OpenNARS 2.0.0 System Design

Introduction

OpenNARS 2.0.0 is a significant redesign of the NARS system architecture. The design is focused to achieve several key improvements over previous versions:

1. An architecture to support a distributed processing model
2. Better support for multicore processing
3. Message passing framework to isolate components (Actor model)
4. Enhanced control capabilities (throughput and timing)
5. Separation of concerns into three key aspects
    a. Perception/Action Driven Input
    b. Memory Management System
    c. General Inference Cycle
6. Redesigned local inference:
    a. Anticipation handling
    b. Task management
    c. Supporting general concept data structure
7. Introduction of temporal concepts
    a. Supporting temporal chaining via general inference
    b. Enhanced perception sequence formation

Development Stack

Whilst the design is agnostic in regard to which tools, languages and frameworks are used to implement it, there are some consideration that should be highlighted along with our choice of tools for the stack.

The decision was made that a functional language would be the best choice for the design implementation. Key factors in this decision were, immutability, composability, testablility and conciseness of representation. After careful review Clojure was selected as the language of choice, one of the key reasons being its flexibility from its macro capability and the feature of Lisp style languages to treat "code as data".

The actor model was chosen, for reasons highlighted elsewhere, and the framework of choice was the parallel universe pulsar implementation. There were several factors involved in the decision: pulsar has a native Clojure API, it uses an instrumentation injection approach with the JVM to enhance performance, and finally it has a distributed capability from the Galaxy IMDG which supports dynamic actor migration, based on 'closeness'of data.

For an IDE we opted for Intellij IDEA with the Cursive plugin.

Github is used for the source repository.

All components of the development stack are open source and free for use (in open source development projects).

System Overview (Actor Framework)

The overall system design is composed of four sections: Perception/Action driven input, memory management system, general inference cycle and Display. The system is defined within a message passing framework, implemented as an actor system. The figure 1 shows the four core aspects of the system, complete with their respective actors and message flow.

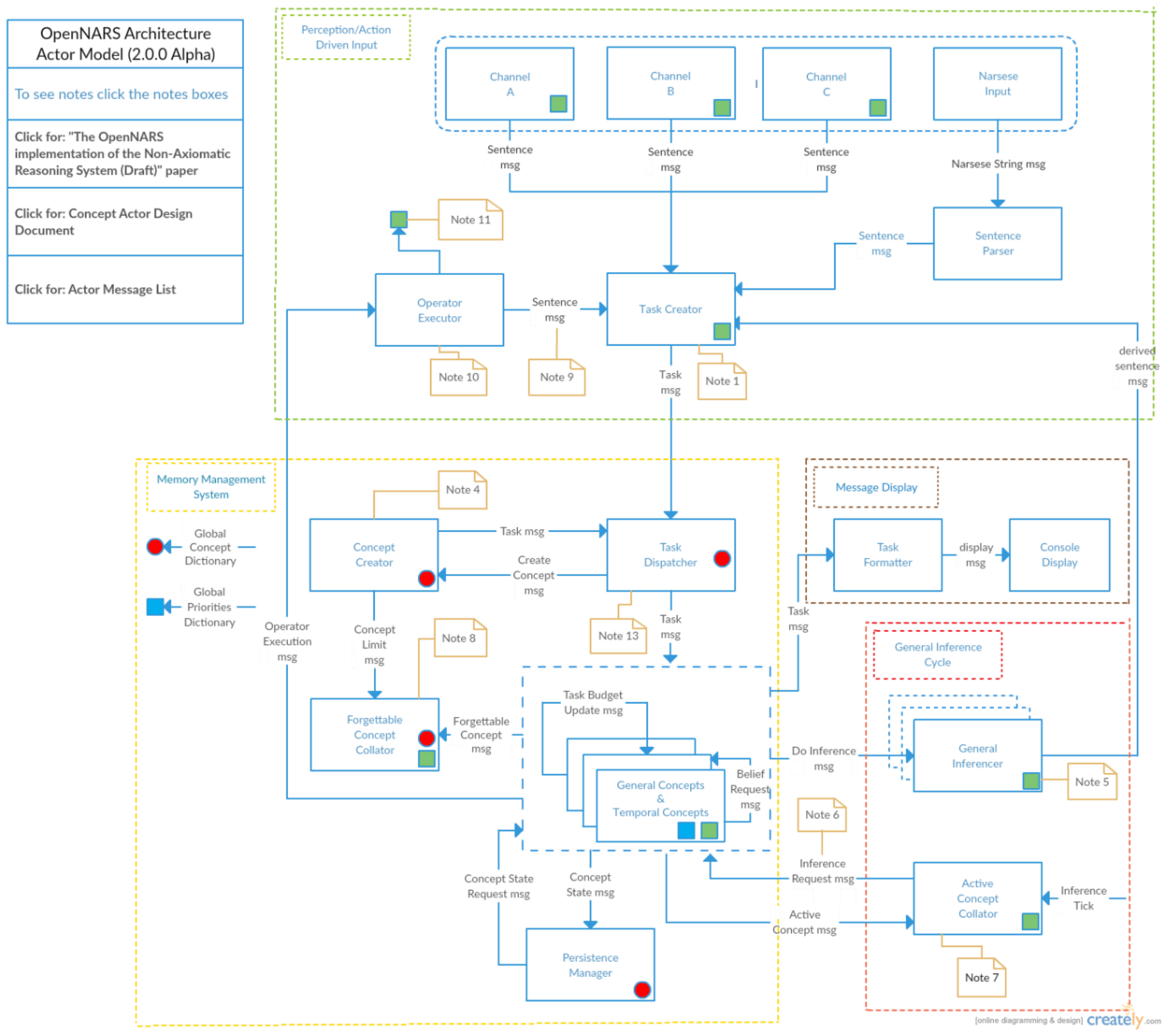

Figure 1- System Overview

Additional notes:

1. The green squares represent actors that can be modified by introspective operators.
2. The red circles represent a requirement to access the global concept actor hash table. In future updates this hash table will be replaced with an In-Memory Data Grid (IMDG).
3. The blue square represents access to the global concept priorities dictionary. This holds the current priority of each concept actor and is updated for each task processed by a concept.
4. Multiple Narsese input channels are supported by a task source property, which defines, among other things, the origin of the statement. This can be another NARS unit or multiple users.

