Soar Technology
Thinking *inside* the box.

SoarML: A Graphical Modeling Language for Agents

Glenn Taylor, Jacob Crossman
glenn@soartech.com, jcrossman@soartech.com

25 May 2006

www.soartech.com

What is SoarML?

- A **visual language** for representing single agent designs
  - Based roughly on the Prometheus agent design methodology
  - Customized for human behavior modeling
- Initially developed as part of HLSR effort
  - Iteratively improved over two years
- Allows **code/architecture independent** descriptions of an agent's behavior
  - In general, is NOT specific to Soar
  - Was used initially to document HLSR designs
- Used for a several Soar Technology systems
  - Adversarial reasoning module
  - Indirect Fire (IF)-Soar
  - Deontics additions to Command and Control
  - AutoATC

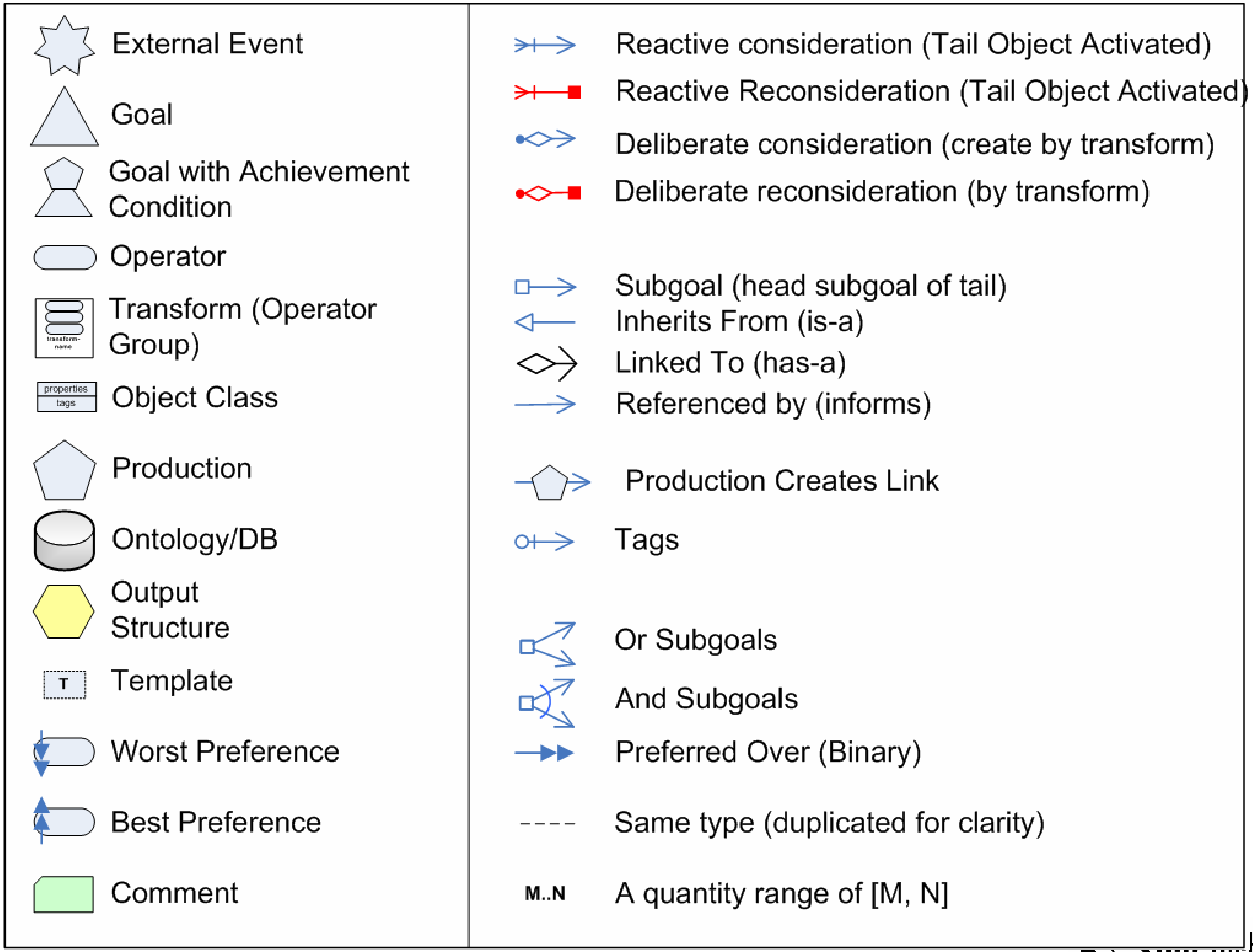
Motivation for a Modeling Language for Agents

