Fundamentals of Modeling Systems and a System Approach to Simulation Optimization

Jun Ma
IEMS, Northwestern University
02/02/2005

OUTLINE

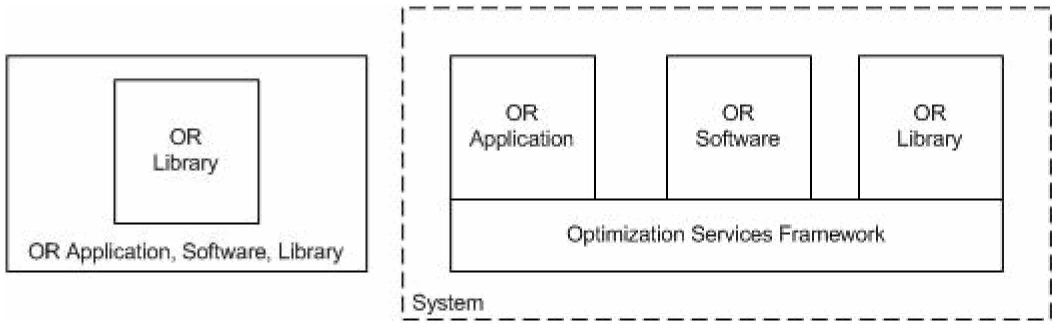
1. History and Background
2. Optimization Systems – Design and Architecture
3. System Components
4. AMPL-NEOS System
5. Motorola Intelligent Optimization System and Simulation Optimization
6. Conclusion

History and Background

- Linear programming by George Dantzig in the late 1940's
  - Intensive labor in translation from model to solver
  - Human labor alone
- Matrix generator (till early 1980's)
  - A computer code to generate coefficient matrices
  - Translation task divided between human and computer
- Modeling Language (mid 1980's till now)
  - GAMS, AMPL, LINDO, AIMMS, MPL, OPL, MOSEK
  - Translation entirely shifted to computer
  - Separation of data from model
  - Separation of modeling language form solver
  - Verifiable, modifiable, documentable, independent, simple
- Optimization server (mid 1990's)
  - Optimization web pages
  - Online optimization solvers
  - NEOS
- Optimization Services (current)
  - Registry
  - Decentralization (peer to peer)
  - XML and Web Services
  - Standards

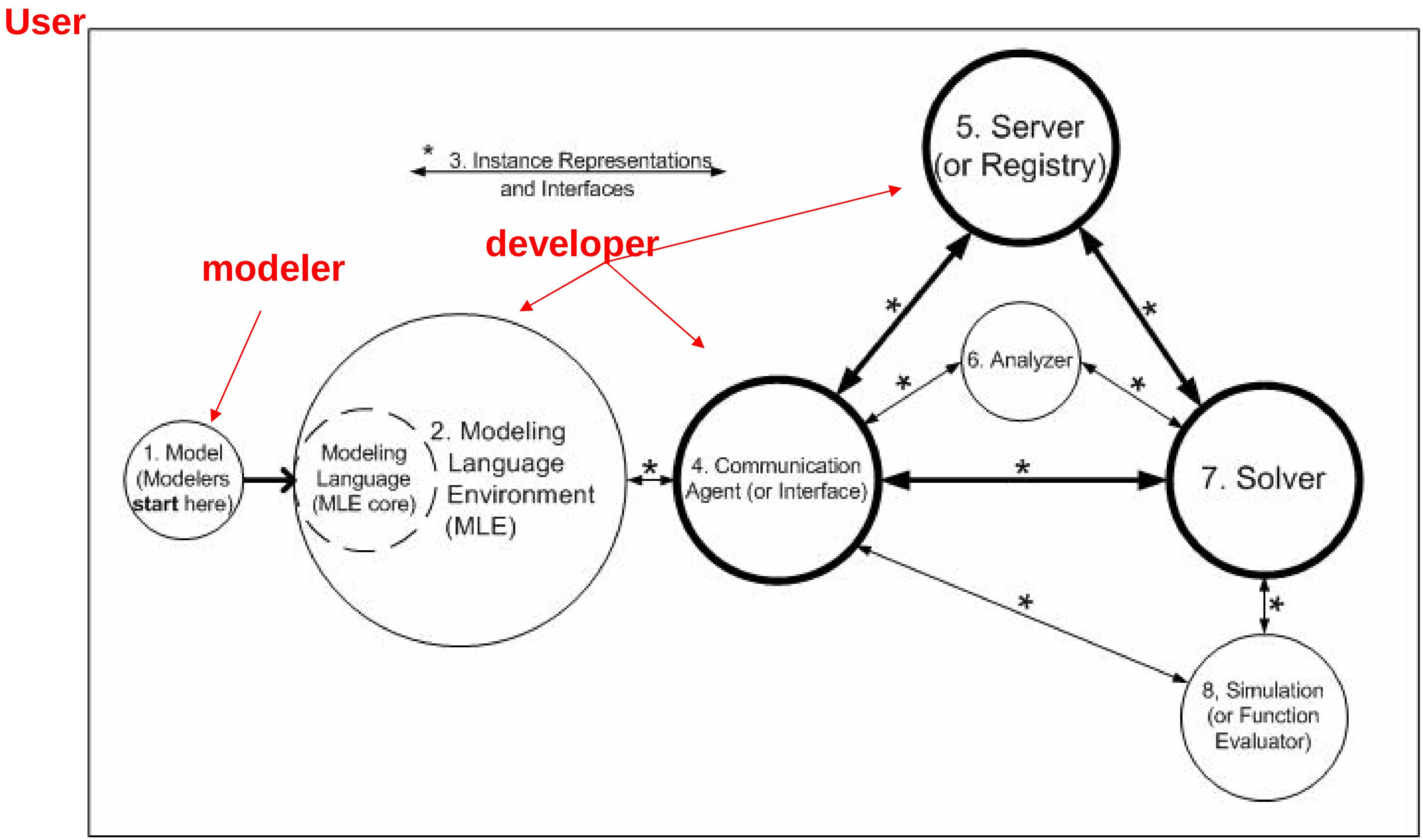
Optimization Systems
Terminology

- Modeling system (?)
- Modeling language environment (MLE)
  - Model language
  - Compiler
  - Auxiliary tools
- Optimization system
  - All the components discussed next
  - Including solvers
  - Local or distributed
- Library, system and framework



Optimization Systems

Design and Architecture



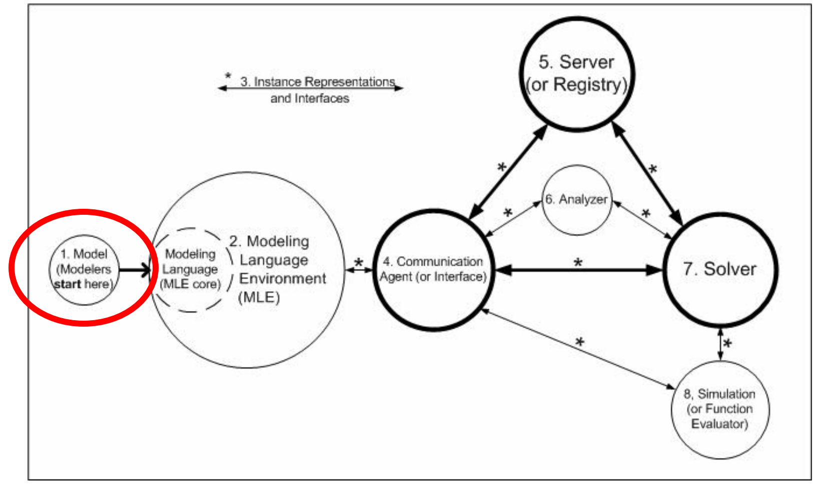
System Components

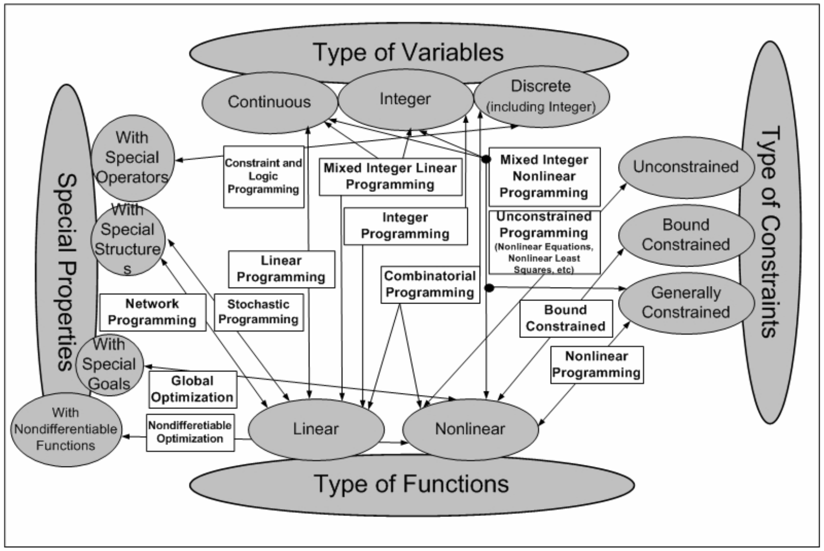
Model

- Different forms
  - Flowchart
  - Graphics
  - Mathematical program

$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & cx \\
\text{subject to} \quad & Ax = b \\
& x \geq 0
\end{aligned}
$$

