Decision Guidance Systems and Applications
To Manufacturing, Power Grid, Supply Chain and IoT

Alex Brodsky
Department of Computer Science
Volgenau School of Engineering
George Mason University
cs.gmu.edu/~brodsky

ICEIS 2018 Keynote Address
Acknowledgements: GMU faculty co-authors

- Prof. Paul Amman
- Prof. Chun-Hung Chen
- Prof. Carlotta Domeniconi
- Prof. Igor Griva
- Prof. Sushil Jajodia
- Prof. Larry Kerschberg
- Prof. Jessica Lin
- Prof. Daniel Menasce
- Prof. Ami Motro
- Prof. Lance Sherry
- Prof. John Shortle
- Prof. Joao Sousa
- Prof. X. Sean Wang
- Prof. Jon Whittle
- Prof. Duminda Wijesekera
Acknowledgements: my PhD students

Graduated

- Dr. John McDowal, *A Framework for Optimal Service Composition and Execution Based on Business Process Management Notation (BPMN)*, 2014 (jointly w/ Prof. Kerschberg)
- Dr. Gordon Shao, *Decision Guidance for Sustainable Manufacturing*, 2013. (jointly with Prof. Ammann)
- Dr. Khalid Alodhaibi, *Decision-Guided Recommenders with Composite Alternatives*, 2011.
- Prof. Samuel Varas *On Optimal Constraint Decomposition, Monitoring and Management in Distributed Environments*, 1998. (jointly with Prof. Kerschberg)

Current

- M. Omar Nachawati, *Algorithms for Decision Guidance Management System*
- Fernando Bocanera, *Decision Guidance for Healthcare Individual and Group Policy*
What is it about?

Decision Guidance (DG) systems are a class of decision support systems geared to

- elicit knowledge from domain experts and
- provide actionable recommendations to human decision-makers,
- with the goal of arriving at the best possible course of action.
Examples of Decision Guidance Systems: Supply Chain Management
Examples of Decision Guidance Systems: Renewable Power and Storage

GMU pilot project for Dominion Virginia Power
Examples of Decision Guidance Systems:
Tesla prep and body shop

GMU project for the National Institute of Standards & Technology
Outline

- DG systems: need, challenges, vision
- DG language & tool example:
  - DG Analytics Language (DGAL) & Management System (Unity DGMS)
- DG application example:
  - Manufacturing and supply service networks based on model repository
- DG algorithm example:
  - Optimizing multistage service networks based on preprocessing and decomposition
- Broader view on DG research: languages/tools, algorithms, applications
- Three grand challenges:
  - IoT + DG = (Smart) Cyber Physical Service Networks
  - Design (e.g., product, process, architectural, …) + DG = (Smart) Parametric Design
  - Public policy (e.g., renewable energy) + DG + Group decision methods = (Smart) public policy
- Conclusions
Decision Guidance Systems (DGS)

DGS need to

• use and mine large amounts of data
• elicit knowledge about model structure from domain experts
• learn deterministic or stochastic models
• elicit metrics, KPI and decision objectives
• perform analysis tasks, incl. monitoring, diagnosis, prediction, optimization
• explain actionable recommendations to decision-makers
• solicit decision-makers feedback for iterative improvement
TODAY every task is implemented:

- one off
- from scratch
- non-reusable
- non-modular
- requires math, OR, IT & domain-specific expertise
- high cost
- long development cycle
- difficult to modify/extend

Tools

- **DBMS**: SQL, XQuery, JSONiq
- **Learning/Mining**: PMML, PFA, ...
- **Simulation**: Modelica-based, Simulink, ...
- **Optimization**: MP/CP using OPL, AMPL, ...

Domain Specific Modeling & Analytics GUI

Interfaces / Mappers
Domain Specific Modeling & Analytics GUI

Reusable, extensible, modular KB of analytic models (AM)

- Domain-specific analytics views
- Domain-specific atomic models & instances
- Domain-specific composite models & instances

Tools
- **DBMS:** SQL, XQuery, JSONiq
- **Learning/Mining:** PMML, PFA, ...
- **Simulation:** Modelica-based, Simulink, ...
- **Optimization:** MP/CP using OPL, AMPL, ...