System Actor Overview

Actor Listing

ID	Actor name	System Aspect
1	Sentence Parser	Perception/Action Driven Input
2	Task Creator	Perception/Action Driven Input
3	Operator Executor	Perception/Action Driven Input
4	Task Dispatcher	Memory Management System
5	Concept Creator	Memory Management System
6	General Concept	Memory Management System
7	Temporal Concept	Memory Management System
8	Forgettable Concept Collator	Memory Management System
9	Persistence Manager	Memory Management System
10	General Inferencer	General Inference Cycle
11	Active Concept Collator	General Inference Cycle
12	Task Formatter	Message Display
13	Console Display	Message Display

Actor Descriptions

Actor name	Description
Sentence Parser	The Sentence Parser accepts a string representation of a Narsese statement and parses a valid sentence or generates a parser error to errout if invalid.
Task Creator	The Task Creator accepts a valid sentence or derived sentence, then sets the source property based on origin, adds a serial-no and converts tense to occurrence time.
Operator Executor	The Operator Executor accepts an operation execution request, and a parameter list to apply to the operator if applicable. The Operator Executor maintains a list of registered operators and executes the relevant operator on receipt of a :operator-execution-msg. Proprioceptive responses are generated when required by the respective operators and sent to task creator.
Task Dispatcher	The Task Dispatcher accepts tasks and dispatches them to their respective concepts. If a required concept does not exist, the Task Dispatcher makes a request of the Concept Creator to create the necessary concepts. The task is then dispatched, once it is returned from the Concept Creator.
Concept Creator	The Concept Creator is responsible for creating new concepts based on the :task-msg. Sub-term concept creation, along with task and budget propagation also occurs as part of the concept creation. Once the concept (tree) is created a :task-msg is passed back to the Task Dispatcher for processing now that the required concepts exist. Additionally, the Concept Creator is responsible for creating Temporal Concepts for input events. The concept Creator maintains a count of the concepts created, and if a predefined concept limit is reached, a :concept-limit-msg is posted to the Forgettable Concept collator, where concepts are forgotten based on lowest priority.

General Concept	General concepts have three key responsibilities: carry out local inference, provide tasks and beliefs for general inference and store concept related data.
Temporal Concept	Temporal concepts have two roles: maintain a concept budget and store events with the same occurrence time for selection by general inference.
Forgettable Concept Collator	When a concepts activation level falls below a dynamically specified threshold, it sends a :forgettable-concept-msg to the Forgettable Concept Collator. These are stored in a capped priority queue, where the lowest activation level is selected as the top of queue. When the actor receives a :concept-limit-msg it selects the top of queue for deletion. A count of : concept-limit-msg is maintained to ensure there is a deletion for each message received as it is possible that there is no concept available for deletion. In this case the dynamic deletion threshold is lowered. The queue is ordered based on $average(quality(concept), p(concept))$. See Budget Function (Forgetting) for additional details.
Persistence Manager	The Persistence Manager creates a persistent copy of the state of the systems concept structure to backing store. This is achieved by sending a :concept-state-request-msg to each concept stored in the global concept dictionary. The concepts in turn send their respective state back to the persistence manager, which then copies the state to backing store.
General Inferencer	General Inferencer consists of the NAL Meta Rule Trie and deriver. Multiple instances can be created to enable elastic processing of inference requests. The General Inferencer accepts a :do-inference-msg with its associated task and belief pair, generates the required derivations according to the inference rules, and forwards the results, as derived sentences, to the Task Creator
Active Concept Collator	Each inference cycle the Active Concept Collator collates the active concepts, where n concepts are selected via a selection function (pluggable) and :inference-request-msg(s) are posted to the selected concepts. The collection is then reset to empty, ready for the next cycle. There are two aspects to the selection process: general concepts are placed in a bag and a concept is selected using bag semantics. In the case of temporal concepts, a concepts activation is projected to the current moment, and then added to a temporal concept bag. A pair of temporal concepts are selected from the bag for temporal inference.
Task Formatter	Accepts a :task-msg and formats it as a display string and posts it as a :display_msg
Console Display	Accepts a :display-msg and displays it on the console

Design Notes

The following design notes should be used in conjunction with the system overview diagram, in the previous section, where notes are referred to by numbered note references. Please note that there is some redundancy with the actor description section above. The notes were retained in this form as they are referenced from the System Overview diagram.

Perception/Action Actors

These notes apply to the following actors: Sentence Parser, Operator Executor and Task Creator.

Note	Description
1	Creates task from a sentence, sets source property based on origin, adds serial-no and converts tense to occurrence time
9	Proprioceptive event sentence issued by respective operators when required
10	The Operator Executor maintains a list of registered operators and executes the relevant operator on receipt of a :operator-execution-msg. Proprioceptive responses are generated when required by the respective operators and sent to task creator.
11	Denotes that the actor can be influenced by operator control via message passing

Memory Management System Actors

These notes apply to the following actors: Task Dispatcher, Concept Creator, General Concepts, Temporal Concepts, Forgettable Concept Collator and Persistence Manager.

Note	Description
4	The Concept Creator is responsible for creating new concepts based on the :task-msg. Sub-term concept creation, along with task and budget propagation also occurs as part of the concept creation. This Concept Creator offloads some of the responsibility of the Task Dispatcher, which needs to be high throughput. Once the concept (tree) is created a :task-msg is passed back to the Task Dispatcher for processing now that the required concepts exist. Additionally, the Concept Creator is responsible for creating Temporal Concepts for input events. Where temporally related events are grouped and selected for inference in the general inference cycle
8	When a concepts activation level falls below a dynamically specified threshold, it sends a :forgettable-concept-msg to the Forgettable Concept Collator. These are stored in a priority queue, where the lowest activation level is selected as the top of queue. When the actor receives a : concept-limit-msg it selects the top of queue for deletion. A count of : concept-limit-msg is maintained to ensure there is a deletion for each message received as it is possible that there is no concept available for deletion. In this case the dynamic deletion threshold is lowered.
13	When a :task-msg arrives at Task Dispatcher the actor ref hashmap is checked to see it the task related concept exists. If the required concept does not exist the :task-msg is forwarded to the Concept Creator

General Inference Cycle Actors

These notes apply to the following actors: General Inferencer and Active Concept Collator.