- Different variation
  - Language variation
  - Algebraic variation
  - Type variation



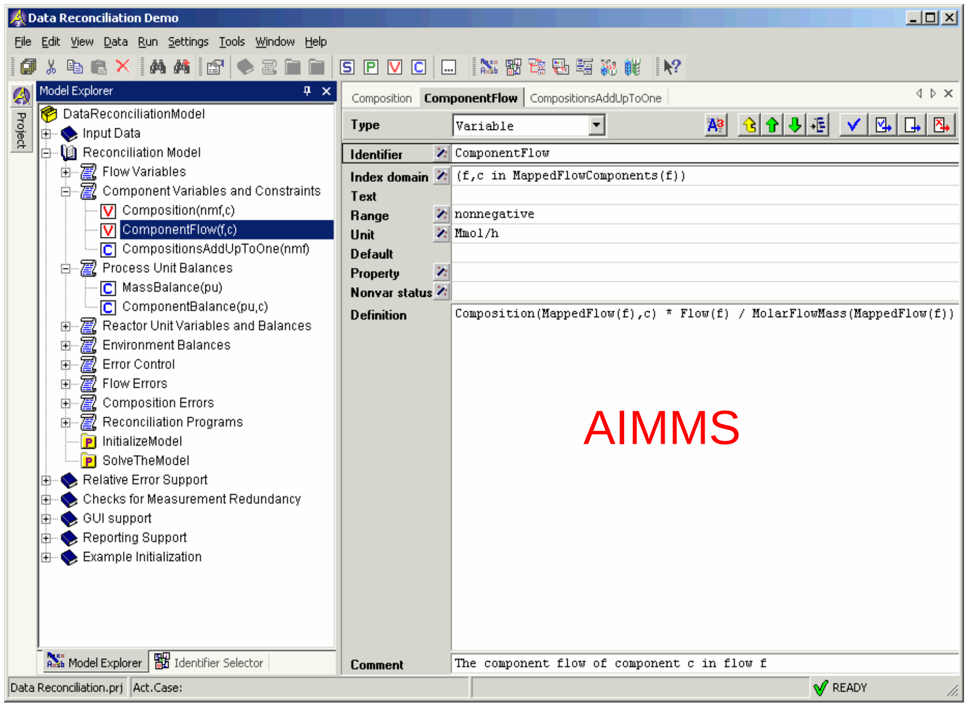


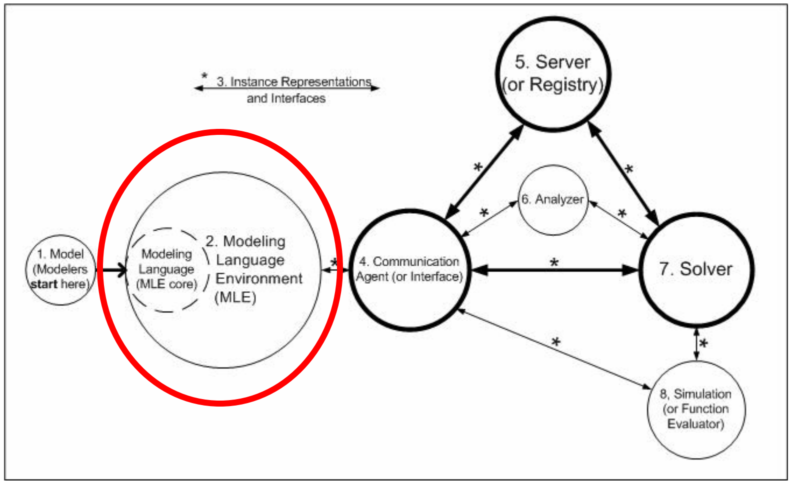
- Symbolic
- General
- Concise
- Understandable

System Components

Modeling Language Environment (MLE)

- Language design
- Compilation
- Auxiliary tools
  - Analyzer
  - Preprocessor
  - GUI





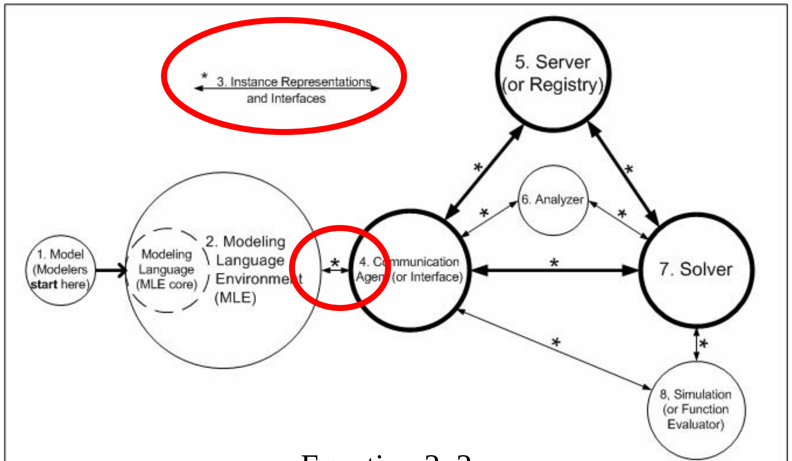
- Low-level instance generation

System Components
Instance Representation

- Characteristics
  - **explicit** rather than symbolic
  - **specific** rather than general
  - **redundant** rather than concise
  - **convenient** rather than understandable

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & -x_2 + 1/2(2x_1^2 - 3x_1x_3 + 4x_2^2 + 5x_3^2) \\ \text{subject to} \quad & 6x_1 + 7x_2 - 8x_3 \geq 9 \\ & \mathrm{x}_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0 x_1 \end{aligned}$$

```
NAME          qpEx
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        6
    x2        obj       -1
    x2        c1        7
    x3        c1        -8
RHS
    rhs       c1        9
QSECTION      obj
    x1        x1        2
    x1        x3        -3
    x2        x2        4
    x3        x3        5
ENDATA
```

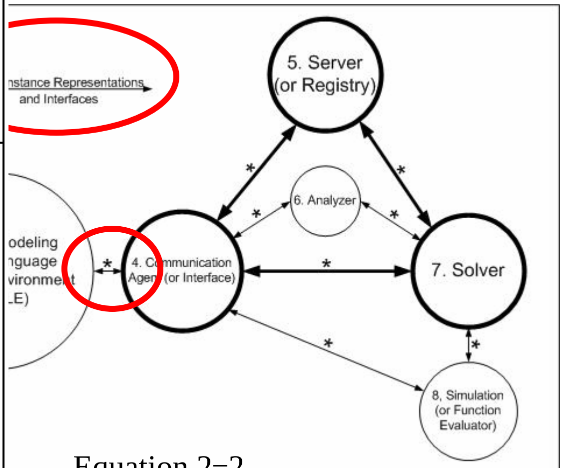


Equation 2−2

System Components

Instance Representation

Linear Programming Quadratic Programming Mixed Integer Linear Programming	MPS, xMPS, LP, CPLEX, GMP, GLP, PuLP, MLE instances
Nonlinearly Constrained Optimization Bounded Constrained Optimization Mixed Integer Nonlinearly Constrained Optimization Complementarity Problems Nondifferentiable Optimization Global Optimization	MLE instances SIF (only for Lancelot solver)
Semidefinite & Second Order Cone Programming	Sparse SDPA, SDPLR
Linear Network Optimization	NETGEN, NETFLO, DIMACS, RELAX4
Stochastic Linear Programming	sMPS
Stochastic Nonlinear Programming	None
Combinatorial Optimization	None (except for TSP input, only intended for solving Traveling Sales Person problems.
Constraint and Logic Programming	None
Optimization with Distributed Data	None
Optimization via Simulation	None

