A Constraint Programming Approach for Managing End-to-end Requirements in Sensor Network Macroprogramming

Farshid Hassani Bijarbooneh       Animesh Pathak
Justin Pearson                Valerie Issarny
Bengt Jonsson

ASTRA and Mobility Research Groups
Department of Information Technology
Uppsala University, Sweden
http://www.it.uu.se/research/group/astra
http://www.it.uu.se/research/group/core
Project ARLES, INRIA Paris-Rocquencourt
Paris, France
http://www-rocq.inria.fr

Lisbon, Portugal, Jan 7 – 9, 2014
Previous Research

■ Energy-efficient task-mapping for data-driven sensor network macroprogramming using constraint programming. [ModRef 2010, ICS 2011]

■ An optimisation-based approach for wireless sensor deployment in mobile sensing environments. [WCNC 2012]

■ Macroprogramming of wireless sensor networks using task graphs and constraint solving. [SNCNW 2012]

■ Optimising quality of information in data collection for mobile sensor networks. [IWQoS 2013]

■ A constraint programming approach for managing end-to-end requirements in sensor network macroprogramming. [SensorNets 2014]
Outline

1. Previous Research
2. The Problem
3. Model and Example
4. Experiments
5. Conclusion
Application of the network are implemented in **abstract tasks**:

- An **instantiated task** is an exact copy of an abstract task:
  - To be mapped to the sensor nodes.
  - Belongs to a specific region on the network.
- A **region** is a physical location in the network (e.g., a floor in a building).
  - Sensor nodes are deployed in regions.
Application: Highway Traffic Management

Reduce the congestion of vehicles on a highway:

- Control speed limits.
- Control highway access.
Contributions and Highlights

- Extension of the macroprogramming ATaG semantics to support end-to-end requirements.

- The formulation of the task-mapping problem arising to satisfy end-to-end requirements.

- Using constraint programming (CP) to solve efficiently the problem.

- Extension of the approach to improve latency performance by replicating tasks.
Challenges and Motivation

- Programming of sensor tasks is very time consuming.
- **Data-driven macroprogramming:**
  Create a task graph based on the flow of data (ATaG), subject to placement and end-to-end requirements:
Goal and Motivation

- Include end-to-end requirements in the existing macroprogramming platform (ATaG), and solve them using constraint programming (CP).

- **Latency** is an important requirement on an end point:

  \[
  \text{Prob (latency between regions } r_i \text{ and } r_j \leq \text{maxDelay}) \geq \text{minProbability}
  \]

- **Objective:**
  Minimise the number of required replicated tasks to satisfy the end-to-end requirements
Methodology

An Optimal Mapping of Tasks to Nodes

Input

Constraint Programming Model

Output

Network Description
Instantiated Task Graph
Abstract Task Graph
Placement Constraints
Path Delay Constraints
Probability Model

CP\textsubscript{main}

CP\textsubscript{sub} on path 1
CP\textsubscript{sub} on path 2
\ldots
CP\textsubscript{sub} on path p
Mathematical Model (continued)

- Decision variables:
  
  \( \text{node}[t] \in N, \)
  
  the node assigned to task \( t \in T \)

- Placement constraints:
  
  \( \text{node}[t] \neq n, \)
  
  for every task \( t \) that cannot be mapped to node \( n \)

- Path delay constraints:
  
  \( \forall t_i \in S, t_j \in E, \)
  
  \( \text{Prob} \left( L[t_i, t_j] \leq \text{maxDelay} \right) \geq \text{minProbability} \)

- Objective function, to be minimised:
  
  The number of replicated tasks in \( T' \)
Mathematical Model (continued)

- Decision variables:
  \[ node[t] \in N, \]
  the node assigned to task \( t \in T \)

- Placement constraints:
  \[ node[t] \neq n, \]
  for every task \( t \) that cannot be mapped to node \( n \)

- Path delay constraints:
  \[ \forall t_i \in S, \ t_j \in E, \]
  \[ F_L[t_i, t_j] (\text{maxDelay}) \geq \text{minProbability} \]

- Objective function, to be minimised:
  The number of replicated tasks in \( T' \)
Path delay random variable:

\[ L[t_1, t_3] = D[t_1, t_2] + D[t_2, t_3] \]

and the c.d.f. of \( L[t_1, t_3] \) becomes:

\[ L[t_1, t_3] \sim N(0.8, 2.5) \]