- **Promotes High-level design**
  - Almost always better to think through design before coding
  - Text and code are not always best ways to encode designs
  - A modeling provides constructs that map to design concepts and ignore low-level details
- **Communication to Management**
  - PI needs a way to express/understand what is going on in an agent without looking at code
  - Customers sometimes need design documentation or key algorithms/processes explained
- **Communication within a development team**
  - Understanding what is happening in a Soar program is hard
  - Understanding is easier when the high-level concept is clear before looking at the code

Another Modeling Language?

- **Others exist: Why invent our own?**
  - Existing methods: OO UML, AUML, Prometheus
  - **But:**
  - Most agent MLs focus on multi-agent aspects, little detail at the individual agent level
  - None capture *cognitive architecture* aspects (goals, truth maintenance, deliberate consideration, preferences, etc.)
  - In many cases UML is helpful to cover other areas; SoarML focus is on areas UML doesn't cover well for agents

Graphical Design Language Key



External Event

Goal

Goal with Achievement Condition

Operator

Transform (Operator Group)

Object Class

Production

Ontology/DB

Output Structure

Template

Worst Preference

Best Preference

Comment

Reactive consideration (Tail Object Activated)

Reactive Reconsideration (Tail Object Activated)

Deliberate consideration (create by transform)

Deliberate reconsideration (by transform)

Subgoal (head subgoal of tail)

Inherits From (is-a)

Linked To (has-a)

Referenced by (informs)

Production Creates Link

Tags

Or Subgoals

And Subgoals

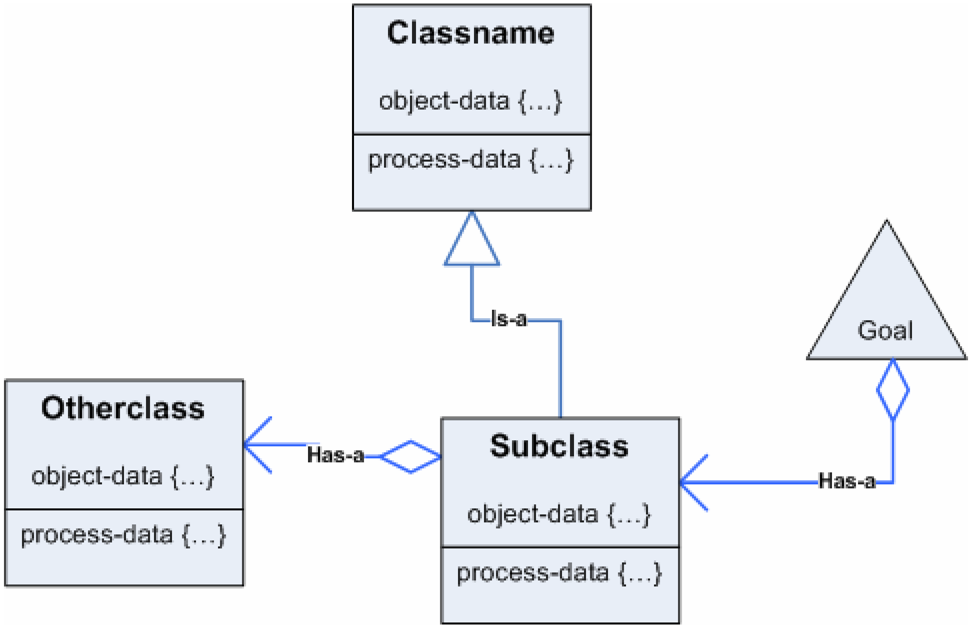
Preferred Over (Binary)

