Design-Space Exploration with FORMULA

Markus Dahlweid, René Hülswitt, Thomas Santen, Dirk Seifert
Ethan Jackson, Nikolaj Bjørner, Wolfram Schulte
Automotive Embedded Software

• More than 60 *Embedded Control Units* (ECUs) in a car
  – head unit, engine, breaks, gyro, seats, doors, lights, ...
  – connected by several busses (CAN, FlexRay, ...)
• ECUs become increasingly powerful

➢ Complex distributed system with challenging requirements
  ➢ functional correctness, hard real-time, energy efficiency, safety and security, resource limitations: memory, bandwidth

➢ Compositional design is hard to achieve
  ➢ requirements relate to emergent properties
  ➢ complex interaction of different kinds of requirements

High pressure for further consolidation
• integrate more SW functionality on fewer ECUs to provide space for new features
  ➢ increasing complexity of interactions on single ECU and between ECUs

➢ Similar observations also apply for other industry domains
Engineering Embedded Systems

Typical tasks in a development process

1. System architecture and design of software and hardware components
2. Development of software components
3. Mapping of components to ECUs
4. Specification of communication matrix
5. Mapping of runnable entities to tasks
6. Configure the OS including scheduling
7. Configure the basic SW stack

Verification often only late in the process by simulation or testing

- Provide **early feedback** about potential problems to save time and budget
- Assist in finding **first-time-right** solutions
For each software component, determine the hardware node on which to deploy it to.

- specified before the components are built
- crucial for overall functionality and performance
- verified only at system integration time
- complexity requires sophisticated tool support
FORMULA Introduction
Exploring architectures

Abstraction: Tasks must be deployed onto a capacity-balanced network respecting conflicts.

Instance of Functionality

Must find Mapping of Tasks to Nodes that respects Conflicts

Instance of Network
A new FORMULA for Modeling

Provide a general language for capturing model-based abstractions, and support automated model synthesis in any direction.

Core formal specification language  
(CLIP with negation over regular types)

Module system for composing specs and crossing abstraction boundaries

Formal descriptions of design spaces and reachability problems

Formalize domain-specific abstractions
Compose specifications
Model synthesis and design-space exploration
Formally specify complex integration problems using concise logic programming. FORMULA provides recursive data structures, arbitrary unions and order-sorted base types.

Abstraction:
There are things called Task that can be in Conflict

Unit of abstraction
Data type constructors with labeled arguments
Predefined annotations reduce size of specification

domain Functionality
{
    Task ::= (id: String).
    [Closed]
    Conflict ::= (t1: Task, t2: Task).
}
Formally specify complex integration problems using concise logic programming. FORMULA provides recursive data structures, arbitrary unions and order-sorted base types.

Abstraction:
There are things called Task that can be in Conflict

Concrete instance using entities of Functionality

Specify instances of type constructors

Use aliases for convenience

```latex
model Instance of Functionality
{
  t1 is Task("UI")
  t2 is Task("Voice Rec")
  t3 is Task("Network Controller")
  Conflict(t1, t2) Conflict(t2, t3)
}
```
Abstraction: There are nodes connected by channels with capacities.

Nodes with different properties can execute functionality.

Nodes have at most two incoming and two outgoing Channels.

Capacities of Channels must be balanced.

Partial instance of abstraction:
We may use the specified processors, the channels are undefined.
Logic programs provide a formal and declarative approach for specifying abstractions. Queries encode Boolean conditions and rules perform complex computation.

```plaintext
(Queries)  query-name  :=  pattern-expression.
(Rules)    f(t_1,t_2,...,t_n)  :=  pattern-expression.
```

Logic Programming

Annotations introduce constraints

Implicit typing for succinct rules

Constraints as Boolean conditions

Aggregation supported

Defining the rules of the abstraction

---

```plaintext
domain  Network
{
    Node  ::=  (id: String).
    [Closed] [Unique(from, to -> capacity)]
    Channel  ::=  (from: Node, to: Node, capacity: PosInteger).

    // Ensure that capacities are balanced
    mustBal(n)  :=  Channel(n,_,_), Channel(_,n,_,).
    clog  :=  mustBal(n),
            sum(Channel(_,n,_,),2)  !=  sum(Channel(n,_,_),2).

    // Ensure each node has 2 input/output channels at max
    bigFanIn  :=  n is Node, count(Channel(_,n,_,))  >  2.
    bigFanOut  :=  n is Node, count(Channel(n,_,_,))  >  2.

    conforms  :=  !clog & !bigFanIn & !bigFanOut.
}
```
Exploring architectures

Abstraction: Tasks must be deployed onto a capacity-balanced network respecting conflicts.

Instance of Functionality

Must find Mapping of Tasks to Nodes that respects Conflicts

Instance of Network
Exploring the Design-Space

Use FORMULA to automatically explore the space of candidate architectures by enumerating many different solutions to the deployment problem.

Integration with Visual Studio 2010 supports inspection and automated visualization of solutions.

- Self-referencing channels
- Unconnected nodes
• Software architecture and platform are designed using model based methods
• Domain experts provide annotations for extra-functional properties

What to consider?
  – Structural constraints
  – Timing constraints
  – Resource constraints
  – Energy consumption constraints
  – ...

Challenge:
Find solutions that satisfy all types of constraints **simultaneously**
**Facet**: View on a system regarding a class of extra-functional properties
Core Solution Space

Memory Facet

Timing Facet

Safety Facet

Over-approximation of valid solutions

Hard to find without simultaneous consideration of facets
Summary

• FORMULA is a formal specification language targeting model-based development
• FORMULA is based on algebraic data types and constraint logic programming
• FORMULA supports model finding and design-space exploration
• FORMULA allows independent description but simultaneous reasoning about facet properties

Try it:  http://www.rise4fun.com/formula
Get it:  http://research.microsoft.com/formula/
Thank you!

http://research.microsoft.com/formula/

Markus Dahlweid  Ethan Jackson  Dirk Seifert

Contact us: formula@microsoft.com