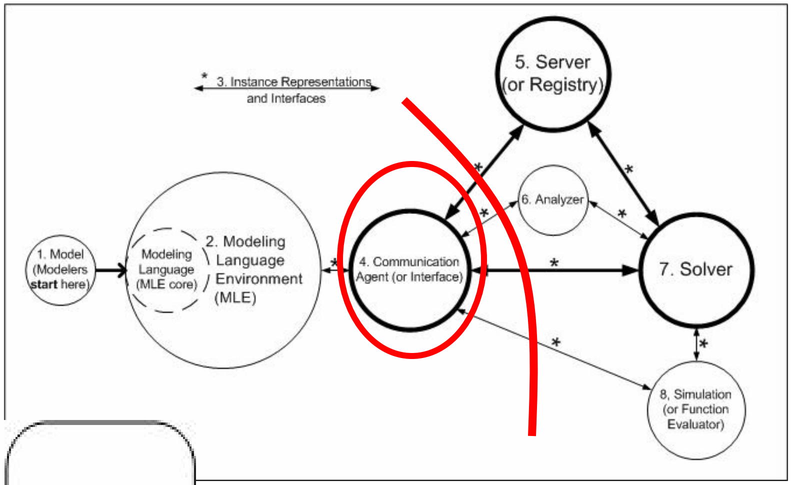


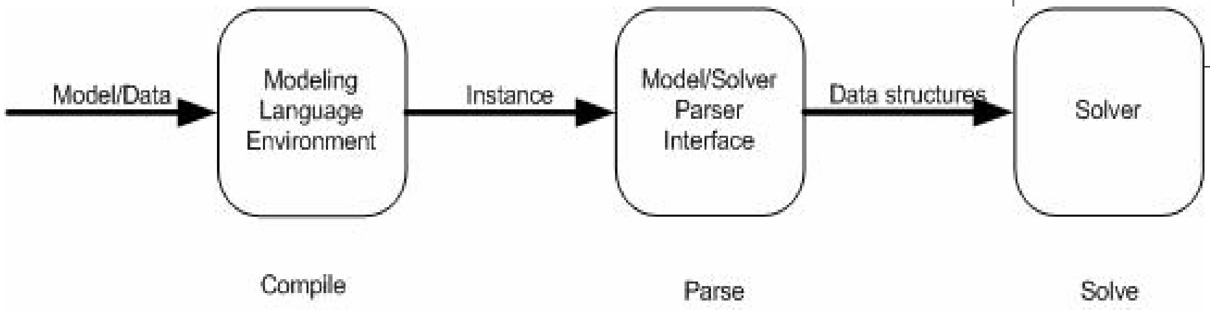
Equation 2−2

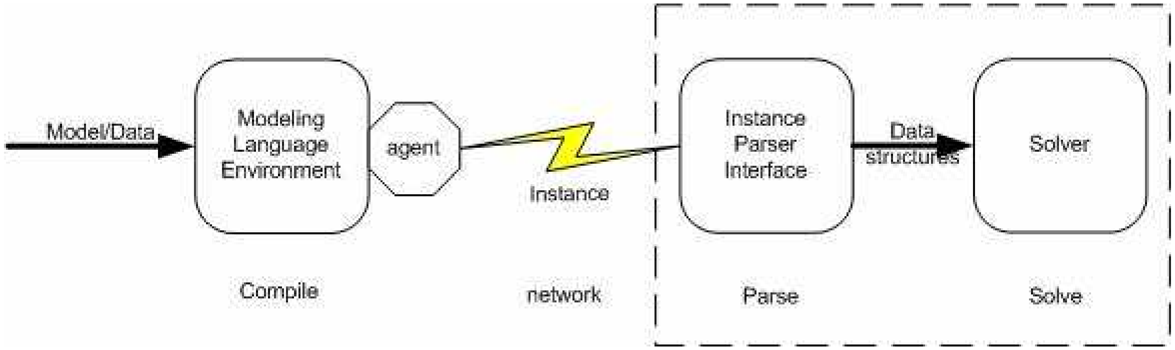
System Components

Interface(Local)/Communication Agent (Distributed)

- Interface
  - Between any two components
  - Compatibility (language, format etc.)
- Communication agent (agent)
  - Protocol
  - Compatibility (platform, protocol, system etc.)



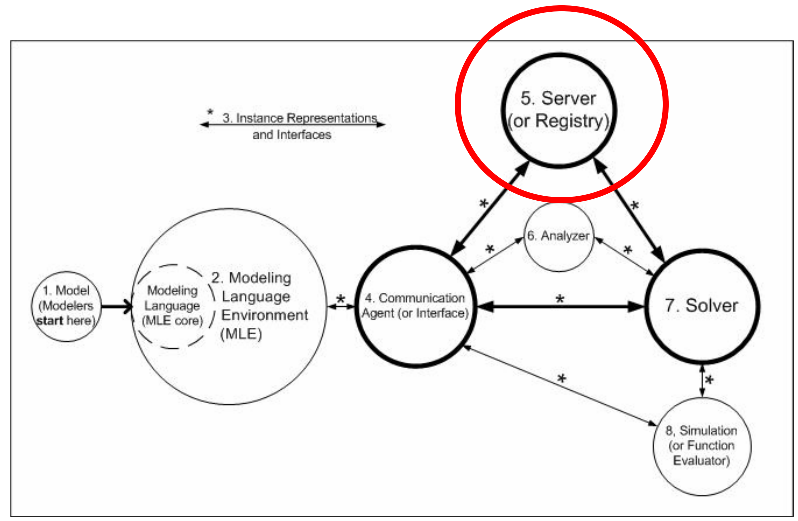


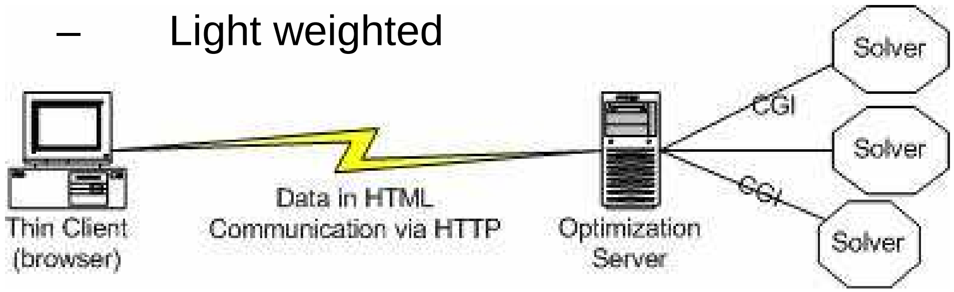


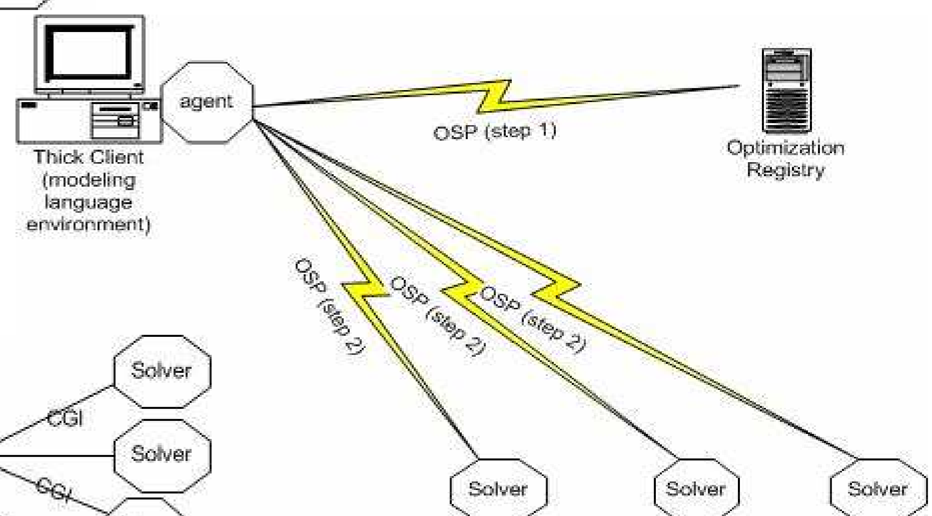
System Components

Server and Registry

- Server
  - Centralized
  - Heavy weighted
- Registry
  - Decentralized
  - Light weighted