Decision Guidance Management System

DG Analytics Language:
- Computation
- Prediction
- Optimization
- Learning
- Pareto analysis …
Reusable, extensible, modular KB of analytic models (AM)

Domain Specific Modeling & Analytics GUI

Interfaces / Mappers

Decision Guidance Management System

DG Analytics Language:
- Computation
- Prediction
- Optimization
- Learning
- Pareto analysis …

DBMS: SQL, XQuery, JSONiq

Learning/Mining: PMML, PFA, …

Simulation: Modelica-based, Simulink, …

Optimization: MP/CP using OPL, AMPL, …

Tools
Outline

- DG systems: need, challenges, vision
- DG language & tool example:
  - DG Analytics Language (DGAL) & Management System (Unity DGMS)
- DG application example:
  - Manufacturing and supply service networks based on model repository
- DG algorithm example:
  - Optimizing multistage service networks based on preprocessing and decomposition
- Broader view on DG research: languages/tools, algorithms, applications
- Three grand challenges:
  - IoT + DG = (Smart) Cyber Physical Service Networks
  - Design (e.g., product, process, architectural, …) + DG = (Smart) Parametric Design
  - Public policy (e.g., renewable energy) + DG + Group decision methods = (Smart) public policy
- Conclusions
### Background: JSON (Java Script Object Notation) and JSONiq query language

<table>
<thead>
<tr>
<th>Data format/model</th>
<th>Query/data manipulation language</th>
<th>Type of data</th>
<th>Used for</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Relational (tabular) | SQL | • structured  
• flat tables  
• human readable | Relational databases, such as Oracle, SQL server | • dominant in Buz Info Sys  
• Not as data interchange format |
| XML | XQuery | • semi-structured  
• User-defined tags, HTML-like for data  
• hierarchical  
• human and machine readable | • data interchange  
• web-services | • Popular  
• Verbose |
| JSON | JSONiq  
= SQL for NoSQL stores | • semi-structured  
• objects = key-values pairs  
• hierarchical  
• human and machine readable  
• compact as compared to XML | • lightweight data interchange  
• web-services  
• REST SOA  
• IoT stack  
• Data in NoSQL stores, incl. MongoDB, | • Highly popular  
• Identical w/JS data model  
• similar Python & Java data models  
• Dominant in REST, IoT, asynchronous client/server communication, replacing XML |
$P$: fixed params
$V$: control params

$M$: metrics
$C$: constraints
P: fixed params
V: control params

M: metrics
C: constraints

```json
{
    "demand": {"chair": 16, "table": 4},
    "purchaseInfo": {
        "ppu": {"supplier1": {"chair": 50.00, "table": 110.00},
                  "supplier2": {"chair": 60.00, "table": 100.00}
        },
        "available": {"supplier1": {"chair": 15, "table": 3},
                       "supplier2": {"chair": 20, "table": 2}
        },
        "qty": {"supplier1": {"chair": 10, "table": 2},
                 "supplier2": {"chair": 20, "table": 2}
        }
    }
}
```
declare function procurementPM($procurementInput) {
let $demand := $procurementInput.demand,
    $suppliers := keys($procurementInput.purchaseInfo.ppu),
    $ppu := $procurementInput.purchaseInfo.ppu,
    $available := $procurementInput.purchaseInfo.available,
    $qty := $procurementInput.purchaseInfo.qty
let $cost := sum ( for $s in $suppliers, $i in keys($qty($s))
    return $ppu($s)($i) * $qty($s)($i)
)
let $availabilityConstraint := ( 
    every $s in $suppliers, $i in keys($qty($s))
    satisfies ( $qty($s)($i) <= $available($s)($i) )
)
let $supply := {} | 
    for $i in keys($demand)
    return {i: sum ( for $s in $suppliers return $qty($s)($i) )} |
let $demandSatisfiedConstraint := 
    every $i in keys($demand)
    satisfies $demand($i) <= $supply($i)
let $feasibilityConstraint :=
    $availabilityConstraint and $demandSatisfiedConstraint
return { cost: $cost, constraints: $feasibilityConstraint }
P: fixed params
V: control params