Note	Description

5	General Inferencer consists of the NAL Meta Rule Trie and deriver. Multiple instances can be created to enable elastic processing of inference requests.
6	An :inference-request-msg is posted to the target concept. where a task and term are selected. This 'package' is then posted to the term designated concept as a :belief-request-msg. If the target concept is active then the :belief-request-msg is processed and a :do-inference-msg is sent to the General Inferencer, otherwise it is discarded (maybe influences budget TBD).
7	Each inference cycle the Active Concept Collator collects the active concepts in a collection, where n concepts are selected via a selection function (pluggable) and :inference-request-msg(s) are posted to the selected concepts. The collection is then reset to empty, ready for the next cycle.

Concepts

There are two forms of concept: general concepts and temporal concepts. Following is a description of the structure and message flow of each concept type.

General Concept

General concepts have three key responsibilities: carry out local inference, provide tasks and beliefs for general inference and store concept related data. The following diagram shows the data structure and general concept related message flow.

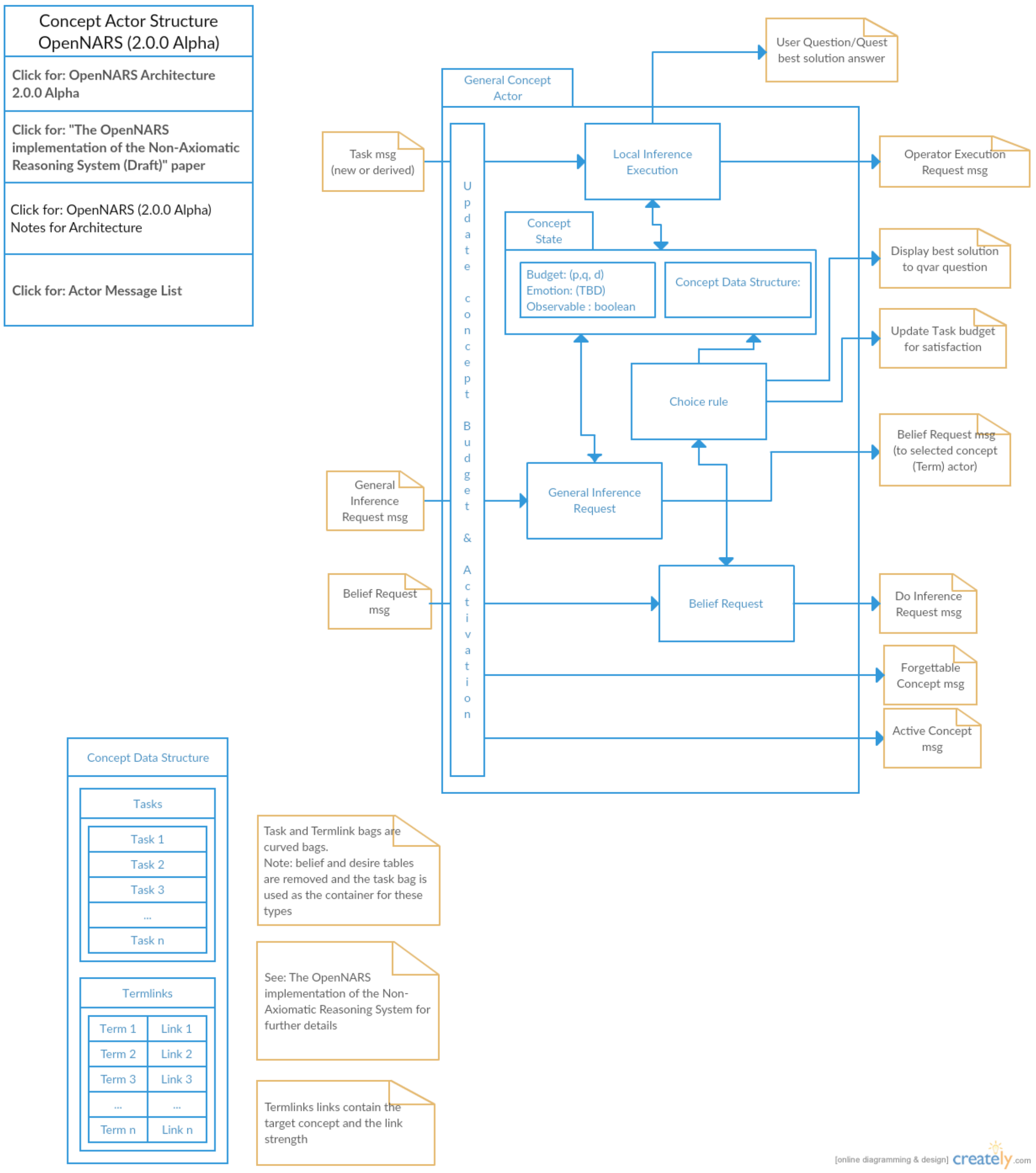
Concept Actor Structure OpenNARS (2.0.0 Alpha)
Click for: OpenNARS Architecture 2.0.0 Alpha
Click for: "The OpenNARS implementation of the Non-Axiomatic Reasoning System (Draft)" paper
Click for: OpenNARS (2.0.0 Alpha) Notes for Architecture
Click for: Actor Message List
User Question/Quest best solution answer
General Concept Actor
Task msg (new or derived)
Update concept Budget & Activation
Local Inference Execution
Operator Execution Request msg
Concept State
Budget: (p,q, d)
Emotion: (TBD)
Observable : boolean
Concept Data Structure:
Display best solution to qvar question
Update Task budget for satisfaction
Choice rule
Belief Request msg (to selected concept (Term) actor)
General Inference Request msg
General Inference Request
Belief Request msg
Belief Request
Do Inference Request msg
Forgettable Concept msg
Active Concept msg
Concept Data Structure
Tasks
Task 1
Task 2
Task 3
...
Task n
Termlinks
Term 1
Link 1
Term 2
Link 2
Term 3
Link 3
...
...
Term n
Link n
Task and Termlink bags are curved bags. Note: belief and desire tables are removed and the task bag is used as the container for these types
See: The OpenNARS implementation of the Non-Axiomatic Reasoning System for further details
Termlinks links contain the target concept and the link strength
[online diagramming & design] creately.com

Figure 2- General Concept

Temporal Concept

Temporal concepts are more specific than general concepts in that they are not required to support local inference. There primary role is to enable temporal chaining within general inference by enabling temporally related statements to be selected for temporal inference. The figure 3 shows the data structure and temporal concept related message flow.