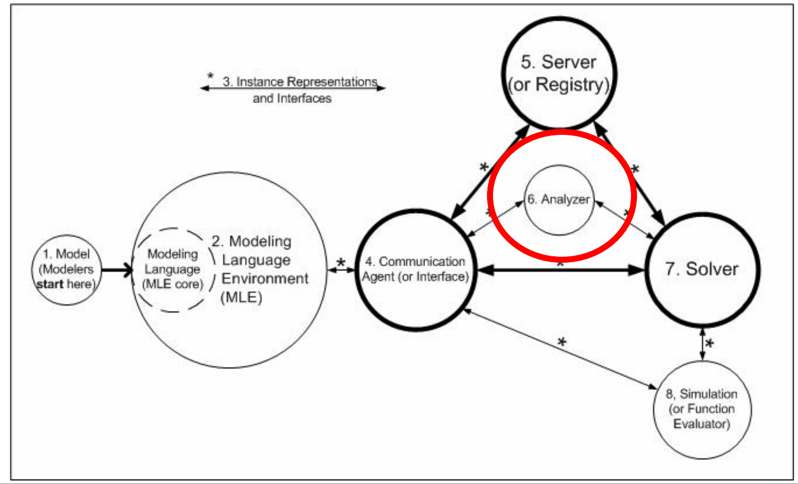


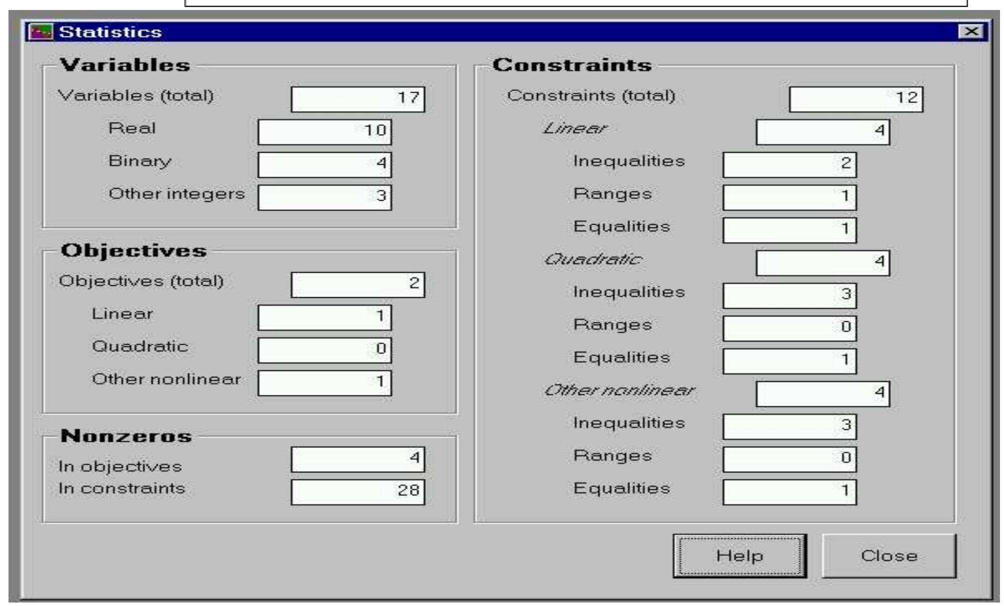


System Components
Analyzer

- Analyzer:Modeling Language :: Debugger:Programming Language
- Analyze low-level instance, NOT high-level modeling
- Some analysis are easy and involves only parsing
- Some involves computational analysis but can generate definite answer (e.g. network flow problem, quadratic problem)
- Some are hard and uncertain (e.g. convexity)

- **Analyzer is a separate component in an optimization system; it plays a key role in automation (no human interaction).**



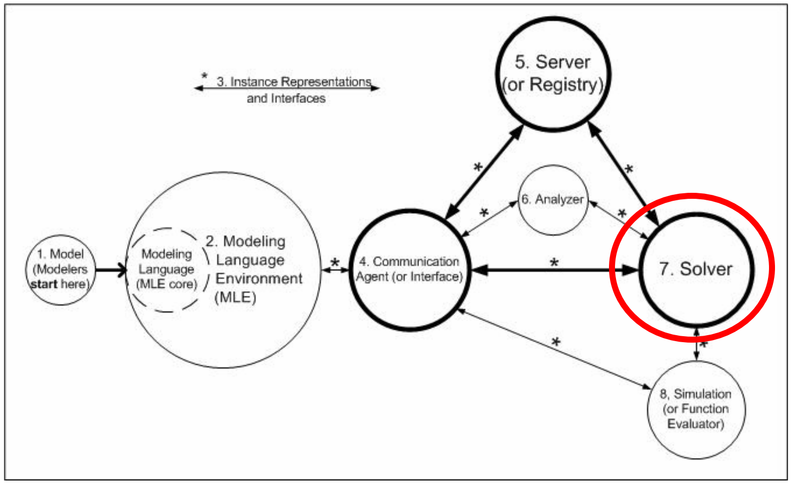


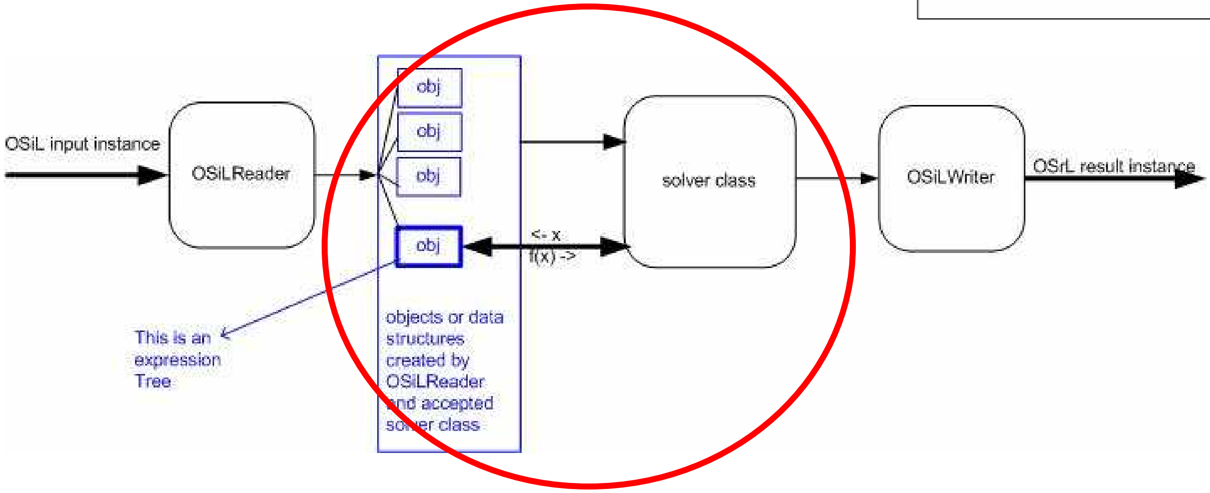
System Components

Solver

- The "contents" of an optimization system
- Solver discovery – FULLY automatic
- Solver registration – NOT automatic
  - Entity information
  - Process information
  - Option information
  - Benchmark information