AM

M: metrics
C: constraints

```json
{
  "demand": {
    "chair": 16, "table": 4,
  },
  "purchaseInfo": {
    "ppu": {
      "supplier1": {
        "chair": 50.00, "table": 110.00,
      },
      "supplier2": {
        "chair": 60.00, "table": 100.00
      }
    },
    "available": {
      "supplier1": {
        "chair": 15, "table": 3,
      },
      "supplier2": {
        "chair": 20, "table": 2
      }
    },
    "qty": {
      "supplier1": {
        "chair": 10, "table": 2,
      },
      "supplier2": {
        "chair": 20, "table": 2
      }
    }
  },
  "cost": 2120,
  "constraints": true
}
```
P: fixed params
V: control params

{  
  "demand": {"chair": 16, "table": 4},
  "purchaseInfo": {  
    "ppu": { "supplier1": {"chair": 50.00, "table": 110.00},
             "supplier2": {"chair": 60.00, "table": 100.00}  
    },
    "available": { "supplier1": {"chair": 15, "table": 3},
                    "supplier2": {"chair": 20, "table": 2}  
    },
    "qty": { "supplier1": {"chair": ???, "table": ??? },
             "supplier2": {"chair": ???, "table": ???}  
    }
  },
  "cost": ???,
  "constraints": true
}
P: fixed params
V: control params

M: metrics
C: constraints

P: fixed params
V: ???
O: objective

Minimize/Maximize

P: params
V: optimal control
Soundness and Completeness of Reduction

\[ \min f(V) \quad \text{s.t.} \quad C(V) \quad \rightarrow \quad \Phi(q,d) \quad \rightarrow \quad ? \quad \Phi'(q,d) \quad \rightarrow \quad I(V) \]

semantics

reduction

symbolic output structure

reconstruction

optimization solver
Soundness and Completeness of Reduction

**Theorem:**
The reduction procedure is

1. **sound:** \( \Phi'(q,d) \subseteq \Phi(q,d) \)
2. **complete:** \( \Phi'(q,d) \supseteq \Phi(q,d) \)

\[ \min_{f(V)} \text{s.t. } C(V) \]

\[ \Phi(q,d) \quad ? \quad \Phi'(q,d) \]

\[ I(V) \]

\[ \text{optimization solver} \]

\[ \text{symbolic output structure} \]

\[ \text{reduction} \]

\[ \text{semantics} \]
declare function local:stochProcurementPM($procurementInput){
  let $demand := $procurementInput.demand,,
  $suppliers := keys($procurementInput.purchaseInfo.ppu),
  $ppu := $procurementInput.purchaseInfo.ppu,
  $available := $procurementInput.purchaseInfo.available,
  $qty := $procurementInput.purchaseInfo.qty

  let $cost := sum ( for $s in $suppliers, $i in keys($qty($s))
    let $stochPpu := $ppu($s)($i) + G(0.0, 0.01*$ppu($s)($i))
    return $stochPpu * $qty($s)($i)
  )

  let $availabilityConstraint := (every $s in $suppliers, $i in keys($qty($s))
    satisfies ( $qty($s)($i) <= $available($s)($i) )
  )

  let $supply := {}
    for $i in keys($demand)
      return {$i: sum ( for $s in $suppliers return $qty($s)($i) )}

  if $supply != {} then $supply := $supply

  let $demandSatisfiedConstraint :=
    every $i in keys($demand)
    satisfies $supply($i) >= $demand($i) + G(0.0, 0.02 * $demand($i))

  let $feasibilityConstraint :=
    $availabilityConstraint and $demandSatisfiedConstraint
  return { cost: $cost, constraints: $feasibilityConstraint }
P: params
V: control vars

