain developed in this project is based on an extension of the SPEM standard (eSPEM – enactable SPEM). eSPEM adds a behaviour modeling concept by reusing UML activity and state machine diagrams. In addition, eSPEM provides behaviour modeling concepts that are specific to software development processes, for example, dynamic task creation and scheduling.

In 2012, an overview of the tool chain and eSPEM has been presented at the "First Workshop on Academics Modeling with Eclipse" which was held in conjunction with the "8th European Conference on Modeling Foundations and Applications". In addition, practical experiences from modeling SDPs in industrial projects have shown a rising importance of standards and reference models which are subsequently summarized under the term quality standard. These quality standards are used to specify requirements for target-oriented and effective execution of software development projects. These requirements are thereby defined to address different goals related to e.g. quality and efficiency (Automotive SPICE, CMMI) or safety (ISO 26262 Road Vehicles – Functional Safety) aspects of SWDPMs (Software Development Process Models). In other words, these requirements – often described in terms of best practices – are imposed on the software process definition that is typically described by SWDPMs. Tracing these requirements to the process definition is a precondition for supporting efficient assessment activities and process improvement projects. An additional goal of this research project lies therefore in the integration of these quality standards with SWDPMs with a special focus on environments that requires conformance to more than one quality standard (e.g. CMMI, Automotive SPICE and ISO 26262).

2.9 Embedded Systems Institute

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Stefan Kempf
Dipl.-Inf. Georg Dotzler
Dipl.-Inf. Thorsten Blaß
Dipl.-Inf. Tobias Werth
Dipl.-Inf. Christopher Mutschler

Andreas Kumlehn, M. Sc.
Dr.-Ing. Norbert Oster
Start: 1.9.2007

In September 2007 the ESI – Embedded System Institute – was founded as an interdisciplinary center at the Friedrich-Alexander-University (FAU) with the goal to coordinate and organize research, teaching, and further education in the field of embedded systems.

ESI brings together existing skills within the university and interests, activities, and goals of large and medium sized companies in the field of embedded systems.

Companies obtain access to latest research results and the opportunity to develop common projects, to establish ties, and to find co-operation partners. The ESI concentrates the skills of the chairs of computer sciences and makes them usable for co-operation projects. Hence, the latest research results can be transferred into products in a speedy way. Finally, the ESI may serve as a platform for recruiting excellent students and highly qualified young academics at an early stage.

The chair of computer science department 2 (Prof. Philippsen) is one of the active founders of the ESI and carries out research projects within the ESI.

More information can be found at http://www.esi.uni-erlangen.de and http://www.esi-anwendungszentrum.de

2.10 Graphs and Graph Transformations

Project manager:
Prof. em. Dr. Hans Jürgen Schneider
Start: 1.10.2004
Contact:
Prof. em. Dr. Hans Jürgen Schneider
Phone: +49-9131-85-27620
Fax: +49-9131-85-28809
E-Mail: hans.juergen.schneider@fau.de

Graphs are often used as an intuitive aid for the clarification of complex matters. Examples of outside computer science include, e.g., chemistry where molecules are modeled in a graphical way. In computer science, data or control flow charts are often used as well as entity relationship charts or Petri-nets to visualize software or hardware architectures. Graph grammars and graph transformations combine ideas from the fields of graph theory, algebra, logic, and category theory, to formally describe changes in graphs.

Category theory is an attractive tool for the description of different structures in a uniform way, e.g., the different models for asynchronous processes: Petri-Nets are based on standard labeled graphs, state charts use hierarchical graphs, parallel logic programming can be interpreted in a graph-theoretical way using so-called jungles, and the actor systems can be visualized as graphs, whose labeling alphabet is a set of term graphs.

In 2012, we have concentrated our attention on a theoretical aspect.

Our work on graph transformation is based on notions borrowed from category theory. The so-called double-pushout approach represents a production by two morphisms starting at a common interface graph. One pushout glues the left-hand side of the production into the context, the other does with the right-hand side. Effectively constructing a derivation step, however, requires finding a pushout complement on the left-hand side. Some people consider this disadvantageous. In 1984, Raoult has proposed to model graph rewriting by a single pushout; Loewe has extensively studied this approach, but the discussion was mainly restricted to injective morphisms. Under this assumption, the approaches are equivalent. Some relevant applications such as term graph rewriting, however, lead to non-injective morphisms. We have examined these cases in detail, and we could show that the equivalence also holds for non-injective cases as long as the handle satisfies some reasonable conditions.