- Right now the issues are NOT computation, but communication

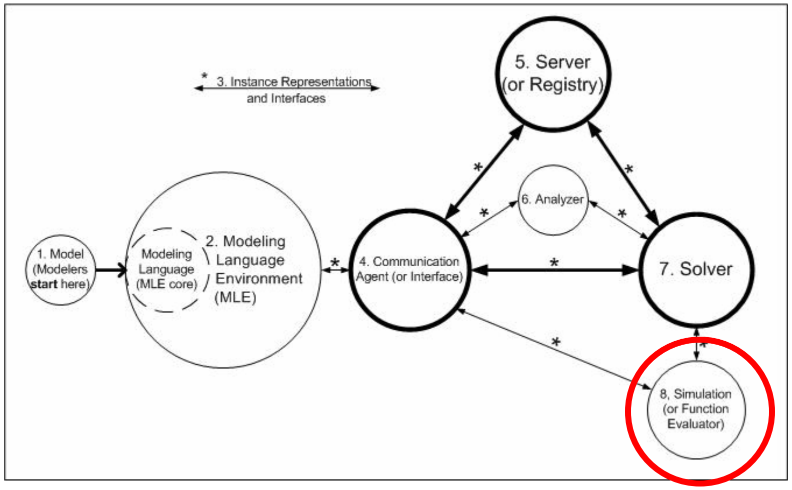


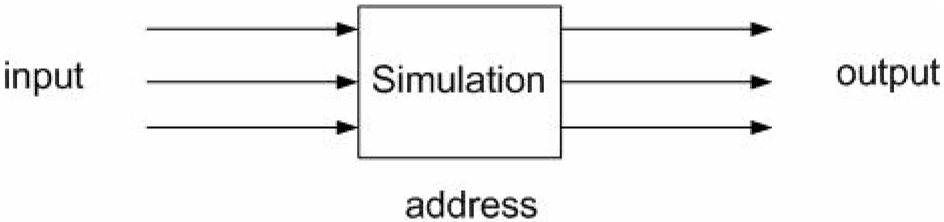


System Components

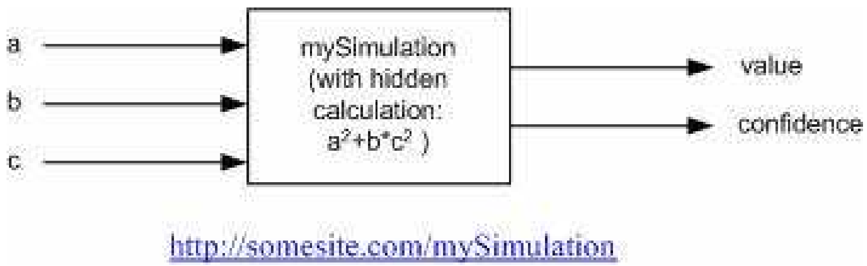
Simulation

- Any function evaluation
  - Function pointer: local, closed form
  - Simulation: remote, non-closed form
- Other properties of simulation (too complex, proprietary, multiple services, hard to move)





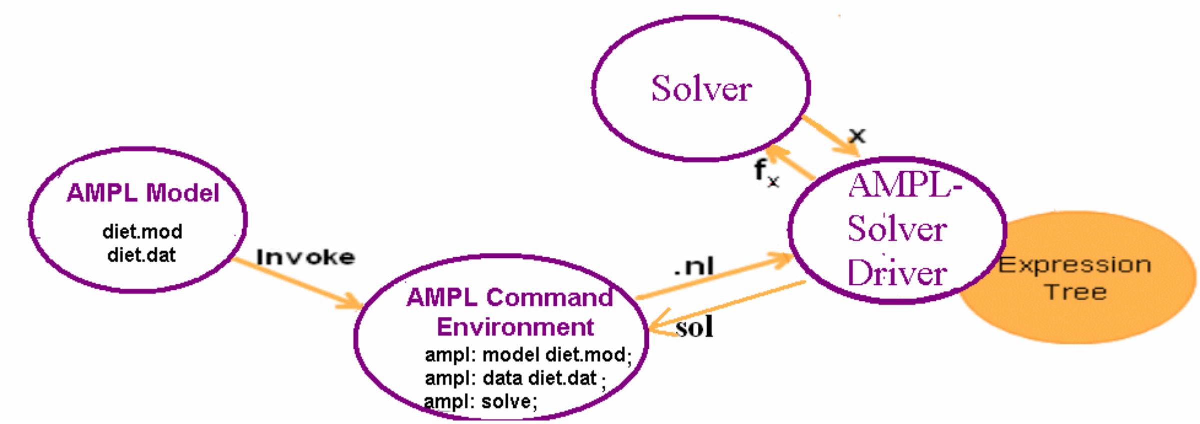
$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & x_1^2 + 2x_2^2 \\
\text{subject to} \quad & 2x_1 + 3x_2 \geq 9 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{aligned}
$$



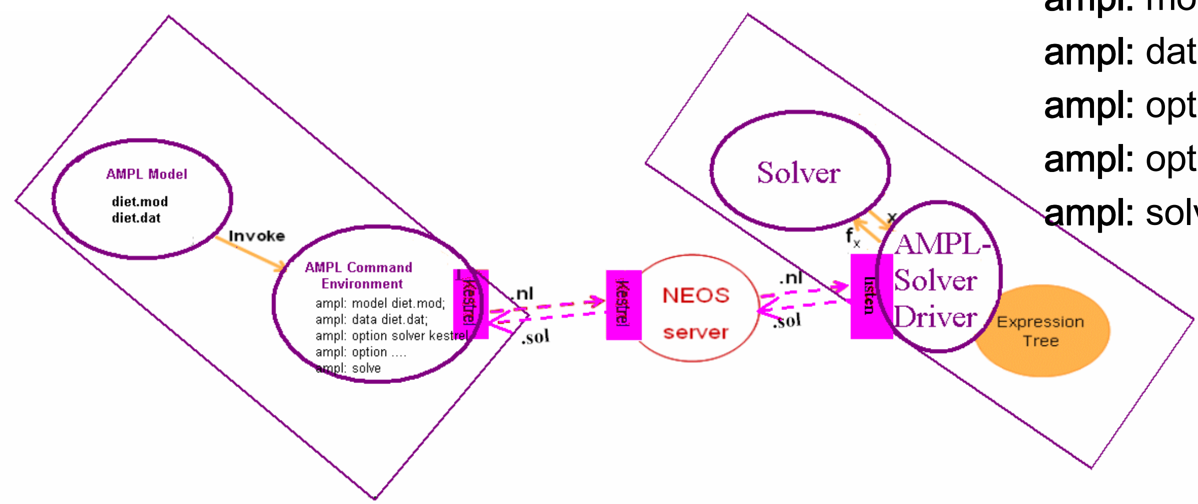
$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & mySimulation \\
\text{subject to} \quad & 2x_1 + 3x_2 \geq 9 \\
& x_1 \geq 0,\ x_2 \geq 0
\end{aligned}
$$

$$
\begin{aligned}
& mySimulation\{ \\
& address = http://somesite.com/mySimulation \\
& input: \\
& a = x_1 \\
& b = 2 \\
& c = x_2 \\
& output: \\
& value + confidence * 0 \\
& \}
\end{aligned}
$$

AMPL-NEOS System



ampl: model diet.mod;
ampl: data diet.dat;
ampl: option solver minos;
ampl: solve;



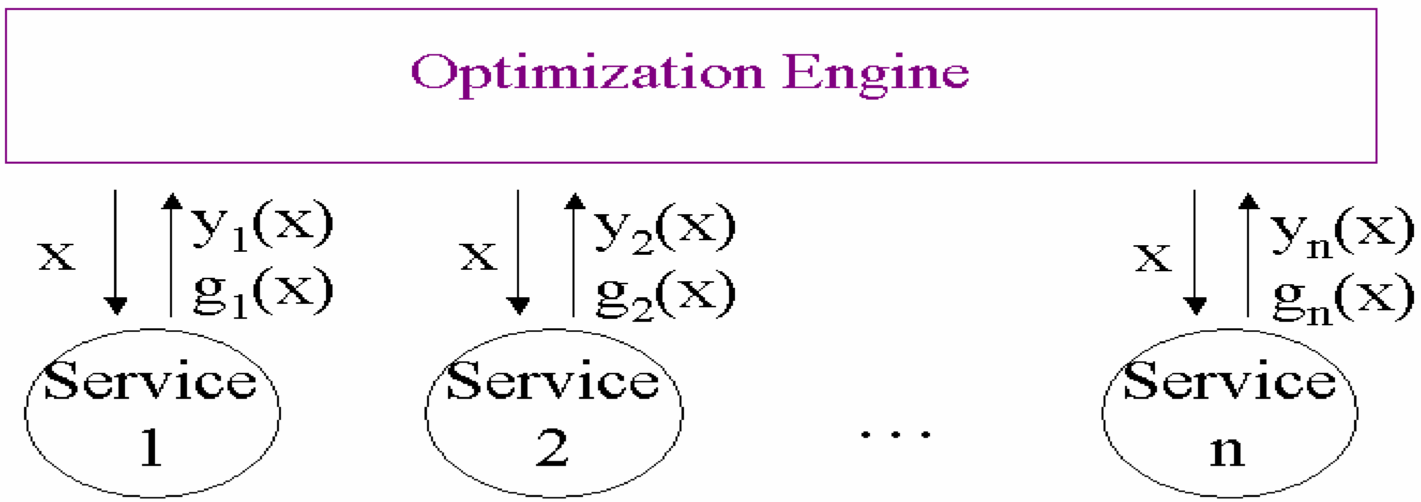
ampl: model diet.mod;
ampl: data diet.dat;
ampl: option solver kestrel;
ampl: option kestrel_options ‘solver=minos’;
ampl: solve;