Stochastic
AM

M: metrics
C: constraints

P: params
V: control vars

(stochastic) prediction

E(M): mean & std. dev. of metrics
P(C): prob. of constraint satisfaction

{ "cost": {"mean": 2120, "sigma": 8.76},
 "constraints": { "prob": 0.92},
}
DGAL summary:

- Is based on KB of analytic (performance) models (AMs) that:
  - Express constraints and metrics of interest as a function of fixed & control parameters (of a process)
  - Are independent of analytical tasks & tools/algorithms
- Allows reuse of AMs as operands to diverse analytics operators (forming analytics algebra on AMs):
  - Simulation
  - Prediction
  - Deterministic or stochastic optimization
  - Learning parameters of AM for regression or classification
  - …..
- Performs these operators/tasks by automatic reduction to specialized low-level models and algorithms, w/out the need to manually craft low-level models.
- Uses query / data manipulation language (JSONiq) over JSON data format to
  - Express AMs
  - Express analytics algebra operators / tasks
Unity DGMS architecture
Unity DGMS: application management layer
Unity DGMS: tool management layer
Unity DGMS: analytics management layer
Compiling Optimization Queries

Six steps:
1. Reusable Analytic Model Resolution
2. Source-to-Source Transformation
3. Symbolic Execution
4. Target Model Generation
5. Target Solver Execution
6. Input Instantiation

Steps 1-3 & 6 (tasks) can be used to implement other DGAL operators: e.g. learn & predict
Preliminary performance evaluation

- **Hypothesis:** Execution time overhead of compiled reusable analytic models (into task- and tool-specific models) is within a constant factor of that of manually crafted ones.
- **Method:** Compare execution times of compiled DGAL model versus manually crafted OPL model on randomized input pairs.
- **Preliminary Results:** Compiled DGAL models are currently 2.3 times slower than manually crafted OPL models.
- We are investigating techniques to improve performance of compiled models.
Outline

- DG systems: need, challenges, vision
- DG language & tool example:
  - DG Analytics Language (DGAL) & Management System (Unity DGMS)
- DG application example:
  - Manufacturing and supply service networks based on model repository
- DG algorithm example:
  - Optimizing multistage service networks based on preprocessing and decomposition
- Broader view on DG research: languages/tools, algorithms, applications
- Three grand challenges:
  - IoT + DG = (Smart) Cyber Physical Service Networks
  - Design (e.g., product, process, architectural, …) + DG = (Smart) Parametric Design
  - Public policy (e.g., renewable energy) + DG + Group decision methods = (Smart) public policy
- Conclusions
Heat sink (HS) assembly manufacturing & supply service network

Heat Sink Supply Chain (L0)

- Alumina (L2)
- Accessories (L2)
- Supply (L1)

Alumina Powder

Manufacturing (L1)

Aluminum Plate

- Smelting (L2)
- Aluminum Plate Contract Manufacture (L2)

Heat Sink Base

- HS Base Shearing (L3)
- HS Base Drilling (L3)
- HS Base PL (L2)
- HS Base Contract Manufacture (L2)

HS PL (L2)

- HS Shearing (L3)
- Anodizing (L3)
- CNC Machining (L3)
- Quality Inspection (L3)
- Assembly (L3)

Assembly (L3)

Finished Heat Sink Part

Demand (L1)
Machining Sequence

#1 Milling
millType: face
depth: 8
distOffset: 1.261

#2 Milling
millType: peripheral
depth: 1.5
distOffset: 1.261

#3 Milling
millType: peripheral
depth: 1.5
distOffset: 1.261

#4 Milling
millType: peripheral
depth: 1.5
distOffset: 1.261

#5 Milling
millType: peripheral
depth: 1.5
distOffset: 1.261

#6 Milling
millType: peripheral
depth: 1.5
distOffset: 1.261

#7 Milling
millType: peripheral
depth: 1.5
distOffset: 1.261

#8 Drilling
D: 6.4
p_coolant: 1
t_loading: 9.3

#9 Drilling
D: 4.76
p_coolant: 1
t_loading: 9.3

#10 Drilling
D: 3.8
p_coolant: 1
t_loading: 9.3

#11 Milling
millType: peripheral
depth: 1.5
distOffset: 1.261
Defining the nodes in the service network