Path delay probability:

\[
\text{Prob}(L[t_1, t_3] \leq 3.1) = \Phi_{0.8, 2.5}(3.1) = 0.927117
\]

Only if node\( [t_1] \neq \text{node}[t_2] \neq \text{node}[t_3] \)
Example

Path delay random variable:

\[ L[t_1, t_3] = D[t_1, t_2] + D[t_2, t_3] \]

and the c.d.f. of \( L[t_1, t_3] \) becomes:

\[ L[t_1, t_3] \sim N(0.8, 2.5) \]

Path delay probability:

\[ \text{Prob}(L[t_1, t_3] \leq 3.1) = \Phi_{0.8,2.5}(3.1) = 0.927117 \]

Only if \( \text{node}[t_1] \neq \text{node}[t_2] \neq \text{node}[t_3] \)
Example (continued)

- If \( \text{node}[t_1] = \text{node}[t_2] \neq \text{node}[t_3] \):
  \[
  D[t_1, t_2] \sim N(0.5, 1.0) \\
  D[t_2, t_3] \sim N(0.3, 1.5)
  \]

  then the c.d.f. of \( L[t_1, t_3] \) becomes:
  \[
  L[t_1, t_3] \sim N(0.3, 1.5)
  \]

- Path delay probability:
  \[
  \text{Prob}(L[t_1, t_3] \leq 3.1) = \Phi_{0.3,1.5}(3.1) = 0.988878
  \]

- Task mapping matters!
Example (continued)

- If \( node[t_1] = node[t_2] \neq node[t_3] \):
  \[
  D[t_1, t_2] \sim N(0.5, 1.0)
  \]
  \[
  L[t_1, t_3] = D[t_1, t_2] + D[t_2, t_3] = D[t_2, t_3]
  \]
  then the c.d.f. of \( L[t_1, t_3] \) becomes:
  \[
  L[t_1, t_3] \sim N(0.3, 1.5)
  \]
- Path delay probability:
  \[
  \text{Prob}(L[t_1, t_3] \leq 3.1) = \Phi_{0.3,1.5}(3.1) = 0.988878
  \]
- Task mapping matters!
Example (continued)

- c.d.f. of the latency on an edge:

\[
F_D[t_i, t_j] = \begin{cases} 
F(D; 0) & \text{if } node[t_i] = node[t_j] \\
\Phi_{\mu_{i,j},\sigma_{i,j}} & \text{otherwise}
\end{cases}
\]

- c.d.f. of the latency on the path from \( t_1 \) to \( t_3 \):

\[
F_L[t_1, t_3] = \begin{cases} 
F(D; 0) & \text{if } node[t_1] = node[t_2] = node[t_3] \\
\Phi_{\mu_{1,2},\sigma_{1,2}} & \text{if } node[t_1] \neq node[t_2] = node[t_3] \\
\Phi_{\mu_{2,3},\sigma_{2,3}} & \text{if } node[t_1] = node[t_2] \neq node[t_3] \\
\Phi_{\mu_{1,2}+\mu_{2,3},\sigma_{1,2}+\sigma_{2,3}} & \text{if } node[t_1] \neq node[t_2] \neq node[t_3]
\end{cases}
\]
End-to-End Requirements (Generic Case)

\[
p_1 : (t_1 \rightarrow t_2 \rightarrow t_3)
\]

\[
p_2 : (t'_1 \rightarrow t'_2 \rightarrow t_3)
\]

\[
p_3 : (t'_1 \rightarrow t'_2 \rightarrow t_3)
\]

Latency random variable becomes:

\[
L[t_i, t_j] = \sum_{(t_k, t_j) \in A'_j} (L[t_i, t_k] \oplus D[t_i, t_j])
\]

The c.d.f. of \( L[t_i, t_j] \) becomes:

\[
\forall t_i \in S, \ t_j \in E,
F_L[t_i, t_j] = 1 - (1 - F_L[t'_1, t_j]) \cdot (1 - F_L[t'_2, t_j]) \cdot \ldots \cdot (1 - F_L[t'_l, t_j])
\]
Probability Model (Generic Case)

The probability model must specify the operators:

- $\oplus$: how to combine the c.d.f. along a path between two end points.
  With latency: operator $\oplus$ becomes $+$.

