CONSONA: Constraint Networks for the Synthesis of Networked Applications

Lambert Meertens
Cordell Green

Kestrel Institute

Palo Alto, California
http://consona.kestrel.edu/

NEST Kickoff Meeting, Napa, CA, June 5–7, 2001

The CONSONA team

Lambert Meertens	co-PI
Cordell Green	co-PI
Doug Smith	research scientist
Stephen Westfold	research scientist

Relevant Kestrel technology

- Provably correct refinement from high-level specs to executable code (Specware)
- Generator for highly optimized off-line schedulers based on (hard) constraint-propagation compilation (Planware)
- Design taxonomies (Designware)
- Anytime scheduling based on soft constraints (DARPA ANTs program)

Problems specific to creating NEST services & applications

- In large-scale distributed fine-grained systems the "state space" lacks manageable structure
  - ➡ traditional methods for developing distributed applications are not suited to handling the requirements of the NEST program
- IPC stacks (for example) do not exploit application-level properties to boost performance
  - ➡ NEST applications built the traditional way on top of a *pre-compiled layer* of middleware services will incur heavy performance overhead penalties

Aim of the CONSONA project

- Develop model-based methods and tools for the *goal-oriented* integrated design and synthesis of NEST applications and services
  - Use system-wide constraints to specify ⟨what is to be *achieved*⟩, **not** ⟨what is to be *done*⟩
  - Iteratively match constraint requirements to middleware service (coordination) *schemas* and instantiate to *refine* the design, expressed as a constraint network
  - Generate optimized code from such constraint-network models using constraint maintenance and propagation

Example: UAV swarm

- **Assumptions:**
  - **UAVs communicate through wireless broadcasts**
  - **range is limited (scalability!)**
  - **signal strength can be used to estimate distance**

Example problem requirements

- Safety requirements:
  – *vehicles must maintain safe distance*
- Progress requirements:
  – *patrol given area*
  – *collect information timely*
- "Non-functional" requirements:
  – *minimize energy expenditure*

(The example happens to be homogeneous, but that is not essential to the approach)

Example requirement:

▸ *Maintain safe distance*

- *System-wide* constraint: Projected flight paths (“cones” with increasing uncertainty) don’t intersect – a tough problem when time is of the essence
- This constraint can be maintained by adjusting the flight paths
  - ➡ requires maintaining knowledge of relative positions, velocities, …
  - ➡ which is a *newly introduced* requirement!

Newly introduced requirement:

▸ *Maintain knowledge of relative positions*

- Constraint *network*:
  - Each UAV has a map of some other UAVs' positions
  - Each UAV's map must be consistent with observed signal strengths
- Constraint can be maintained by adjusting estimated positions
- But doing this just locally is bound to create inconsistencies between the various maps
  - ➡ yet another *introduced requirement*: an instance of the general requirement of *consistency in distributed knowledge*!

Now the newest requirement:

- *Maintain ‘inter-map’ consistency*

- Constraint *propagation*: knowledge maintained by proximate nodes must be compatible
  - – UAV maps must agree on overlap to within some tolerance/latency
- This kind of constraint can be maintained by comparing and reconciling knowledge, in this case the maps
- In general: need to confine this to “relevant” knowledge

Roadmap

- We expressed the application requirements as *system-wide* constraints
- We decomposed these constraints into a constraint *network* (basically a conjunction of “local” constraints)
- We refined the constraints using *applicable schemas*,
- which identify *constraint-maintenance* methods, expressible as *symbolic code*
- *Actual code* can be generated as “residual code” after symbolic constraint propagation and simplification

Refinement

Set of constraints:

$\{\ldots, P, \ldots\}$

$\downarrow$

$\{\ldots, S\theta, \ldots\}$

the refinement

Applicable schema:

$R \leftarrow S$

where

$R\theta = P$

for unifier θ

Technical Approach

- Model requirements as *soft constraints*
  - – better suited to real-time, distributed systems: hard constraints lead to intractability
- Identify applicable constraint schemas (patterns) suited to *distributed maintenance*
- Use *model-based transformations* for high-level optimization
  - – e.g., flattening middleware layers
- Use symbolic *constraint propagation* for optimized code generation

Claims

- Modeling method is amenable to *composition* and *parameterization*
  - keyword: *modular*
- Soft constraints can model *resource aggregation* and *dynamic selection* of task-execution strategies
  - keyword: *adaptive*
- They are particularly suited for obtaining “graceful degradation” in case of *physical malfunction* or *task overload*
  - keyword: *robust*

Project tasks (highlights)

- Modeling using constraints :
  – basic protocols and algorithms
  – increasingly complex applications
  – composition and parameterization
- Constraint technology:
  – analysis/propagation for soft constraints
- Toolset:
  – modeler
  – knowledge base of middleware schemas
  – constraint-solver generator

Main deliverables

- Modeling using constraints:
  – Models of basic protocols and algorithms (December 2001)
  – Suite of coordination services (September 2003)
- Constraint technology:
  – Solver-driven integration of services for OEP architecture(s) (June 2002)
- Toolset:
  – Preliminary design (June 2002)
  – Prototype modeling toolset (March 2003)
  – Prototype generator (June 2003)
  – Integrated modeler-generator (March 2004)

OEP integration

- The generator will target one or more OEP architectures
- All software will be installed at one or more OEP labs
- Demo of modeler and generator for large example NEST application on one or more OEPs

Further integration

- Choice of protocols/algorithms/services modeled will be inspired and informed by the needs and results of other groups in the NEST program
- We'll welcome and encourage others to experiment with the modeling toolset and generator