**Vendor** – organization that provides a finished product

**Contract Manufacturer** – organization that provide a manufacturing service

**Internal Manufacturer** – internal activity controlled by OEM

**Production Line** – a chain of internal manufacturer activities controlled by OEM
Architecture around NIST model repository
Composition of service network

L0
- supply_11
- manufacturing_11
- demand_11

L1
- Alumina
- Accessories package
- smelting_12
- hs_pl_12
- hs_base_contract_manuf_12
- hs_base_pl_12
- Heat Sink

L2
- Aluminum Plate
- Accessories package
- Heat Sink Base
- quality_inspection_13
- cnc_machining_13
- anodizing_13
- assembly_13
- Heat Sink

L3
- Cut Aluminum Plate
- atomic PM
- Anodized Plate
Service network analytic model
Service network analytic model (SV-AM): key steps

1. If Service atomic, invoke its corresponding AM
2. Else (* if service has sub-services *)
   a. Recursively invoke SV-AM for every sub-service
   b. Aggregate metrics
   c. Evaluate constraints, which comprise of:
      i. All sub-service constraints
      ii. Bounds on Control (decision) variables
      iii. Zero-sum flow constraints
3. Return output that comprise of:
   a. Aggregated metrics
   b. Evaluated constraints
   c. For every descendent sub-service, its aggregated metrics & evaluated constraints
Architecture around NIST model repository
Example: Pareto front computation
Initial deployment architecture

- **GitLab Browser-based client**
- **REST call to Analytical Function, e.g., argmin, argmax**
- **add, commit, push**
- **GMU Datacenter**
System workflow
Outline

• DG systems: need, challenges, vision
• DG language & tool example:
  • DG Analytics Language (DGAL) & Management System (Unity DGMS)
• DG application example:
  • Manufacturing and supply service networks based on model repository
• DG algorithm example:
  • Optimizing multistage service networks based on preprocessing and decomposition
• Broader view on DG research: languages/tools, algorithms, applications
• Three grand challenges:
  • IoT + DG = (Smart) Cyber Physical Service Networks
  • Design (e.g., product, process, architectural, …) + DG = (Smart) Parametric Design
  • Public policy (e.g., renewable energy) + DG + Group decision methods = (Smart) public policy
• Conclusions
Problem: cost minimization in multistage production network
Multi-stage production optimization problem:

• **Decision variables:**
  - For every machine: *ON* flag & *thru* (or low level controls)
  - *thru* of all flows

• **Constraints:**
  - For every machine:
    - *cost* as a function of *thru*
    - *thru* bounds
    - Input material *qty* as a function of output *qty*
  - Zero-sum flow
  - Demand satisfaction

• **Objective:** minimize total cost
MINIMIZE total_cost:
    SUM[m IN Machines] Cost[m];

SUBJECT TO
    machine_operation {m IN Machines}:
        MinQty[m] <= Qty[m] <= MaxQty[m];
    machine_cost {m IN Machines}:
        Active[m] = 0 ==> Cost[m] = 0
        ELSE Cost[m] = c3[m]*Qty[m]^3 + c2[m]*Qty[m]^2 + c1[m]*Qty[m] + c0[m];
    machine_production {m IN Machines}:
        Active[m] = 0 ==> MachQty[m] = 0
        ELSE MachQty[m] = Qty[m];
    assembly_production {a IN Assemblies}:
        AsmQty[a] = SUM[m IN Machines] Production[a,m] * MachQty[m];
    product_production {p IN Products}:
        Produced[p] = SUM{a IN Assemblies} Output[p,a] * AsmQty[a];
    demand_vs_produced {p IN Products}:
        Produced[p] >= Demand[p] + 
        SUM{a IN Assemblies} Resources[a,p] * AsmQty[a];
Intuition behind approximations and problem decomposition
Principles of Preprocessing & Decomposition