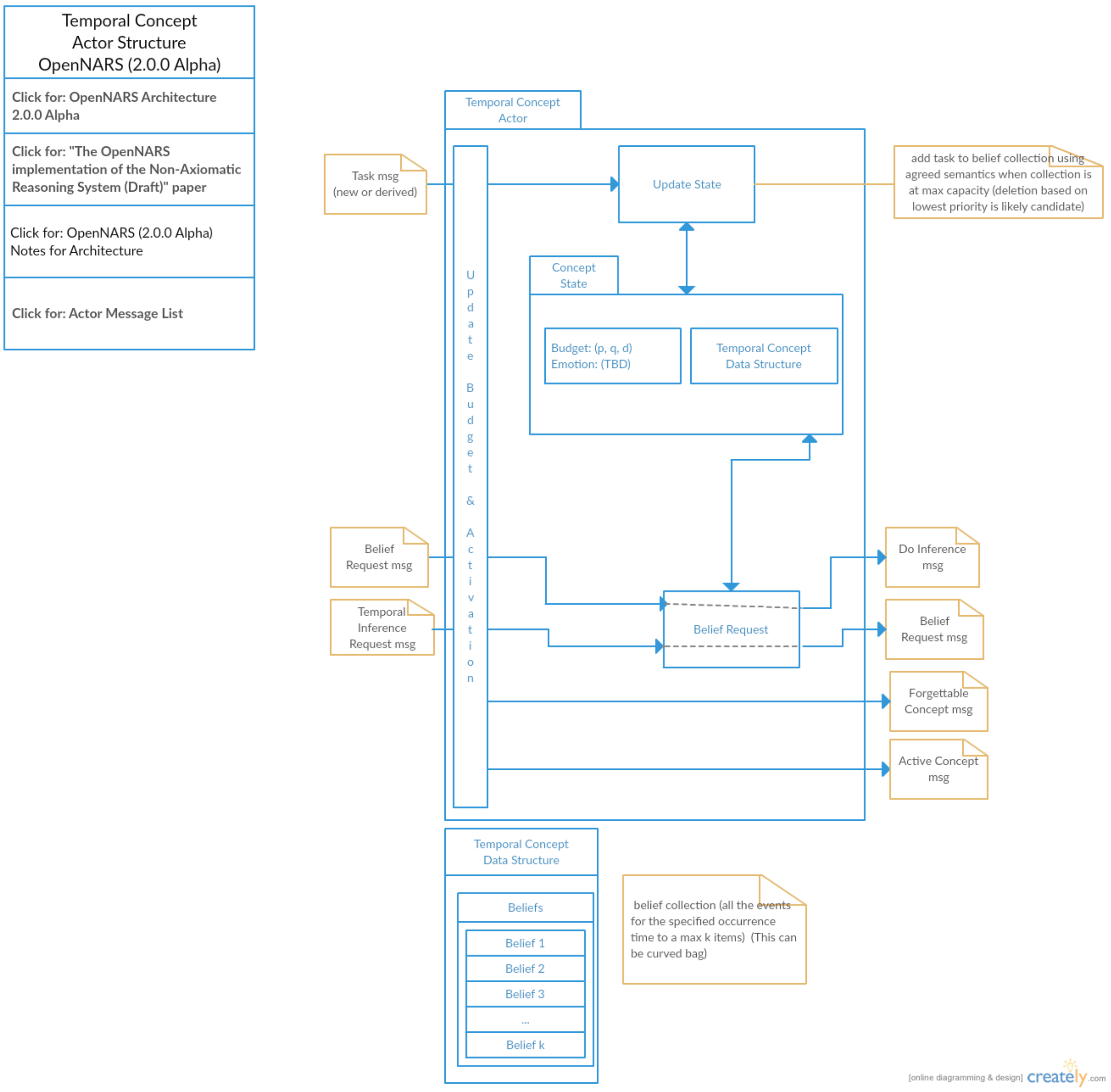
Temporal Concept Actor Structure OpenNARS (2.0.0 Alpha)
Click for: OpenNARS Architecture 2.0.0 Alpha
Click for: "The OpenNARS implementation of the Non-Axiomatic Reasoning System (Draft)" paper
Click for: OpenNARS (2.0.0 Alpha) Notes for Architecture
Click for: Actor Message List
Temporal Concept Actor
Task msg (new or derived)
Update State
add task to belief collection using agreed semantics when collection is at max capacity (deletion based on lowest priority is likely candidate)
Update Budget & Activation
Concept State
Budget: (p, q, d) Emotion: (TBD)
Temporal Concept Data Structure
Belief Request msg
Temporal Inference Request msg
Belief Request
Do Inference msg
Belief Request msg
Forgettable Concept msg
Active Concept msg
Temporal Concept Data Structure
Beliefs
Belief 1
Belief 2
Belief 3
...
Belief k
belief collection (all the events for the specified occurrence time to a max k items) (This can be curved bag)
[online diagramming & design] creately.com

Figure 3 - Temporal Concept

Concept Creation

Concepts are formed, by the Concept Creator, according to the following process:

1. Create a concept named for the statement content (and the occurrence time of the task), if it does not already exist.
2. Create concept for each term in the statements sub terms, in the statement meta data, if they do not exist.
3. Propagate the task to the primary concept and each sub term concept.

Local Inference

Local inference is carried out within a general concept and must not generate any derived tasks. It is an important consideration, given that OpenNARS 2.0.0 is a distributed system, to ensure that all local inference processing is local to the concept. Local inference can be further sub-divided into four sections: Goal task processing, Belief task processing, Question task processing and Quest task processing.

Goal Task Processing

When a goal arrives for processing, a check is carried out to see if a solution, that satisfies the goal, already exists. The check is carried out by first projecting all the containing concepts beliefs to the occurrence time of the goal task. The results are then ranked by confidence value and the belief with the highest confidence is selected. In the case when there are no satisfying beliefs the next step is bypassed. The budgets of the goal task and selected belief are adjusted: the goal task budget is decreased as it has been partially satisfied by the selected belief, whilst the belief budget is increased due to its usefulness in satisfying a goal. Now, all the goals in the concept are projected to the goal tasks occurrence time, then all goals, that do not have overlapping evidence with the goal task, are revised with it. The goal task and the revised task (if it exists, as it may not due to all goals having overlapping evidence) are then added to the concepts Tasklink collection. The revised goal task (or the goal task if there is no revised goal task) is then checked to see if it answers any outstanding quests (desire related questions) in the Tasks. The revised goal (or task goal, if no revised goal) is checked to see if it is an operation, if it's not then the process finishes, if it is, the revised goal task (or goal task, if no revised task) is projected to the current moment, if the projected tasks confidence exceeds a threshold (system parameter) an Operator Execution request is made, then the process finishes.