- $\ominus$: how to combine the c.d.f. of each path fan-in at an end point.
  With latency: operator $\ominus$ becomes max.
Constraint Programming (CP) Model

- **CP\textsubscript{sub}**: solves each path delay constraint separately
  - Smaller variable domains and smaller search space
  - More efficient to evaluate each constraint

- **CP\textsubscript{main}**: enforces the solutions to each CP\textsubscript{sub}:
  - Using the extensional constraint.
  - CP\textsubscript{main} creates task replicates if no solution is found.
  - Enforces a lower bound on the task replication:
    \[
    r \geq \frac{\log(1 - \text{minProbability})}{\log(1 - \max (F_L[t_i, t_j]))}
    \]
  - Increases task replicates until all path delay constraints are satisfied.
  - The first solution to CP\textsubscript{main} is the optimum solution.
Platform

- CP solver: *Gecode* (version 4.2.0, open-source)

- Operating system: Mac OS X 10.8.4 (64-bit)

- CPU: Intel Core i5 2.6GHz, 3MB cache

- Memory: 8GB
Results: Optimisation Runtime

![Graph showing optimisation runtime for different maxDelay values.](image)

- **maxDelay = 3**
- **maxDelay = 2**
- **maxDelay = 1**

Runtime (seconds)

Highway traffic instances

SensorNets'14 Uppsala University, Sweden

Hassani, Pathak, Pearson, Issarny, Jonsson
Results: Number of Replicated Tasks

- maxDelay = 3
- maxDelay = 2
- maxDelay = 1

Highway traffic instances
Results: Number of Replicated Tasks on Instance \(\langle 132, 189, 21 \rangle\)
Summary and Conclusion

- Our approach is generic:
  - The model can capture many end-to-end requirements, given by the probability model.
  - We only need to change the two operators $\oplus$ and $\ominus$ depending on the application.

- It is effective and efficient to use Constraint Programming (CP):
  - Directly captures the mathematical model.
  - Solves the problem to optimality.
Future Work

- Integrate our approach into the publicly available Srijan toolkit\(^1\) for the ATaG compilation process.

- Explore the overhead introduced by the task replicates.

- Investigate the application of our work in cloud computing and related technologies.

---

\(^1\)http://code.google.com/p/srijan-toolkit/
This research is sponsored by the Swedish Foundation for Strategic Research (SSF) under research grant RIT08-0065 for the project *ProFuN: A Programming Platform for Future Wireless Sensor Networks*. 
Questions

- Thank you!
- Questions:
  - farshid.hassani@it.uu.se
Mathematical Model

- **Constants:**
  - $N = \text{set of wireless sensor network nodes}$
  - $T = \text{set of tasks}$
  - $T' = \text{set of replicated tasks}$
  - $A = \text{set of arcs in the task graph } (T, A)$
  - $F_D[t_i, t_j] = \text{cumulative distribution function (c.d.f.) of the delay random variable } D[t_i, t_j], (t_i, t_j) \in A$
  - $S \subseteq T = \text{the set of (start) tasks where a triggering event is produced}$
  - $E \subseteq T = \text{the set of (end) tasks where producing an output within a given latency time is required}$
Constraint Programming (CP) Model - cont.

```
input : N, T, S, E, minProbability
output: node

1 solved ← false
2 add placement constraints to CP_{main}
3 while not solved do
4    taskCopiesRequired ← false
5    for all t_i ∈ S, t_j ∈ E do
6       tupleSet[t_i, t_j] ← ∅
7       taskCopies ← 0
8         for all solutions s in CP_{sub}(p[t_i, t_j]) do
9            if checker(s, MF) then
10               tupleSet[t_i, t_j] = tupleSet[t_i, t_j] ∪ s
11            if tupleSet[t_i, t_j] ≠ ∅ then
12               add extentional constraints to CP_{main}
13            else
14               taskCopies ← \frac{\log(1 - minProbability)}{\log(1 - MF)}
15               replicate(p[t_i, t_j], taskCopies)
16               taskCopiesRequired ← true
17         if not taskCopiesRequired then
18            solve(CP_{main})
19            if CP_{main} has a solution then
20               solved ← true
21               return node
22            else
23               replicate(1)
```