\min f(x) \text{ s.t. } C(x)

of the form:

\min f_1(x_1, y) + \ldots + f_n(x_n, y) \quad (I)
\text{ s.t. } C_1(x_1, y) \land \ldots \land C_n(x_n, y) \land C_0(y)

**Problem:** $x_1, \ldots, x_n$ may involve many finite domain or binary variables
Decomposition Key Idea

Find optimal values for interface variables $y$ and fix to decompose problem.

**Define:**

$$F_1(y) = \min_{x_1} f_1(y, x_1) \quad K_1(y) = (\exists x_1) \ C_1(y, x_1)$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$F_n(y) = \min_{x_n} f_n(y, x_n) \quad K_n(y) = (\exists x_n) \ C_n(y, x_n)$$

**Step 1**

**Solve:**

$$\min \ F_1(y) + \ldots + F_n(y) \quad (II)$$

s.t. $K_1(y) \land \ldots \land K_n(y) \land C_0(y)$

**Claim:**

- A solution to (II) is a partial solution to the original problem, i.e.,
if $y^*$ is a solution to (II) then there exists a solution $(y^*, x_1^*, \ldots, x_n^*)$ to (I)

- A solution to the original problem (I) gives a solution to (II), i.e.,
if $(y^*, x_1^*, \ldots, x_n^*)$ is a solution to (I) then $y^*$ is a solution to (II)
**Step 2**

**Solve:**

\[
\begin{align*}
\min_{x_1} f_1(y^*, x_1) & \quad \text{s.t.} \quad C_1(y^*, x_1) \\
\vdots & \quad \vdots \\
\min_{x_n} f_n(y^*, x_n) & \quad \text{s.t.} \quad C_n(y^*, x_n)
\end{align*}
\]

**Claim:** Step 1 and Step 2 give a solution to the original problem, i.e., if \( x_1^* \) is a solution to (1), then \( (y^*, x_1^*, \ldots, x_n^*) \) is a solution to (I)

\[
\begin{align*}
\vdots & \\
x_n^* \text{ is a solution to (n)},
\end{align*}
\]

Essentially, Step 1 “decomposes” a (large) combinatorial problem into \( n \) (smaller) combinatorial problems which are easier to solve.

**Problem:** \( F_1(y), \ldots, F_n(y) \) and \( K_1(y), \ldots, K_n(y) \) may not have an analytical form to solve (II) using existing solver technology (e.g., LP, MILP, QP, NLP, etc.)
Approach

Approximate $F_1, \ldots, F_n$ and $K_1, \ldots, K_n$ using smooth functions for which we can use efficient solver techniques (LP, NLP, etc.).

Preprocessing:
1. Partition input space $y$, solve sub-problem exactly to build lookup table
2. Regression analysis to learning smooth approximation of $F_1, \ldots, F_n, K_1, \ldots, K_n$

On-line:
1. Solve (II) for $y^*$ using approximation for $F_1, \ldots, F_n$ and $K_1 \ldots, K_n$
2. Solve (1) … (n) based on $y^*$, find finite domain and binary variables $x_1, \ldots, x_n$
3. Solve original problem (I), where combinatorial part of the problem is fixed using the solution from Step 2
Online Decomposition Algorithm (ODA)

**Offline**
- Decompose problem into components
- Scan output range and solve MILP for optimal config

**Online**
- Create problem instance based on dynamic params
- Replace static components with approximation
- Solve approx. problem to fix interface variable
- Use look-up table to fix the internal variables
- Solve original problem without combinatorics
- Use heuristic search to select from look-up table

Pause and do heuristic search whenever *any* improved feasible solution to approx. problem is found.
Preprocessed Cost Function
Experimental Results
Outline