Belief Task Processing

When a belief task arrives for processing, a check is carried out to see if it satisfies an existing goal. The check is carried out by first projecting all the containing concepts goals to the occurrence time of the belief task. The results are then ranked by confidence value and the goal with the highest confidence is selected. In the case when there are no goals the next step is bypassed. The budgets of the selected goal and belief task are adjusted: the goal budget is decreased as it has been partially satisfied by the belief task, whilst the belief task budget is increased due to its usefulness in satisfying a goal.

A significant enhancement to local inference is the inclusion of anticipation handling within the belief task processing. The key to the new anticipation handling is the introduction of an anticipation task which is created for each anticipated event and subsequently revised with input events. Expired anticipation tasks generate a negative confirmation in relation to the truth value of the anticipation task.

At this point in the process, if the belief task is an input event task (not a derived task), the following occurs: all the anticipation tasks, in Tasks, are collated, and the belief task is revised with all the collated anticipation tasks.

Now, all the beliefs in the concept are projected to the belief tasks occurrence time, then all beliefs, that do not have overlapping evidence with the belief task, are revised with it. The belief task and the revised task (if it exists, as it may not due to all beliefs having overlapping evidence) are then added to the concepts Tasks collection.

The revised belief task (or the belief task if there is no revised belief task) is then checked to see if it answers any outstanding questions (truth related yes/no questions) in the Tasks.

Now, each anticipation task is checked to see if it has expired (expired = tcur > texp), if it has, a negative confirmation (based on $st(not(tv))$), of the anticipated task, is added to the Tasks, with a high priority. The anticipation task is removed from the task bag once it has expired. If the belief task is not confirmable (where confirmable means has an occurrence time in the future) and observable (the containing concept is marked as observable) then we finish the process, otherwise we create an anticipation task for the belief task and add it to the Tasks, then we finish the process.

Question task Processing

When a question (a yes/no question) arrives for processing, all the beliefs, in the tasks, are projected to the occurrence time of the question task. The results are then ranked by confidence value and the belief with the highest confidence is selected. The selected belief is added to the question task as the best solution (so far) and the budgets of the question task and selected belief are adjusted to reflect the partial solution. So the question budget is decreased as it has been partially answered and the selected belief budget is increased as it has been useful in answering a question. If the question task is a user requested question task, then the answer is output. The question task is then added to the Tasks and we finish the process.

Quest Task Processing

When a quest arrives for processing, all the goals, in the tasks collection, are projected to the occurrence time of the quest task. The results are then ranked by confidence value and the goal with the highest confidence is selected. The selected goal is added to the quest task as the best solution (so far) and the budgets of the quest task and selected goal are adjusted to reflect the partial solution. So the quest budget is decreased as it has been partially answered and the selected goal budget is increased as it has been useful in answering a quest. If the quest task is a user requested quest task, then the answer is output. The quest task is then added to the Tasks and we finish the process.

Revision

$$\{st(f_1, c_1), st(f_{2,}c_2)\}| - st\{F_{rev}\}$$

$$f = [f_1c_1(1 - c_2) + f_2c_2(1 - c_1)]/[c_1(1 - c_2) + c_2(1 - c_1)]$$

$$c = [c_1(1 - c_2) + c_2(1 - c_1)]/[c_1(1 - c_2) + c_2(1 - c_1) + (1 - c_1)(1 - c_2)]$$

Local Inference Flowchart

The details of each of these sub-processes can be seen in figure 4. See ‘Ranking, Projection and Eternalization’ section for details on ranking process.

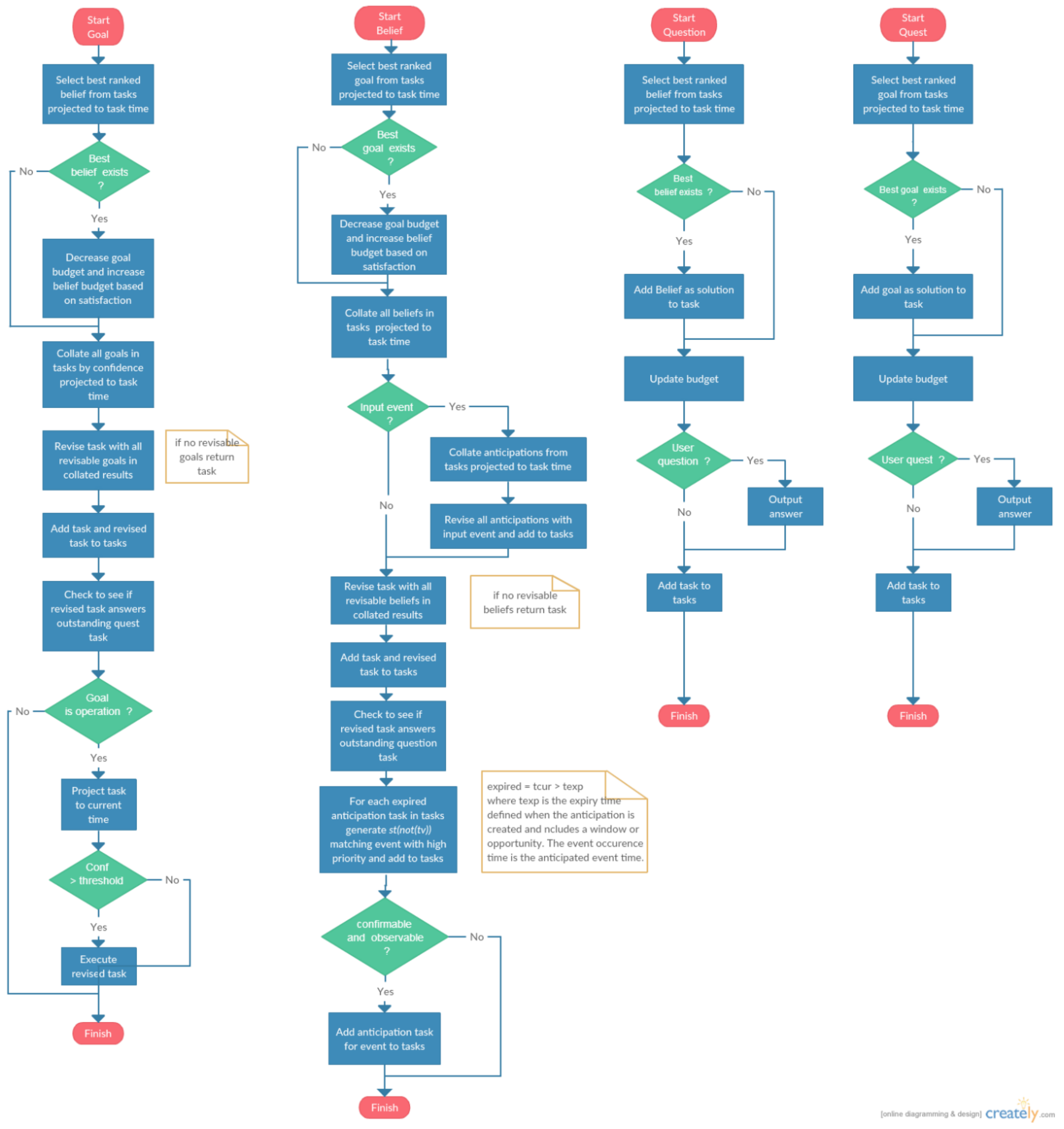
Start Goal
Select best ranked belief from tasks projected to task time
Best belief exists ?
No
Yes
Decrease goal budget and increase belief budget based on satisfaction
Collate all goals in tasks by confidence projected to task time
Revise task with all revisable goals in collated results
if no revisable goals return task
Add task and revised task to tasks
Check to see if revised task answers outstanding quest task
Goal is operation ?
No
Yes
Project task to current time
Conf > threshold
No
Yes
Execute revised task
Finish
Start Belief
Select best ranked goal from tasks projected to task time
Best goal exists ?
No
Yes
Decrease goal budget and increase belief budget based on satisfaction
Collate all beliefs in tasks projected to task time
Input event ?
Yes
Collate anticipations from tasks projected to task time
Revise all anticipations with input event and add to tasks
No
Revise task with all revisable beliefs in collated results
if no revisable beliefs return task
Add task and revised task to tasks
Check to see if revised task answers outstanding question task
For each expired anticipation task in tasks generate st(not(tv)) matching event with high priority and add to tasks
expired = tcur > texp where texp is the expiry time defined when the anticipation is created and ncludes a window or opportunity. The event occurence time is the anticipated event time.
confirmable and observable ?
No
Yes
Add anticipation task for event to tasks
Finish
Start Question
Select best ranked belief from tasks projected to task time
Best belief exists ?
No
Yes
Add Belief as solution to task
Update budget
User question ?
Yes
Output answer
No
Add task to tasks
Finish
Start Quest
Select best ranked goal from tasks projected to task time
Best goal exists ?
No
Yes
Add goal as solution to task
Update budget
User quest ?
Yes
Output answer
No
Add task to tasks
Finish
[online diagramming & design] creately.com

