Topics for Student Theses

Hier eine Themenübersicht, gefolgt von etwas detaillierteren Darstellungen. Generell sind Themenvariationen möglich, und auch selbst definierte Themen aus dem Bereich Echtzeitsysteme/Eingebettete Systeme können gerne besprochen werden. Ein weiterer Weg, um in kompakter Form einen Einblick in aktuelle Themen der Arbeitsgruppe zu bekommen, ist die Teilnahme an dem regelmäßig zu Semesterende bzw. in der vorlesungsfreien Zeit angebotenen Oberseminar.

Hinweis: Es ist Studierenden ausdrücklich empfohlen, sich frühzeitig bei den verschiedenen Arbeitsgruppen über mögliche Themen der Abschlussarbeit zu informieren. WWW-Seiten wie diese hier sind ein guter erster Anlaufpunkt, und es ist eine gute Idee, sich vor einem Gespräch mit einem potenziellen Betreuer (Professor, Assistenten -- generell die Dozenten von Lehrveranstaltungen) über mögliche Themen einen Blick auf diese Seiten zu werfen. Es ist jedoch erfahrungsgemäß schwierig, auf solchen Seiten vollständige und aktuelle Informationen bereitzustellen; sie sollten daher eher als grober Indikator der jeweils möglichen Themenfelder dienen denn als konkrete Ausschreibungen. Um zu erfahren, welche Themen konkret verfügbar sind, zu dem angestrebten Zeitrahmen, sollte man auf jeden Fall die Dozenten konsultieren.

Die möglichen Themen sind im Folgenden thematisch gruppiert. Die Zahlen vor der Themenbeschreibung stehen für Prioritäten. Je kleiner die Zahl, desto wichtiger ist uns das Thema.

Outline

Automatic Graph Layout

Advisors: Christoph Daniel Schulze, Reinhard von Hanxleden.

Ein sehr wichtiges Gebiet für uns ist das automatische Layout von Diagrammen. Hierfür gibt es bereits Werkzeuge, die gute Algorithmen enthalten, so dass viele Diagramme bereits jetzt übersichtlich und automatisiert angeordnet werden können (siehe z.B. Graphviz). Für einige besondere Arten von Diagrammen sind diese allgemeinen Algorithmen jedoch nicht geeignet, da zusätzliche Anforderungen an das Layout erfüllt werden müssen. Außerdem ist häufig die technische Anbindung vorhandener Algorithmen umständlich. Nutzer müssen sich mit der Funktionsweise der Algorithmen beschäftigen, um sie für ihre Anwendung optimal konfigurieren zu können.

Wir verfolgen drei Themenbereiche, die zusammen solche Probleme lösen und den Nutzen von automatischem Layout erhöhen sollen:

» Algorithmen-Entwicklung. Wir implementieren vorhandene Ansätze zum Layout von Graphen in Java und binden sie in unser Projekt ein. Der Schwerpunkt liegt auf dem Entwurf von Erweiterungen, die spezielle Anforderungen unterstützen, z.B. für Datenfluss-Diagramme. Dies ist gut für alle geeignet, die sich gerne mit Graphentheorie, effizienten Algorithmen oder kombinatorischer Optimierung beschäftigen.

» Dienste. Algorithmen und Meta Layout müssen den Anwendern zugänglich gemacht werden, damit ein Nutzen daraus entsteht. Dazu müssen wir verschiedenste graphische Frameworks mit vorhandenen Layout-Bibliotheken integrieren und eine Reihe von Werkzeugen entwickeln, mit denen die Verfügbarkeit unserer Lösungen gesteigert wird. Hierzu gehört z.B. die Unterstützung von Standard-Graphenformaten sowie ein Web-Service für automatisches Layout.

Die Entwicklung geschieht im Eclipse Layout Kernel-Projekt (kurz ELK), einem offiziellen Eclipse-Projekt welches hauptsächlich wir betreuen und weiter entwickeln. Ergebnisse in diesem Bereich fließen damit einer tatsächlich existierenden Nutzerbasis zu.