Motorola Intelligent Optimization System

Data Flow and Knowledge Flow

$$\min \quad f(y_1(x), y_2(x), \ldots, y_n(x))$$

$$\text{s.t. } g_j(x) <= 0 \text{ for all } j = 1,\ldots,n$$



Motorola Intelligent Optimization System

simulation

$$T = T_s \times LF(t) + DT$$

T_s = Service time for a given server;

LF(t) = Load factor as a function of time (t);

DT= Down time.

Three kinds of services with typical behaviors are identified:

Service A:

T_s = Uniform distribution [6, 30] seconds;

LF(t) = 2.0 from 0800 to 1700 hours; 1.0 otherwise;

DT = 5% probability of the service going down for 30 seconds.

This service has automatic “crash detection” and recovery; therefore, the maximum down time is 30 seconds.

Service B:

T_s = Uniform distribution [30, 60] seconds;

LF(t) = 1.25 from 0600 to 1400 hours; 1.0 otherwise;

DT = Insignificantly small;

Service C:

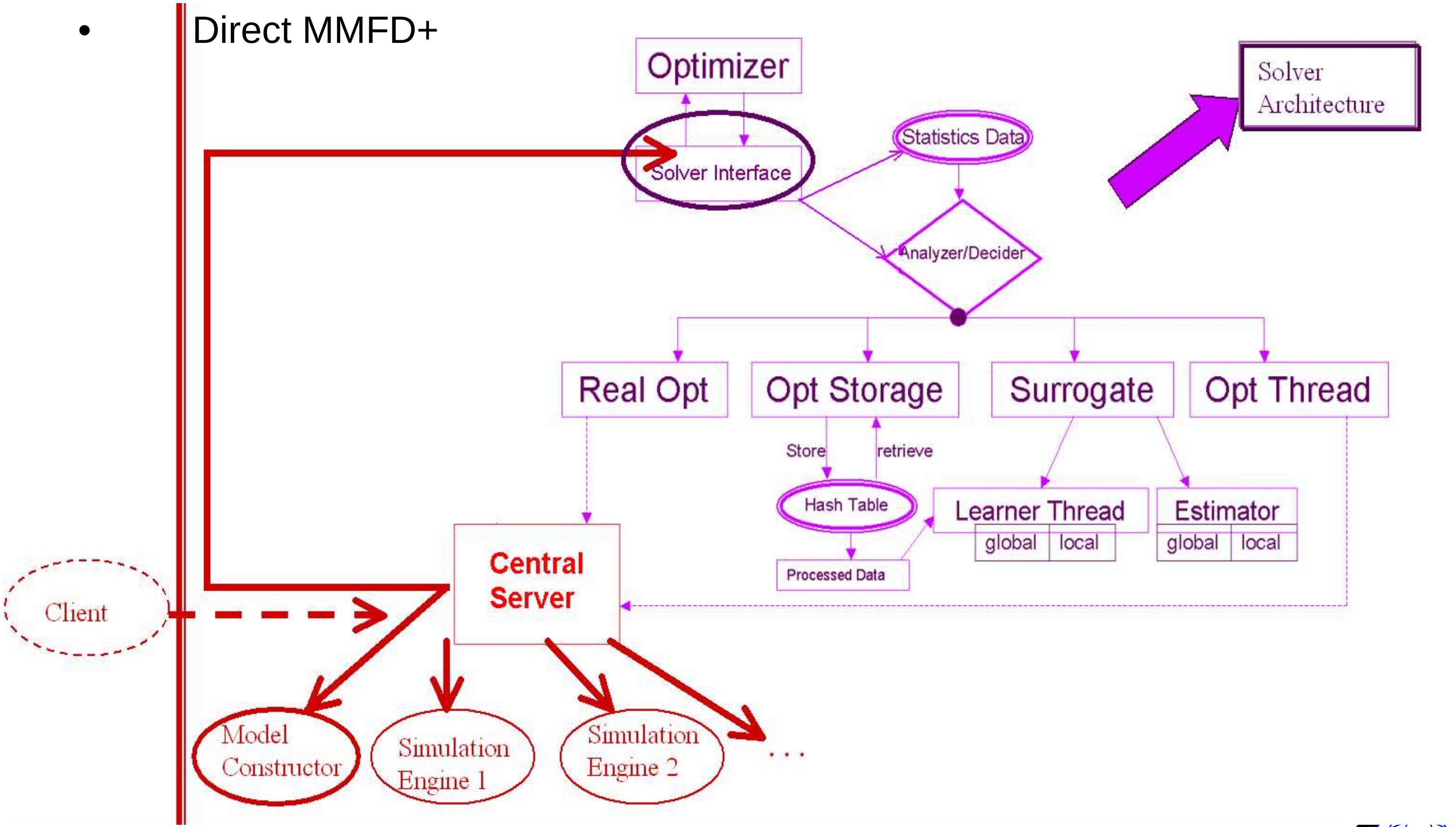
T_s = Uniform distribution [30, 90] seconds;

LF(t) = 2.0 from 0800 to 1700 hours; 1.0 otherwise;

DT = 1% probability of the service going down for anywhere between 15 minutes and 16 hours.

Motorola Intelligent Optimization System
optimization

- MFD
- MFD+
- Direct MMFD
- Direct MMFD+



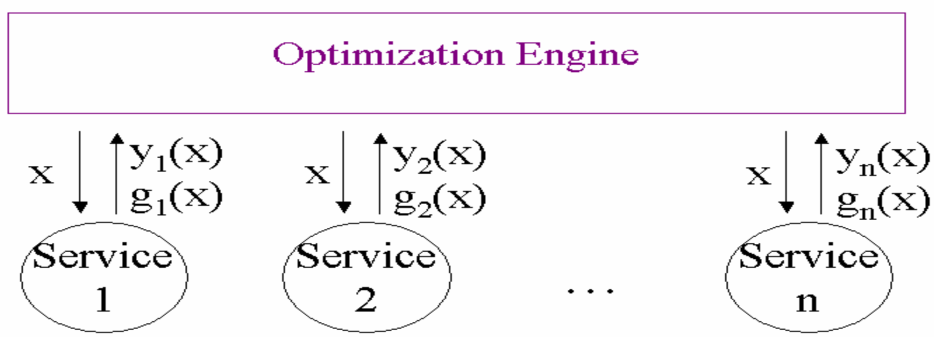
Motorola Intelligent Optimization System
learning and approximation