Same type (duplicated for clarity)

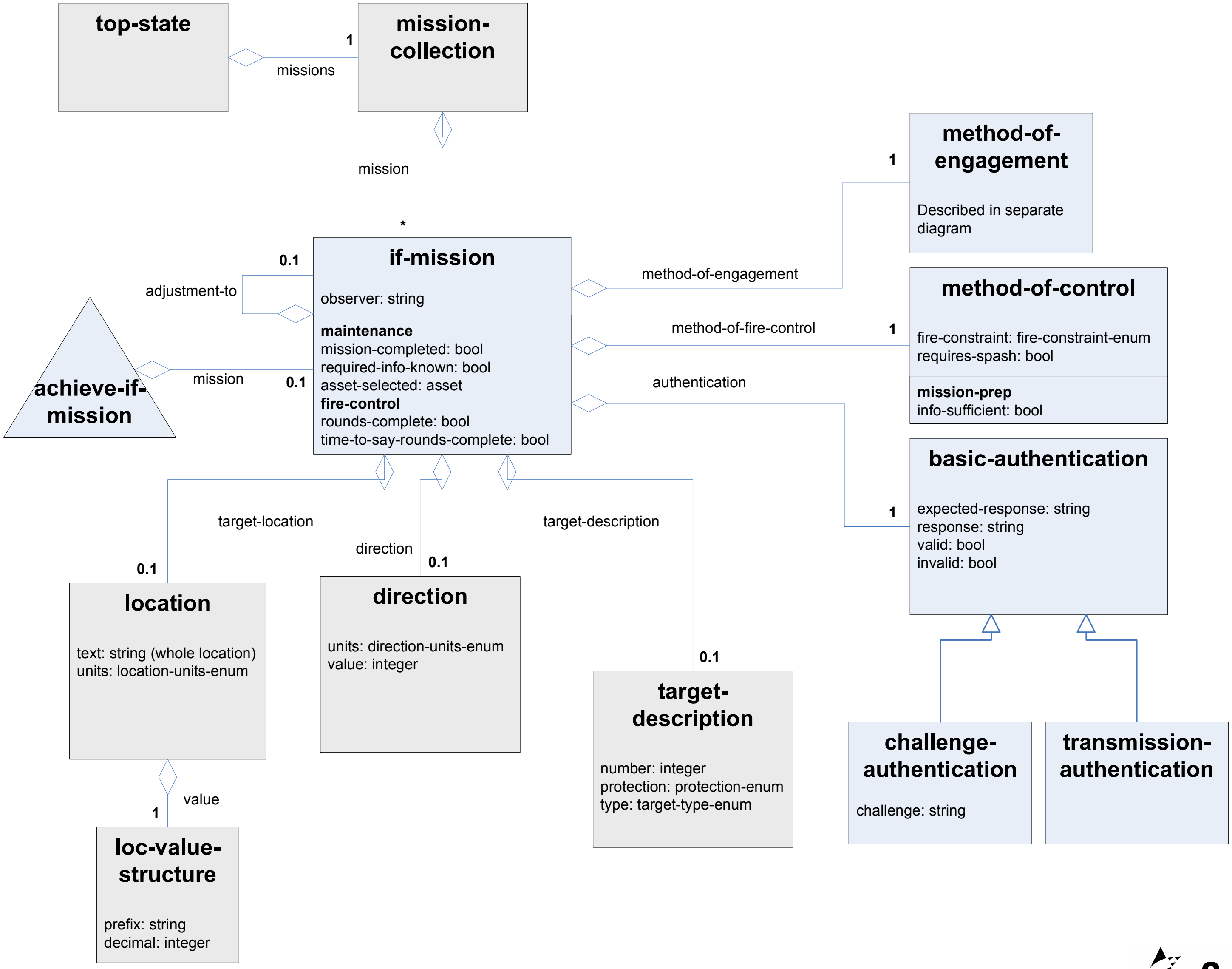
A quantity range of [M, N]