Figure 4 - Local Inference Flow Chart

General Inference

The general inference cycle

The system has two cycles; Perception/Action and general inference. Each of these cycles can run at a different rate. Here we will discuss the general inference cycle which typically runs at a slower rate than the Perception/Action cycle. The cycle rate is managed by an inference tick timer which is set in milliseconds (for example 15ms), by a system parameter.

The general Inference Cycle is as follows:

1. Select n concepts from general concept collection using selection algorithm
2. Send :inference-request-msg to each selected concept
3. Reset active concept collection
4. Each concept on receiving an :inference-request-msg does the following:
    a. Selects a task from the Task bag
    b. Gets all the priorities of the termlink target concepts, from the global priorities dictionary, and ranks the links by priority after applying the fuzzy activation level to each. Active links are added to the results collection.
    c. Selects n termlinks from the ranked termlinks collection, biased by priority
    d. Sends a :belief-request-msg to each target concept, along with the selected task
5. Each concept on receiving a :belief-request-msg, does the following:
    a. Updates its own link strength for the task concept based on the priority level in the :belief-request-msg
    b. Ranks and projects the beliefs in the Task bag to the task time
    c. Selects the best ranked belief
    d. Sends a :do-inference-msg, with the task and selected belief, to General Inferencer
6. General Inferencer sends derived results to Task Creator and subsequently to Task Dispatcher where the tasks are dispatched to the relevant concepts, and the cycle repeats.

Fuzzy activation is defined as: $f: Float \;--> \; Bool$

$$(x -> (x > ACTIVATION\_LEVEL + BINOMIAL\_NOISE\_BETWEEN\_-1\_AND\_1)$$

The temporal inference cycle is as follows:

1. Select n (where n is even) temporal concepts from temporal concept collection using selection algorithm
2. Send :temporal-inference-request-msg to every ‘even’ selected concept with ‘odd’ selected concept as msg parameter
3. Reset active concept collection
4. Each temporal concept on receiving a :temporal-inference-request-msg does the following:
    a. Selects a belief from the belief bag
    b. Sends a :belief-request-msg to the ‘odd’ concept (parameter) with selected belief
5. Each temporal concept on receiving a :belief-request-msg, does the following:
    a. Selects a belief from the belief bag
    b. Sends a :do-inference-msg, with the two selected beliefs, to General Inferencer
6. General Inferencer sends derived results to Task Creator and subsequently to Task Dispatcher where the tasks are dispatched to the relevant concepts, and the cycle repeats.

Processing selective questions and query variables

Unlike yes/no questions, which are handled in local processing, selective question, (questions that contain query variables) are processed within the general inference cycle. The process is the same as for general inference, as described above, with one difference. When a selective question is selected as a task, along with a belief, an additional function is executed, the choice rule for task-belief-pair.

The choice rule checks to see if the selected belief is a match for the task question, if it is and it is the best solution found so far, the question task is updated with the belief as a best solution, if the question task source is from a user then the answer is output. The budget of the task and belief is adjusted based on the quality of the solution, where the task budget is decreased (because the question has been partially answered) and the belief budget is increased (because the belief was useful in answering a question).

The choice rule is a separate function to the normal inference rules because unlike the inference rules it does not generate any conclusion and therefore does not fit within the constraints of the meta rule DSL. The choice rule is executed within the concept that the belief is selected from.

If a best solution is found then the belief budget is updated and a :belief-update-msg is sent to the source concept where the task budget is updated.