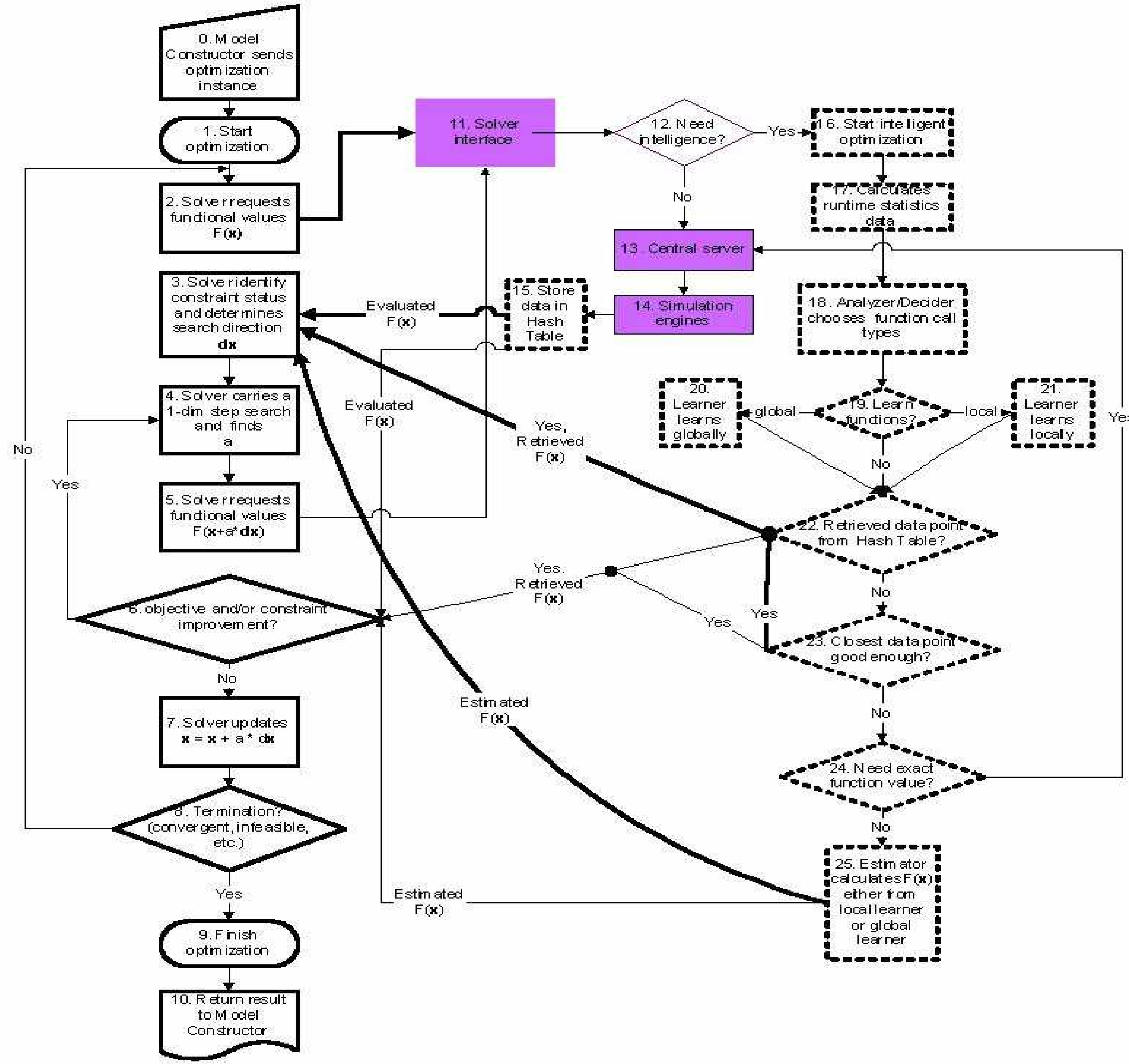
- Simple fitting
- 3-Layer neural network
- Gene expression programming
- Generalized neural network

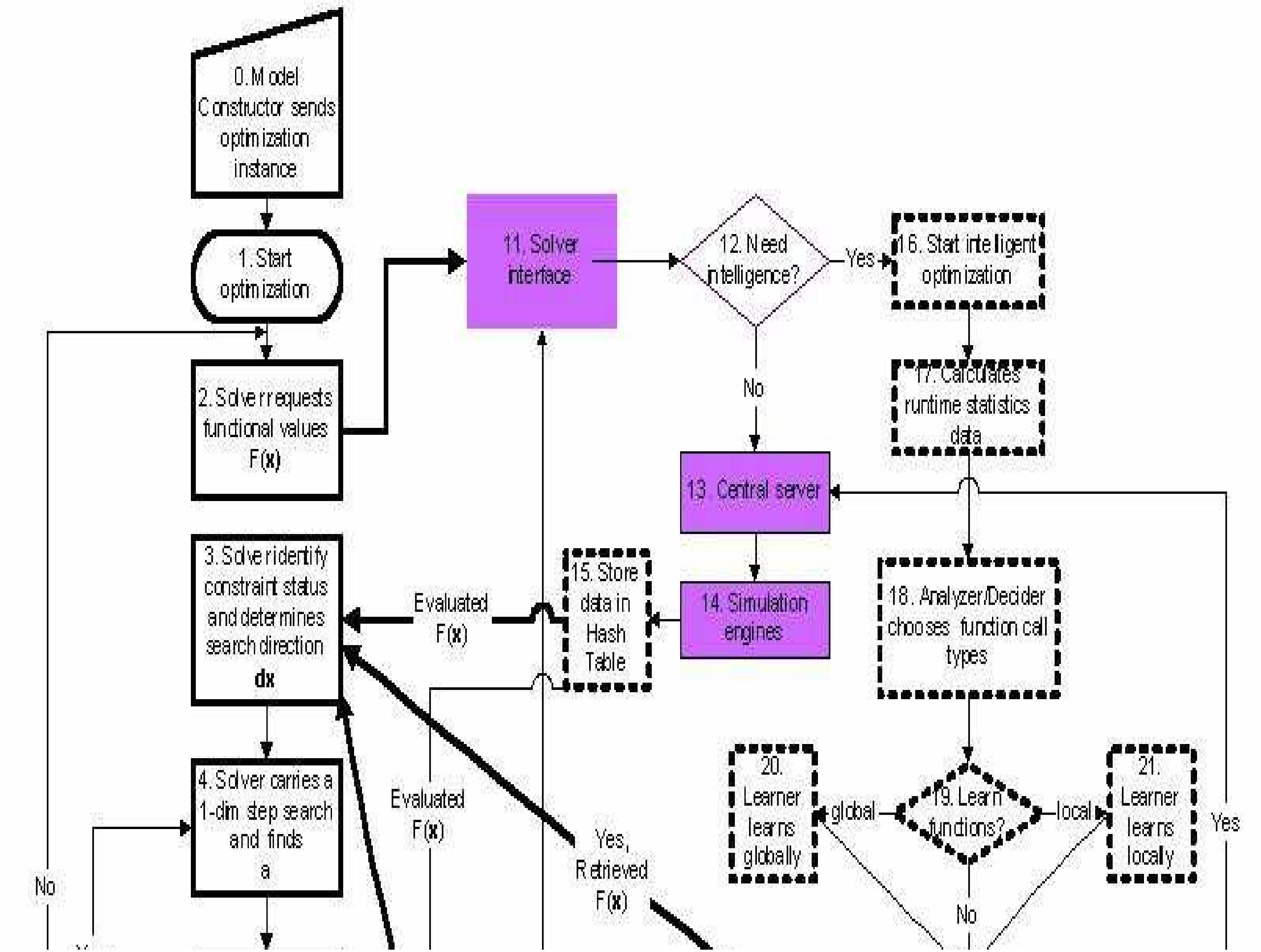
Motorola Intelligent Optimization System issues