Static Structure Diagram Examples

- Description
  - Representation of declarative memory's structure
  - Consistent with OO UML specifications
  - Tagging separates process-centric data (usually only shown in process diagrams)
- Notes
  - Can be used standalone or as part of process diagrams
  - Soar doesn't really directly support structures or inheritance



Knowledge Structure for Indirect Fire



top-state

mission-collection

method-of-engagement

Described in separate diagram

if-mission

observer: string

maintenance
mission-completed: bool
required-info-known: bool
asset-selected: asset
fire-control
rounds-complete: bool
time-to-say-rounds-complete: bool

achieve-if-mission

method-of-control

fire-constraint: fire-constraint-enum
requires-spash: bool

mission-prep
info-sufficient: bool

basic-authentication

expected-response: string
response: string
valid: bool
invalid: bool

location

text: string (whole location)
units: location-units-enum

direction

units: direction-units-enum
value: integer

target-description

number: integer
protection: protection-enum
type: target-type-enum

challenge-authentication

challenge: string

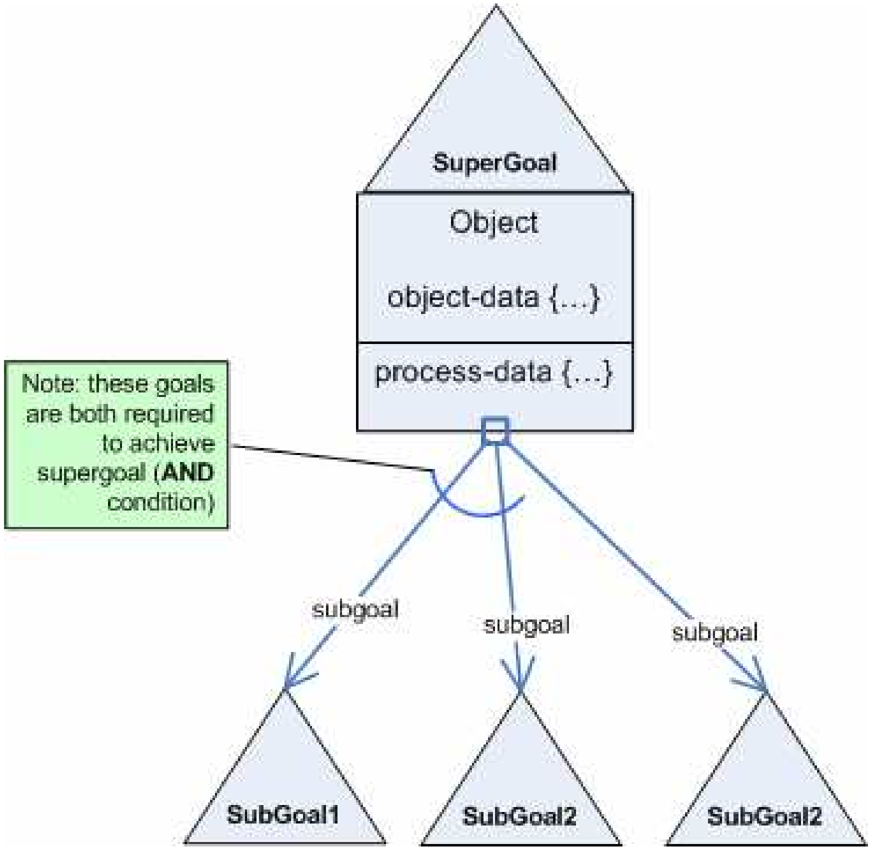
transmission-authentication

loc-value-structure

prefix: string
decimal: integer

Goals and Goal Hierarchies

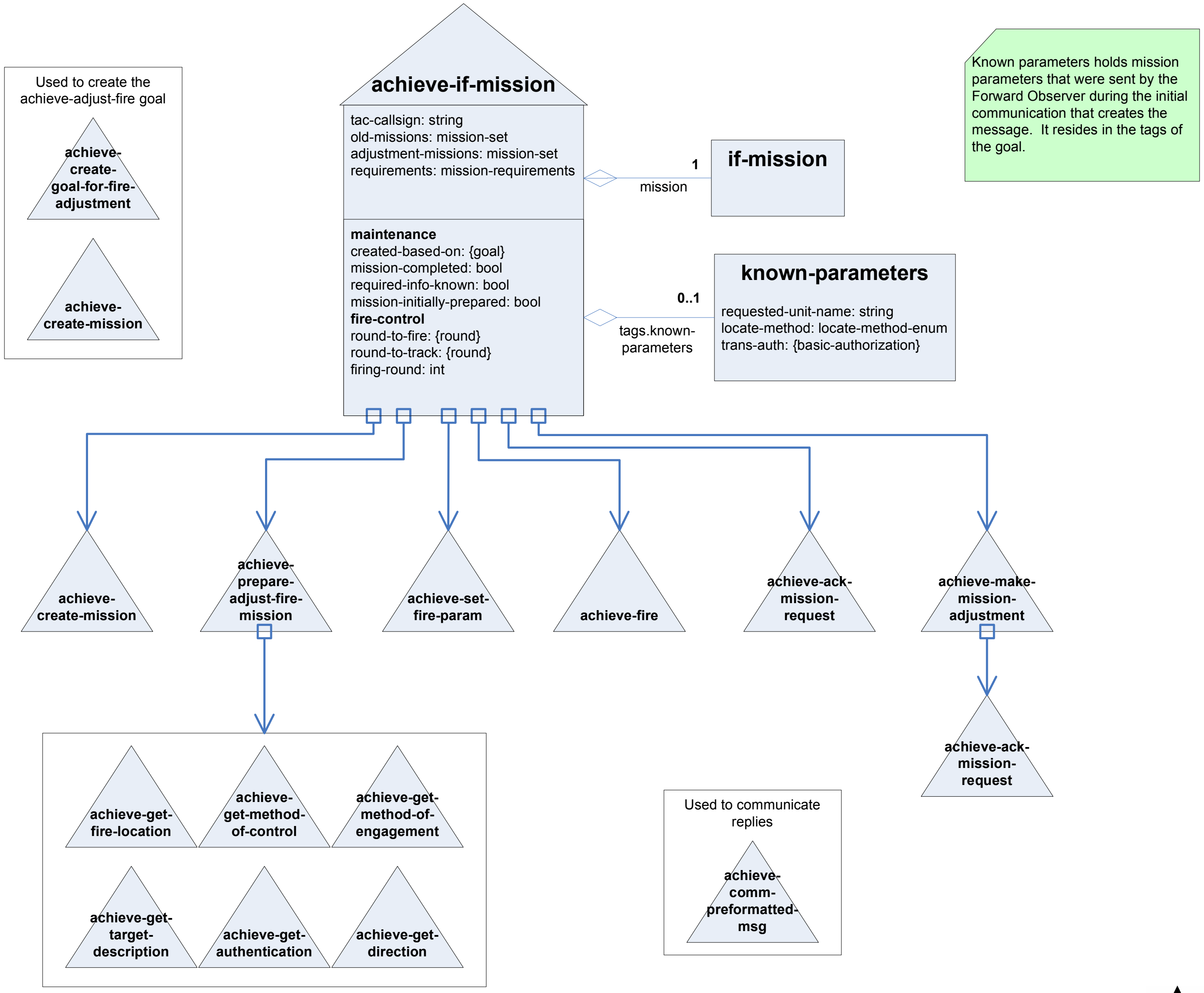
- Description
  - Represent goal hierarchies
  - Supports forests or stacks
  - Does not require any specific Soar implementation (e.g. using impasses or top-state goals)
  - Can augment with “met” condition: a production that marks the goal “achieved”
- Notes
  - Can be used to show goal forest or connection to goals in process diagrams



Note: these goals are both required to achieve supergoal (**AND** condition)

Soar Technology
Thinking *inside* the box.