Projection, Ranking and Eternalization

When two semantically related statements are selected for inference, (if not stated otherwise by the specific rule), it is necessary to map the occurrence time of the belief or goal to the occurrence time of the task. This mapping function is called projection and describes how the truth value of a statement decreases in confidence when projected to another occurrence time. In this operation, the confidence of the belief or goal is decreased by a factor

$$k_c = \frac{|t_B - t_r|}{|t_B - t_C| + |t_r - t_C|}$$

where t_B is the original occurrence time of the belief or goal, t_r the occurrence time it is projected to, and t_C the current time. The new confidence of the belief is then

$$c_{new} = (1 - k_c) * c_{old}$$

Eternalization is a special form of induction, where the occurrence time is dropped, so the conclusion is about the general situation. The eternalized confidence value is obtained with

$$c_{new} = \frac{1}{k + c_{old}}$$

where k is a global evidential horizon personality parameter.

There is one exception to ranking by confidence. In the case of ranking selective questions, they are ranked according to e/C^r, where e is the expectation value of the statement, C its syntactic complexity and r a scaling factor to balance e and C.

In inference, whenever an event is derived, the eternalized version is also derived. However, the existence of eternal statements presents a problem: How to justify inference between two premises, about different times? In order to deal with this scenario, there are two possible routes: the inference is a temporal rule which measures the time between the premises, and takes into account when its conclusion is built, or, one of the following applies:

1. Premise1 is eternal, and premise2 is temporal. Here premise1 is eternalized before applying inference.
2. Premise1 is temporal, and premise2 is eternal. Since premise2 is eternal, it also holds at the occurrence time of premise1, so inference can occur directly.
3. Premise1 is temporal, and premise2 is temporal. In this case premise2 is projected to the occurrence time of premise1, and also eternalized. Inference now happens between premise1 and the stronger in confidence outcome, either the result of the projection or the result of eternalization.
4. Both are eternal, in which case the derivation can happen directly.

In all cases, the occurrence time of the first premise (usually the task), is assigned to the occurrence time of the derived task, and possibly a statement-dependent time-shift as specified by some temporal inference rules, dependent on the term encoded intervals, which measure time between events, is applied.

Task Meta Data

Tasks contain various meta data, that is inserted at various points of the respective cycles. Each task will have some, or all, of these meta data elements:

Meta Data Name	Definition	Description
:truth	(f, c)	Floating point tuple representing the truth value: frequency and confidence, of the statement
:desire	(p, d)	Floating point tuple representing the desire value: plausibility and desirability, of the statement
:budget	(p, q, d)	Floating point tuple representing the budget value: priority; quality; and durability of the statement
:creation	Int64	Cycle time when the task was created. Used by task dispatcher as the current system time
:occurrence	Int64	Cycle time when the task is expected to occur
:source	:user :derived :percept :peer :parent :child	An enumeration of possible statements sources, where :peer, :parent and :child are other NARS systems and will additionally contain a GUID to uniquely identify the specific NARS system. For example: :peer {25892e17-80f6-415f-9c65-7395632f0223}
:id	Int64	A unique id for each statement starting at 0
:evidence	List of int64	A list of serial-no's that represent the individual statements that have been used in the derivation chain for each statement. The list is capped at max-derivations (a system parameter). New evidence is added to the front of the list. When two premises are used in inference their respective evidence lists are interleaved and then capped at max-derivations. This is then used as the evidence for the derived premise.
:sc	int	The number of terms and operators involved in a statement, syntactic complexity.
:terms	List of terms	The list of sub terms included in the statement. The terms are limited to n levels of recursive depth (currently 3, but can be changed by a system parameter). This is to match the term depth limit in the existing inference rules. This list is an optimisation to increase the performance of Task Dispatcher and should only be as deep as required by the statement content (up to the limit imposed by the depth parameter).
:solution	map	Questions use this field to maintain a record of the best solution found so far
:task-type	key	:belief, :goal, :question or :quest
:content	map	A value that represents the content of the task, i.e. the term.

Attention Updating

The attention mechanism is controlled by: emotional control parameters and attention values. This section explains, where and how, attention is updated.

Attention action	Update method
Statement Creation	Default parameters or user defined
Derived statement	Derived Task Budget
Concept creation	Concept Activation
Concept Forgetting	Concept Forgetting
Add task to concept	Concept Activation
Remove task from concept	Currently, no action required
Remove belief from concept	Currently, no action required
Statements used for inference	Task Forgetting
Belief satisfies goal	Reduce goal budget Increase belief budget
Belief satisfies question	Reduce question budget Increase belief budget
Belief satisfies quest	Reduce quest budget Increase belief budget
Goal satisfied by belief	Reduce goal budget Increase belief budget
Question satisfied by belief	Reduce question budget Increase belief budget
Quest satisfied by goal	Reduce quest budget Increase goal budget
Selective question satisfied by belief	Reduce question budget Increase belief budget

The following table defines the Attention function for each attention action:

Update method	Attention function
Derived Task Budget	$p(T) = or(and(p(parent_T), expectation(s(link)), tr(oc_T)), k * emotion\_boost)$ $d(T) = or(d(parent_T) * 1/C, k * satisfaction\_reward)$
Concept Activation	$or(p(T_1), \ldots, p(T_n))$
Concept Forgetting	$diff = p(concept_{new}) - p(concept_{prev})$ $quality' = \max(0, \min(1, quality + k * diff))$ where $k = k_1\ (if\ diff > 0), k_2(if\ diff < 0), 0\ (else)$
Task Forgetting	$p(T)' = and(p(T), d(T))$
Reduce Task budget	$satisfaction = 1 - \lvert expectation(Desire) - expectation(Truth) \rvert$ $p(T)' = and(p(T), (1 - satisfaction))$ $p(T)' = 1 - confidence^1(solution)$ (for questions/quests only)
Increase Task budget	$d(T)' = or(d(T), k * satisfaction)$
Termlink Strength	$w^+ = and(c1, c2)$ $w^- = xor(c1, c2)$ $w = w^+ + w^-$ $s(f,c) = (w^+/w, w/(w+k)$ where k is specific link strength horizon $s(f,c) = revision(s(old), s(new))$

The naming convention is as follows: $a(C) = activation\ level\ of\ concept \in [0,1]$, $p(T) = priority\ of\ task \in (0,1]$, $d(T) = durability\ of\ task \in (0,1)$ and $s(link) = strength\ of\ termlink \in [0,1]$. k is a scaling factor and is a system parameter, it may need to be different for each function. C is the syntactic complexity of the task. $tr(oc_T) = \frac{1}{1+k*|oc_T - now|}$ where $tr(oc_T)$ is the temporal relevance priority factor and oc_T is the occurrence time of the derived task and now is the current system time, in system cycles. Confidence[1] = desirability for quests. Link strength corresponds to the relation: (c1 * c2) --> associated

Termlinks

Termlinks can be conceptually considered as: <(*, a, b) --> associated> %momentum% and the link strength is updated via the revision rule (with a specific Horizon parameter).