• DG systems: need, challenges, vision
• DG language & tool example:
  • DG Analytics Language (DGAL) & Management System (Unity DGMS)
• DG application example:
  • Manufacturing and supply service networks based on model repository
• DG algorithm example:
  • Optimizing multistage service networks based on preprocessing and decomposition
• Broader view on DG research: languages, algorithms, tools/applications
• Three grand challenges:
  • IoT + DG = (Smart) Cyber Physical Service Networks
  • Design (e.g., product, process, architectural, …) + DG = (Smart) Parametric Design
  • Public policy (e.g., renewable energy) + DG + Group decision methods = (Smart) public policy
• Conclusions
Broader overview of my DG research

• **DG languages, semantics & reductions**
  • For DB application developers (SQL, Xquery, JSONiq)
  • For software developers (CoJava)
  • For domain specific end-users

• **DG algorithms**
  • DG reduction algorithms
  • Process optimization by decomposition and pre-processing
  • Stochastic optimization of temporal processes through deterministic approximations
  • Regression of n-dimensional piece-wise linear functions
  • Classification over multivariate time series
  • Top-K recommendations using simulation and regression
  • Probabilistic algorithms to optimize recommendation diversity

• **DG tools, systems & applications**
  • Unity DGMS
  • Manufacturing, energy & power grids, supply chain, service networks, …
  • Package and group recommender systems, e.g., for investment in infrastructure, renewable energy, production capacity, …
Outline

• DG systems: need, challenges, vision
• DG language & tool example:
  • DG Analytics Language (DGAL) & Management System (Unity DGMS)
• DG application example:
  • Manufacturing and supply service networks based on model repository
• DG algorithm example:
  • Optimizing multistage service networks based on preprocessing and decomposition
• Broader view on DG research: languages/tools, algorithms, applications
• Three grand challenges:
  • IoT + DG = (Smart) Cyber Physical Service Networks
  • Design (e.g., product, process, architectural, …) + DG = (Smart) Parametric Design
  • Public policy (e.g., renewable energy) + DG + Group decision methods = (Smart) public policy
• Conclusions
**Question:** IoT + ? = Cyber Physical Systems

<table>
<thead>
<tr>
<th>features</th>
<th>Internet</th>
<th>SOA</th>
<th>IoT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose &amp; value</td>
<td>Easy Sharing of web content</td>
<td>Easy integration of heterogeneous IT systems</td>
<td>Easy development &amp; operation of cyber physical systems (CPS)</td>
</tr>
<tr>
<td>Enable sharing of</td>
<td>Web-content</td>
<td>IT web services</td>
<td>IoT-enabled cyber physical services (CPS)</td>
</tr>
<tr>
<td>Enablers:</td>
<td>Internet protocols Stack: HTML, HTTP, TCP/IP</td>
<td>Web services protocols stack:</td>
<td>IoT protocols stack</td>
</tr>
<tr>
<td>How to make sense of it?</td>
<td>Web search</td>
<td>Service discovery &amp; composition:</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• WSDL – API description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UDDI - discovery</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BPEL – composition &amp; execution</td>
<td></td>
</tr>
</tbody>
</table>
Cyber physical systems = execution of IoT services + DG analytics

- What is happening?
- Why did this happen?
- What will happen if …?
- How should we actuate?
Cyber physical systems =
execution of IoT services + DG analytics
Conclusions and future work

- Technical research challenges with impact on real-world problems
- **Main goal:** a robust DGMS
- **DBMS** have revolutionized the development of modern Information Systems
- Can **DGMS** have similar impact on the development of Decision Guidance Systems?
- Can **DGMS** significantly simplify the development of IoT cyber physical systems?
Conclusions and future work

- Technical research challenges with impact on real-world problems
- **Main goal**: a robust DGMS
- **DBMS** have revolutionized the development of modern Information Systems
- Can **DGMS** have similar impact on the development of Decision Guidance Systems?
- Can **DGMS** significantly simplify the development of IoT cyber physical systems?

Questions ???