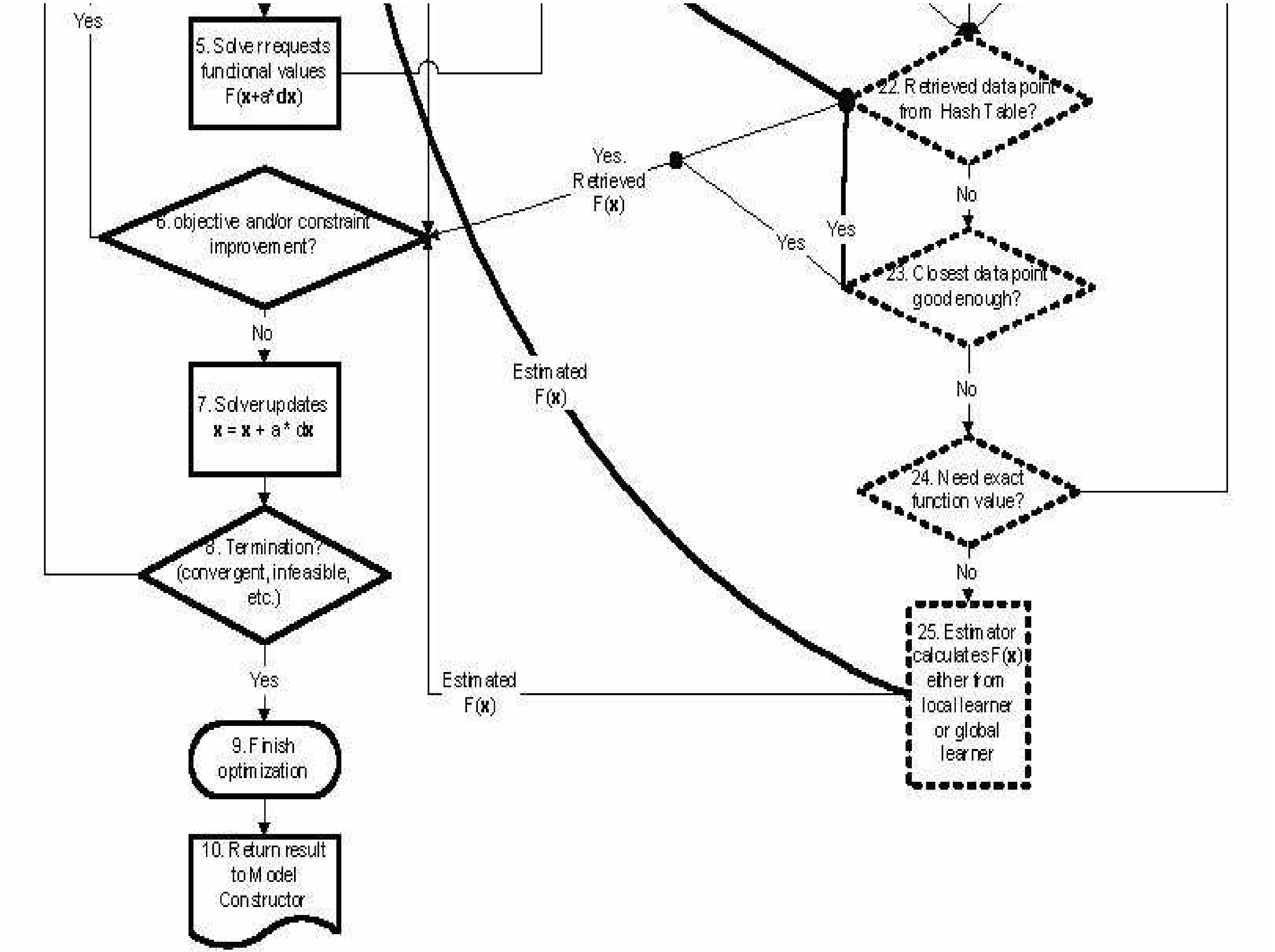
1) Initial Design Generation
2) Common Variable Resolution
3) Objective Construction
4) Constraint Enforcement
5) Result Interpretation
6) Process Coordination
7) Queue/Sequence Arrangement
8) Input Parsing/Output Reporting

$$\min \quad f(y_1(x),\, y_2(x),\, \ldots,\, y_n(x))$$
$$\text{s.t. } g_j(x) <= 0 \text{ for all } j = 1,\ldots,n$$




0. Model Constructor sends optimization instance
1. Start optimization
2. Solver requests functional values F(x)
3. Solver identify constraint status and determines search direction dx
4. Solver carries a 1-dim step search and finds a
5. Solver requests functional values F(x+a*dx)
6. objective and/or constraint improvement?
7. Solver updates x = x + a * dx
8. Termination? (convergent, infeasible, etc.)
9. Finish optimization
10. Return result to Model Constructor
11. Solver interface
12. Need intelligence?
13. Central server
14. Simulation engines
15. Store data in Hash Table
16. Start intelligent optimization
17. Calculates runtime statistics data
18. Analyzer/Decider chooses function call types
19. Learn functions?
20. Learner learns globally
21. Learner learns locally
22. Retrieved data point from Hash Table?
23. Closest data point good enough?
24. Need exact function value?
25. Estimator calculates F(x) either from local learner or global learner
Yes
No
global
local
Evaluated F(x)
Yes, Retrieved F(x)
Estimated F(x)


0. Model Constructor sends optimization instance
1. Start optimization
2. Solver requests functional values F(x)
3. Solver identify constraint status and determines search direction dx
4. Solver carries a 1-dim step search and finds a
11. Solver interface
12. Need intelligence?
Yes
No
13. Central server
14. Simulation engines
15. Store data in Hash Table
16. Start intelligent optimization
17. Calculates runtime statistics data
18. Analyzer/Decider chooses function call types
19. Learn functions?
global
local
No
20. Learner learns globally
21. Learner learns locally
Yes
Evaluated F(x)
Evaluated F(x)
Yes, Retrieved F(x)
No



Yes
5. Solver requests functional values F(x+a*dx)
6. objective and/or constraint improvement?
No
7. Solver updates x = x + a * dx
8. Termination? (convergent, infeasible, etc.)
Yes
9. Finish optimization
10. Return result to Model Constructor
Yes. Retrieved F(x)
Estimated F(x)
Estimated F(x)
22. Retrieved data point from Hash Table?
No
Yes
Yes
23. Closest data point good enough?
No
24. Need exact function value?
No
25. Estimator calculates F(x) either from local learner or global learner

Motorola Intelligent Optimization System

benchmark

service type	MFD	MFD+	Direct MMFD	Direct MMFD+
A	X	X	X	X
B	623	137	310	110
C	X	X	X	X
A+B	X	X	X	X
A+C	X	X	X	X
B+C	X	X	X	X
A+B+C	X	X	X	X

intelligent optimization flow (w/ simple 3-layer neural netowrk learning)

service type	MFD	MFD+	Direct MMFD	Direct MMFD+
A	619	132	376	78
B	645	287	389	172
C	>1500	>1500	422	192
A+B	641	212	358	142
A+C	1231	>1500	401	>1500
B+C	908	333	385	180
A+B+C	1147	324	>1500	202

intelligent optimization flow (w/ gene expression programming learning)

service type	MFD	MFD+	Direct MMFD	Direct MMFD+
A	343	71	210	40
B	360	160	215	91
C	>1500	>1500	230	106
A+B	361	118	190	79
A+C	>1500	190	210	92
B+C	480	846	202	93
A+B+C	647	165	273	114

intelligent optimization flow (w/ an advanced generalized neural network learning)

service type	MFD	MFD+	Direct MMFD	Direct MMFD+
A	182	66	93	49
B	204	87	108	42
C	>1500	1452	105	54
A+B	165	87	92	37
A+C	1002	487	145	49
B+C	229	132	123	45
A+B+C	293	145	123	67

Motorola Intelligent Optimization System

benchmark

- Without "Intelligence" (learning + approximation) : slow or crash.
- Optimization takes longer when simulations take longer, but usually correlates with the simulation that takes the longest, not the number of simulations.
- Direct methods works.
- Intensive linear search helps even more significantly, because it takes much less time than finding direction.
- Direct methods + intensive line search is the best.
- With "Intelligence": erratic but robust.
- Leaning helps: function behavior of simulation not as irregular as benchmark problems.
- Speed and quality of learning algorithms matter significantly.
- Combination of simulation may sometimes help.
- Quality of solutions does not matter too much, partly due to final stage fine tuning and safeguard for convergence, partly due to "good" behavior of simulation function forms, and partly due to high tolerance for termination.
- Curse of dimensionality is still an issue (variable number is around 10-15): good learning algorithms robust in high dimension can help.

Conclusion

- Optimization system history and background (linear programming, matrix generator, modeling language, optimization server, optimization services)
- System architecture and components (model, MLE, representation, interface/agent, server/registry, analyzer, solver, simulation)
- AMPL standalone and AMPL-NEOS architectures (You can still do your homework with 300+ variables in AMPL – “Kestrel Solver”)
- Motorola Intelligent Optimization System (real world is different from text book)
- System approach to simulation optimization
  - Direct methods help
  - Accurate line search help
  - Learning algorithm can help