Modeling Pragmatics

Advisors: Christoph Daniel Schulze, Reinhard von Hanxleden

- **Post-Processing Label Placement with Label Management** (Bachelor, Master)
  This is about implementing a stand-alone label placement algorithm that can place node and edge labels after everything else has already been placed. Since there might not be enough space to place all labels, the algorithm should provide different options of coping with such situations. One would be to hide such labels, another one would be to apply label management to them.
- **Standalone Edge Routing** (Master)
- **Compound Graph Exploration** (Bachelor, Master)
  A new graph exploration approach should be examined which is uses different zoom levels for different compound nodes. This tries to map the "Google Maps approach" of only showing the information of interest at any given zoom level to the field of graph exploration.
- **Improvements to Spline Edge Routing** (Bachelor, Master)
  Spline edge routing closely follows the routes orthogonal edges would take. A Bachelor's thesis could work on improving how splines connect to their end points to make the results look more natural. A Master's thesis could look at improving the routes splines take through a diagram more generally.
- **Interactivity for Further Diagram Elements and Layout Algorithms** (Bachelor, Master)
- **Relative Interactivity Constraints** (Bachelor, Master)
- **Polishing and Evaluating Interactive User Experiences** (Bachelor, Master)
- **Interaction Techniques for Large Diagrams** (Bachelor, Master)
- **Control Flow Graph Exploration / Visualization** (Bachelor)
  Use pragmatics concepts (automatic layout, focus & context) for exploring/visualizing control flow graphs and specific paths, eg. as computed by OTAWA WCET analysis tool, eg. using KLighD.

Further possible thesis topics can be found in ELK's GitHub repository. Note, however, that some issues there may already be worked on.

Semantics, Synchronous Languages and Model-based Design

Advisors: Steven Smyth, Alexander Schulz-Rosengarten, Reinhard v. Hanxleden

Synchronous languages are well-established for the design of embedded, in particular safety-critical systems. One of our research areas concerns the further development of such languages and their efficient compilation. Specifically, we explore the paradigm of "sequential constructiveness" for reconciling familiar, imperative programming concepts with the sound grounding of synchronous languages. One language we have developed to try out and validate our concepts is the SCCharts language, which keeps evolving and thus offers many opportunities for student theses.

SCCharts Code Generation & Optimizations

- **Optimization of the SCCharts compiler/transformations** (Bachelor/Master)
  Profile the actual SCCharts compiler/transformations and apply optimizations; also evaluate the possibility to use multiple cores for compilation.
- **Efficient data dependency & scheduling analyses in SCCharts** (Master/Bachelor)
  Implement analyses for data dependencies and scheduling (e.g. tick boundaries) for SCCharts to improve static scheduling of the compiler.
- **Javascript code generation** (Bachelor/Master)
  Implement a javascript code generation for SCCharts. Integrate with simulation and (environment) visualization to deploy a complete example as standalone web page. Compare with HipHop.js based on Esterel.

SCCharts Simulation

- **Visualization of Model-based Simulation via Tracing** (Bachelor/Master)
  Use the already implemented Model-to-Model-Tracing in KIELER to visualize simulations.
- **Core SCCharts Interpreter with dynamic Scheduling** (Master/Bachelor)
  Implement an Interpreter for Core SCCharts that supports SC Policies.
- **Live Debugging of Statecharts** (Master/Bachelor)
  Implement a dedicated debugging view for SCCharts.
- **Microstep-Simulation of Statecharts** (Master/Bachelor)
  Implement a method to do microstep simulation with SCCharts.

Model-based C Code Compilation

- **Execution of Recursive Dataflow Code** (Master/Bachelor)
- **Execution of Concurrent Dataflow Code** (Master/Bachelor)
  Modify the model-based dataflow compiler in KIELER so that it is able to compile recursive/concurrent C programs.
  For Master students: Implement both.