2.11 International Collegiate Programming Contest at the FAU

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Tobias Werth
Dipl.-Inf. Daniel Brinkers
Dipl.-Math. Jakob Krainz
Start: 1.11.2002
Contact:
Dipl.-Inf. Tobias Werth
Phone: +49-9131-85-28865
Fax: +49-9131-85-28809
E-Mail: tobias.werth@fau.de

The Association for Computing Machinery (ACM) has been hosting the International Collegiate Programming Contest (ICPC) for many years. Teams of three students try to solve nine to eleven programming problems within five hours. What makes this task even harder, is that there is only one computer available per team. The problems demand for solid knowledge of algorithms from all areas of computer science and mathematics, e.g., graphs, combinatorics, strings, algebra, and geometry.

The ICPC consists of three rounds. First, each participating university hosts a local contest to find the three teams that are afterwards competing in one of the various regional contests. Erlangen is situated within the bounds of the Northwestern European Regional Contest (NWERC) where also teams from Great Britain, Benelux and Scandinavia compete. The winners of all the world's regional contests (and some second place holders) advance to the world finals in spring of the following year.

In 2012 two local contests took place in Erlangen. During the winter semester we conducted a team contest with teams consisting of at most three students. The main goal of this contest was to arouse interest of new students in the contests. There has been a record attendance of 25 FAU teams. Also 40 teams from universities from all over Europe participated.

As in the previous years, in the summer term the seminar "Hello World - Programming for Advancers" served to prepare students from different disciplines in algorithms and contest problems. As a result of the summer contest, which was organized as a national German contest for the first time, we selected the representatives of the FAU that later participated in the NWERC 2012 in Delft. 21 teams with students of Computer Science, Computational Engineering, Mathematics as well as Information and Communication Technology took part in the contest. Ten students were selected for the NWERC forming three teams and one reserve. At the NWERC in Delft, our best team solved six problems but missed a medal by just one place. The second team also did a great job solving five problems. With a rank in the middle (38th of 83 teams) for the third team, the NWERC was a great success for the FAU. Again, the training camp served as a perfect way of extra preparation for the student competitors.

3 Publications 2012

– Braunstein, Sören ; Härdtlein, Jochen ; Philippsen, Michael: Expressing Parallelism and Timing in Embedded Real-Time Applications . In: High-Performance and Embedded Architecture and Compilation (HiPEAC) Network of Excellence (Org.) : 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES) 2012 - Poster Abstracts (8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems Fiuggi, Italy July 11, 2012). Ghent (Belgium) : Academia Press, 2012, pp 197-200. - ISBN 978-90-382-1987-5

– Dotzler, Georg ; Veldema, Ronald ; Philippsen, Michael: Annotation Support for Generic Patches . In: Maalej, Walid ; Robillard, Martin ; Walker, Robert J. ; Zim-

mermann, Thomas (Ed.) : Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering (RSSE 12) (International Workshop on Recommendation Systems for Software Engineering Zurich, Switzerland 04.06.2012). 2012, pp 6-10. - ISBN 978-1-4673-1758-0

– Ellner, Ralf ; Al-Hilank, Samir ; Jung, Martin ; Kips, Detlef ; Philippsen, Michael: An Integrated Tool Chain for Software Process Modeling and Execution. In: Störrle, Harald ; Botterweck, Goetz ; Bourdellès, Michel ; Kolovos, Dimitris; Paige, Richard ; Roubtsova, Ella ; Rubin, Julia ; Tolvanen, Juha-Pekka (Ed.): Joint Proceedings of co-located Events at the 8th European Conference on Modeling Foundations and Applications (ECMFA 2012) (8th European Conference on Modeling Foundations and Applications (ECMFA 2012) Lyngby, Denmark 02.-05.07.2012). 2012, pp 73-82. - ISBN 978-87-643-1014-6

– Mutschler, Christopher ; Philippsen, Michael: Apparatus, Method and Computer Program for Migrating an Event Detector Process . Schutzrecht PCT/EP2011/069159 patent application (14.03.2012)

– Mutschler, Christopher ; Philippsen, Michael: Learning Event Detection Rules with Noise Hidden Markov Models . In: Benkrid, Khaled ; Merodio, David (Ed.): Proceedings of the 2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2012) (2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2012) Nuremberg, Germany 25.06.2012). 2012, pp 159-166. - ISBN 978-1-4673-1914-0

– Mutschler, Christopher ; Philippsen, Michael: Towards a Distributed Self-Optimizing Event Processing System for Realtime Locating Systems (RTLS). In: ACM (Ed.) : DEBS PhD Workshops, 6th ACM International Conference on Distributed Event-Based Systems (International Conference on Distributed Event-Based Systems (DEBS) Berlin, Germany July 16-20, 2012). 2012, pp -.