Goal Forest for Indirect Fire



Used to create the achieve-adjust-fire goal

achieve-create-goal-for-fire-adjustment

achieve-create-mission

achieve-if-mission

tac-callsign: string
old-missions: mission-set
adjustment-missions: mission-set
requirements: mission-requirements

maintenance
created-based-on: {goal}
mission-completed: bool
required-info-known: bool
mission-initially-prepared: bool
fire-control
round-to-fire: {round}
round-to-track: {round}
firing-round: int

1
mission

if-mission

0..1
tags.known-parameters

known-parameters

requested-unit-name: string
locate-method: locate-method-enum
trans-auth: {basic-authorization}

Known parameters holds mission parameters that were sent by the Forward Observer during the initial communication that creates the message. It resides in the tags of the goal.

achieve-create-mission

achieve-prepare-adjust-fire-mission

achieve-set-fire-param

achieve-fire

achieve-ack-mission-request

achieve-make-mission-adjustment

achieve-get-fire-location

achieve-get-method-of-control

achieve-get-method-of-engagement

achieve-get-target-description

achieve-get-authentication

achieve-get-direction

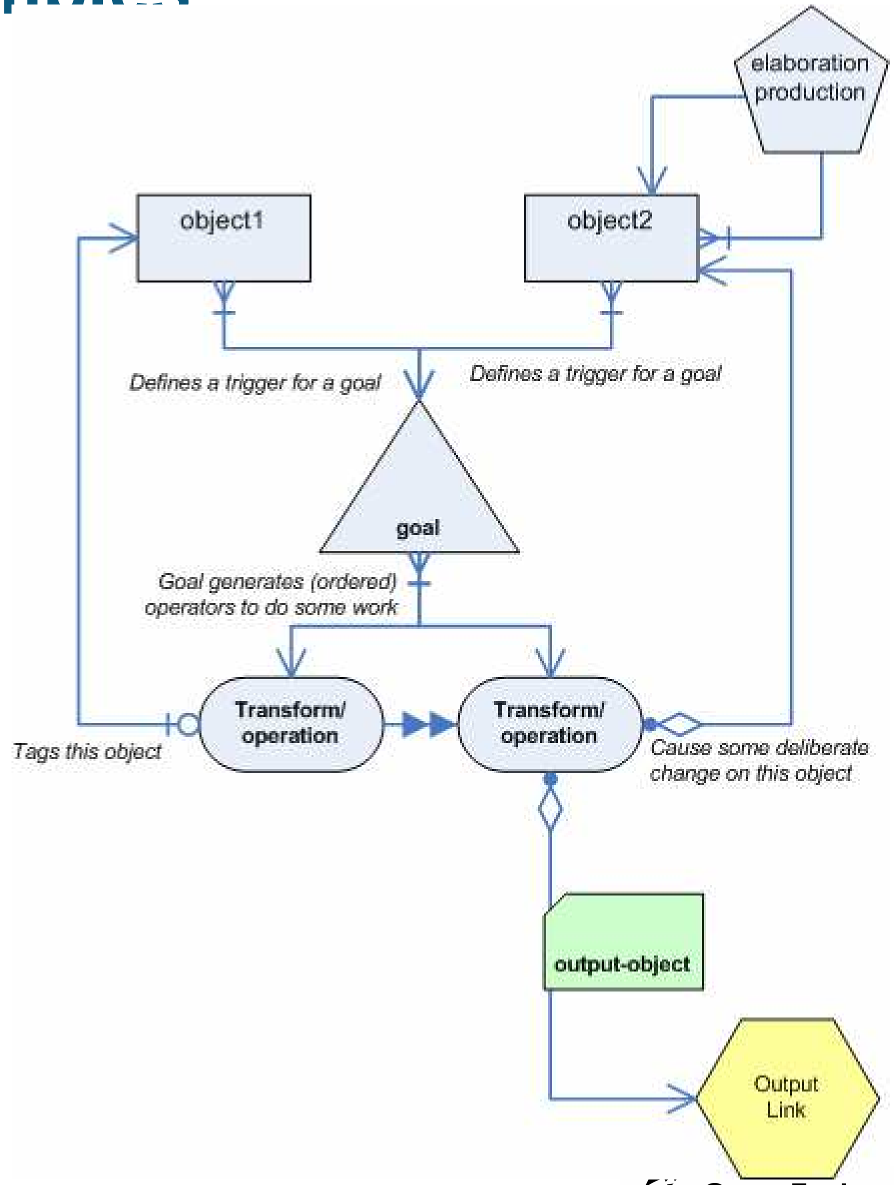