The meaning of a concept, at a point in time, is fluid and evolving. Although, in the present design, all termlinks are selected, for 'potential' inference, they are not necessarily all used. Only the termlink targets, that are active, are used for inference. In this way, only contextually relevant related concepts are used for inference. Furthermore, the link strength is applied as an additional bias in the budget calculation of any derived tasks, thereby ensuring that experienced contextual relevance is also incorporated in the budget calculation.

Forgetting

Concept forgetting is carried out by the Forgettable Concept Collator in conjunction with the Concept Creator, General Concepts and Temporal Concepts. When a concepts quality level falls below a pre-defined threshold (dynamically managed) it sends a :forgettable-concept-msg to the Forgettable Concept Collator. Here the concepts (references) are added to a reverse priority queue and ordered by $average(quality(concept), p(concept))$. When additional memory is required, determined by the existence of : concept-limit-msg (from concept creator), the queue is popped, and the concept is removed from memory (the global concept map is updated to reflect the change).

It is possible that the concept queue could be empty when additional memory is required, in this case the quality level threshold is increased to provide a greater flow of forgettable concepts. In the reverse case, of too many forgettable concepts, the threshold is reduced.

A count of : concept-limit-msg is maintained to ensure that the requested number of concepts is always freed, even if it requires an adjustment to the threshold to provide sufficient concepts for removal.

It is possible that the forgetting threshold could be adjusted above a concepts current activation level. If the concept receives no further messages this concept would effectively be invisible to the forgetting system. To resolve this issue, a garbage collection like sweep process is run periodically. Task Dispatcher records the time, of the last task dispatched to each concept, in the global concept table. The background sweep process periodically checks the global concept table for tasks that have not received any tasks for a pre-defined period. These tasks are then sent messages to check their forgettable state.

Dynamic Thresholds

There are two dynamic thresholds that the system manages: Activation Threshold and Forgetting Threshold. These are implemented as global atoms and updated in response to system throughput and memory pressure. Activation Threshold is managed by the Active Concept Collator and adapts to the volume of active concepts passed to the collator. Similarly, the Forgetting Threshold is managed by the Forgettable Concept Collator and adapts to the volume of concepts available for removal.

Actor Messages

Perception/Action Actor Messages

Actor Name: **Sentence Parser**

Inbound messages:

Message	Parameters	Description
:narsese-msg	String	

Outbound messages:

Message	Parameters	Description
:sentence-msg	sentence	

Actor Name: **Task Creator**

Inbound messages:

Message	Parameters	Description
:sentence-msg	Sentence	

Outbound messages:

Message	Parameters	Description
:task-msg	Task	

Actor Name: **Operator Executor**

Inbound messages:

Message	Parameters	Description
:operator-execution-msg	Operator ID, param list	

Outbound messages:

Message	Parameters	Description
:sentence-msg	Sentence	

Memory Management System Actor Messages

Actor Name: Task Dispatcher

Inbound messages:

Message	Parameters	Description
:task-msg	Task	

Outbound messages:

Message	Parameters	Description
:create-concept-msg	Task	
:task-msg	Task, system time	

Actor Name: Concept Creator

Inbound messages:

Message	Parameters	Description
:create-concept-msg	Task	

Outbound messages:

Message	Parameters	Description
:task-msg	Task	
: concept-limit-msg	n/a	

Actor Name: General Concept

Inbound messages:

Message	Parameters	Description
:task-msg	Task, system time	
:belief-request-msg	Task, p(concept)	
:inference-request-msg	n/a	
:task-budget-update-msg	Task id, satisfaction	

Outbound messages:looks good

Message	Parameters	Description
:active-concept-msg	Actor reference for self	
:belief-request-msg	Task	
:forgettable-concept-msg	Actor reference for self	
:do-inference-msg	Task, Belief	
:operator-execution-msg	Operator id, param list	
:task-budget-update-msg	Task id, satisfaction	

Actor Name: Temporal Concept

Inbound messages:

Message	Parameters	Description
:task-msg	Task, system time	
:belief-request-msg	Belief	
:temporal-inference-request-msg	Temporal Concept ref	

Outbound messages:

Message	Parameters	Description
:active-concept-msg	Actor reference for self	
:belief-request-msg	Task	
:forgettable-concept-msg	Actor reference for self	
:do-inference-msg	Task, Belief	

Actor Name: Forgettable Concept Collator

Inbound messages:

Message	Parameters	Description
: concept-limit-msg	n/a	
:forgettable-concept-msg	Concept id and activation	

Outbound messages:

Message	Parameters	Description
n/a	n/a	

Actor Name: Persistence Manager

Inbound messages:

Message	Parameters	Description
:concept-state-msg	n/a	

Outbound messages:

Message	Parameters	Description
: concept-state-request-msg	n/a	

Message Display Actors

Actor Name: Task Formatter

Inbound messages:

Message	Parameters	Description
:task-msg	task	

Outbound messages:

Message	Parameters	Description
:display-msg	Display string	

Actor Name: Console Display

Inbound messages:

Message	Parameters	Description
:display-msg	Display string	

Outbound messages:

Message	Parameters	Description
n/a	n/a	

General Inference Cycle Actor Messages

Actor Name: **General Inferencer**

Inbound messages:

Message	Parameters	Description
:do-inference-msg	Task, belief	

Outbound messages:

Message	Parameters	Description
:derived-sentence-msg	sentence	

Actor Name: **Active Concept Collator**

Inbound messages:

Message	Parameters	Description
:inference-tick-msg	n/a	
:active-concept-msg	Actor reference for concept	

Outbound messages:

Message	Parameters	Description
:inference-request-msg	n/a	

References

The following references, and associated links, are useful resources for detailed explanations of many of the requirements included in this design document:

1. Hammer P., Lofthouse T., Wang P., The OpenNARS implementation of the Non-Axiomatic Reasoning System, AGI-2016 Conference proceedings (in review) link
2. Lofthouse T., Hammer P., Generalized Temporal Induction with Temporal Concepts in a Non-Axiomatic Reasoning System, AGI-2016 Conference Proceedings (in review) link
3. Wang P., Talanov M., Hammer P., The Emotional Mechanisms in NAR, AGI-2016 Conference Proceedings (in review) link