– Mutschler, Christopher ; Loeffler, Christoffer: Vorrichtung, Verfahren und Computerprogramm zum Verbessern einer Leistungsfähigkeit eines ereignisbasierten verteilten Analysesystems . Schutzrecht DE 10 2012 112 253.9 patent application (13.12.2012)

– Otto, Stephan ; Bretz, Ingmar ; Franke, Norbert ; von der Grün, Thomas ; Mutschler, Christopher: Vorrichtung, Verfahren und Computerprogramm zur Rekonstruktion einer Bewegung eines Objekts . Schutzrecht 10 2012 111 304.1 DE patent application (22.11.2012)

– Pankratius, Victor ; Philippsen, Michael (Ed.): Multicore Software Engineering, Performance and Tools (Proceedings MSEPT 2012) . (International Conference on Multicore Software Engineering, Performance, and Tools (MSEPT 2012) Prague, Czech Republic 31.05. - 01.06.2012) Berlin Heidelberg : Springer, 2012 (Lecture Notes in Computer Science (LNCS) Vol. 7303) . - 95 pages. ISBN 978-3-642-31201-4. ISSN 0302-9743

– Tausch, Norbert ; Philippsen, Michael ; Adersberger, Josef: TracQL: A Domain-Specific Language for Traceability Analysis . In: Ali Babar, M. ; Cuesta, C. ; Savolainen, J. ; Männistö, T. (Ed.) : Proceedings of the 2012 Joint Working Conference on Software Architecture & 6th European Conference on Software Architecture (Joint Working Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA 2012) Helsinki, Finland 20. - 24.08.2012). Los Alamitos, CA : IEEE CPS, 2012, pp 320-325. - ISBN 978-0-7695-4827-2

– Veldema, Ronald ; Philippsen, Michael: Parallel Memory Defragmentation on a GPU . In: Zhang, Lixin ; Mutlu, Onur (Ed.) : Proceedings of the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness (ACM SIGPLAN Workshop on Memory Systems Performance and Correctness (MSPC 12) Beijing, China 16.06.2012). 2012, pp 38-47. - ISBN 978-1-4503-1219-6

4 Exam theses (german only)

– Studienarbeit: Realisierung der konkreten Syntax von ”DeepML” mit dem Graphiti-Framework. Bearbeiter: Elena Klevtsova (beendet am 01.06.2012); Betreuer: Dipl.-Inf. Ralf Ellner; Hon.-Prof. Dr.-Ing. Detlef Kips

– Master Thesis: Blending Agile and Traditional Project Management for Safety Critical Systems Engineering. Bearbeiter: Tobias Freitag (beendet am 09.08.2012); Betreuer: Hon.-Prof. Dr.-Ing. Bernd Hindel

– Bachelor Thesis: Programmierung einer Mini-Anlage mit parallelen Programmierkonstrukten. Bearbeiter: Martin Sturm (beendet am 28.08.2012); Betreuer: PD Dr. Ronald Veldema; Dipl.-Inf. Stefan Kempf; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Anpassung eines Expertensystems zur Programmanalyse und -refaktorisierung für Anwendungen aus der Automatisierungstechnik. Bearbeiter: Johannes Wellhöfer (beendet am 03.09.2012); Betreuer: Dipl.-Inf. Georg Dotzler; Dipl.-Inf. Stefan Kempf; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Evaluation von Algorithmen zur Positionsdatenkompression. Bearbeiter: Jonathan Reuss (beendet am 2.10.2012); Betreuer: Dipl.-Inf. Christopher Mutschler; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Implementierung eines JaMP-Übersetzers zur Generierung von Java/OpenCL Code. Bearbeiter: Florian Habur (beendet am 02.10.2012); Betreuer: Dipl.-Inf. Georg Dotzler; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Netzwerksimulation- und -optimierung von verteilten Ereignisverarbeitungssystemen. Bearbeiter: Löffler Christoffer (beendet am 05.11.2012); Betreuer: Dipl.-Inf. Christopher Mutschler; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Entwicklung eines Werkzeugs zur Extraktion von Mustern aus Software-Archiven zur Quellcode-Optimierung. Bearbeiter: Marius Kamp (beendet am 06.11.2012); Betreuer: Dipl.-Inf. Georg Dotzler; Prof. Dr. Michael Philippsen

- Diplomarbeit: Konzeption und prototypische Realisierung einer Abfrage- und Transformationssprache für DeepML. Bearbeiter: Elena Klevtsova (beendet am 09.11.2012); Betreuer: Dipl.-Inf. Samir Al-Hilank; Dipl.-Inf. Ralf Ellner; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Intelligent Documentation Generation. Bearbeiter: Teodor Shaterov (beendet am 27.11.2012); Betreuer: PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen

- Master Thesis: Transparent use of Java objects on the GPU in the JaMP/OpenMP framework. Bearbeiter: Carolin Wolf (beendet am 06.12.2012); Betreuer: Dipl.-Inf. Georg Dotzler; Prof. Dr. Michael Philippsen