Used to communicate replies

achieve-comm-preformatted-msg

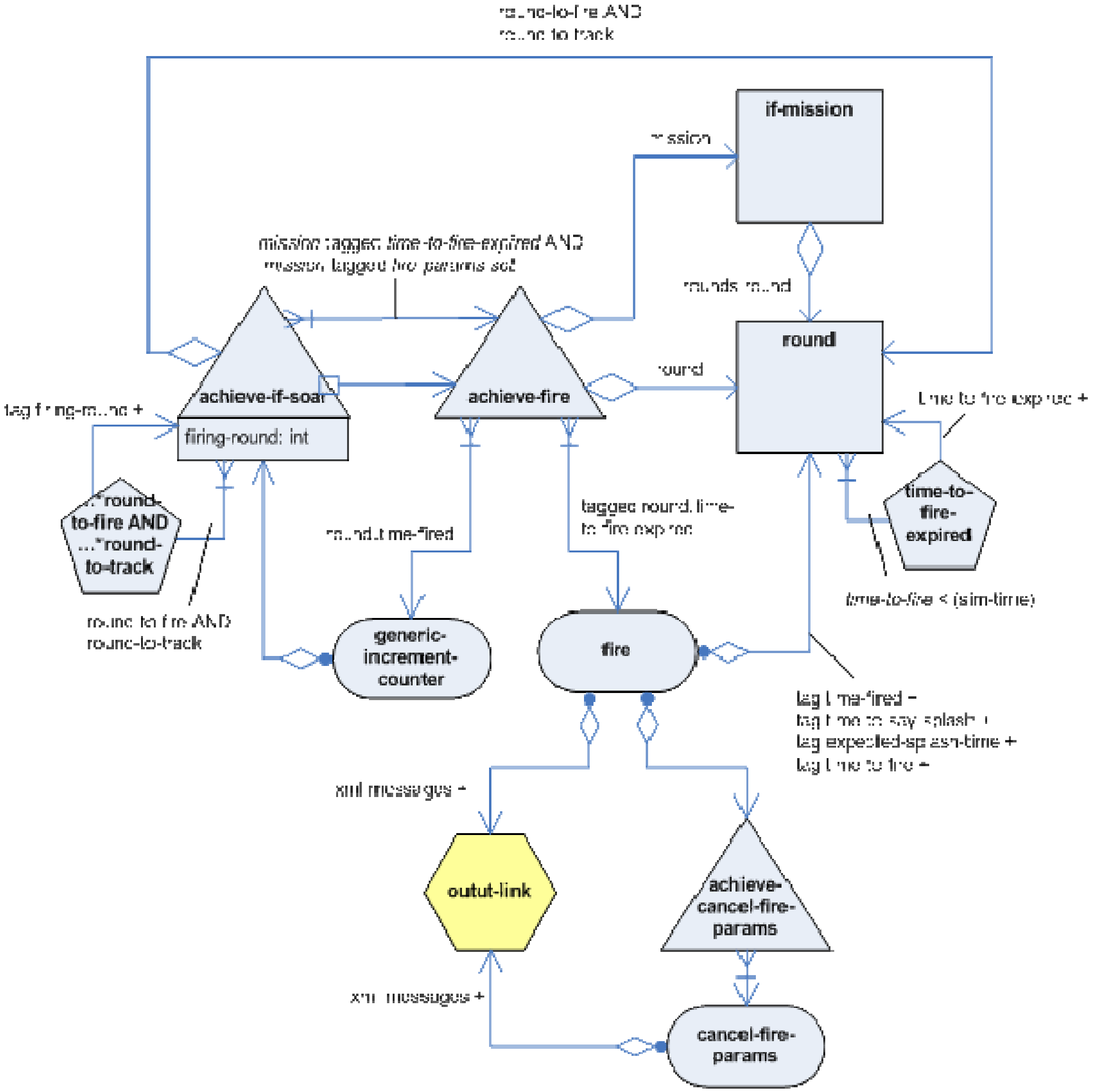
achieve-ack-mission-request

Process Diagram Examples

- Description
  - Represent processes: sets of related operators in Soar
  - Integrates static structure, goals, operators, and preferences
  - Key productions can be highlighted
  - Includes key memory changes and trigger conditions
- Notes
  - Typically processed are documented to the operator level


elaboration production
object1
object2
Defines a trigger for a goal
Defines a trigger for a goal
goal
Goal generates (ordered) operators to do some work
Transform/ operation
Transform/ operation
Tags this object
Cause some deliberate change on this object
output-object
Output Link

The Firing Process for Indirect Fire



When/How to Use

- **Documenting Design Concepts**
  - Purpose:
    - construct and analyze the framework for an agent’s behavior
    - understand major behavior interactions and knowledge structures
  - Guidelines
    - Most effective early in a project
    - Focus on major objects, processes, and relationships
    - Keep abstract: implementation will refine and suggest changes
- **Document Existing Systems**
  - Purpose:
    - Provide an overview of system for maintenance team
    - Provide customer/management with technical details
  - Guidelines
    - Most effective late in development cycle after details solidify
    - Focus on key patterns of behavior and concepts to understand how the agent behaves and how it can be modified
    - Drill down to moderate levels of detail (e.g. provide more firing conditions and knowledge structure details)

Nuggets/Coal

Nuggets

- Useful for design documentation and presentations
- Being used on several projects
- A good way to visually inspect design for flaws/commonalities

Coal

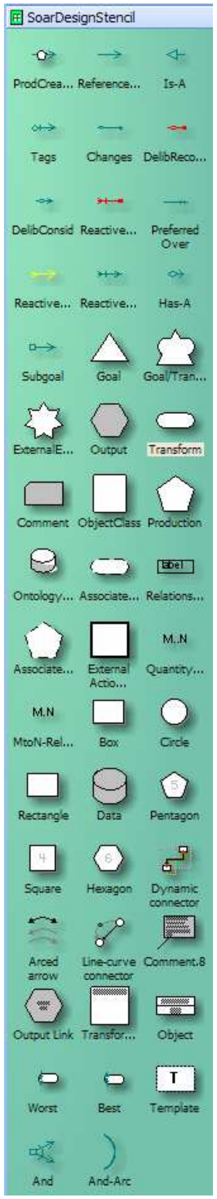
- Hard to get some engineers to design and document
- Only a few people using it regularly
- Doesn’t address multi-agent processes (other MLs might cover this sufficiently)

References

- For Visio Stencil, email Glenn or Jacob
- Prometheus

Padgham, L. and Winikoff, M., Prometheus: A Methodology for Developing Intelligent Agents, Proceedings of the Third International Workshop on AgentOriented Software Engineering, at AAMAS 2002. July, 2002, Bologna, Italy

http://www.cs.rmit.edu.au/agents/prometheus/



Soar Technology
Thinking *inside* the box.

Questions?