The Conceptual Design of OpenNARS 3.1.2

Temple AGI Team

February 14, 2022

The document describes the parts of the conceptual design of OpenNARS 3.1.2 that have been updated from [Wang, 2013, Wang et al., 2020]. The structure mainly corresponds to the OpenNARS Coding Workshop held in Summer 2021.

1 Overall Design

1.1 Supported usages

The following ways to run OpenNARS have been considered:

1. As an application used by a user via a GUI: this is the most common usage of the software.
2. As an application used via command-line: this is when the software is integrated into a larger system.
3. As an app runs in a smartphone: this applies to smartphones running JVM
4. As a controller of a robot: this has been tested in a previous version
5. As a library of reasoning service: OpenNARS can be used as a whole in the above ways, but it requires some integration to be used as a collection of tools.

The above 1-5 will basically use the same code-base, and the custom code needed for an integration will all be added to the *main* package only.

1.2 Architecture and working cycle

There is a single Java class for reasoner (*NAR*) that directly outlines to the overall architecture. The class will be in the *main* package and contains the following major data fields:

- a single Memory
- two Buffers (overall and internal experience buffers)
- a list of Channels
- a group of status and metrics evaluations
- a system clock

When the system is running, in each clock tick a working cycle is finished, which goes through the above fields in roughly a constant time (range of ms). Though it is possible to use multiple threads for increased performance, a single-threaded synchronized approach intentionally is used to allow repeatability of the process. It is also possible to distribute processing time since the amount of time in each field can be adjusted within the corresponding class.

1.3 Package structure

The package structure of the source code was reorganized to fit the architecture logic, as well as to match the components of the system.

A target package structure is:

main: Architecture and usages (Shell, NAR, NarsBatch, Parameter, ...)

gui: Graphical User Interface (main, NAR_GUI, Window Applet Classes, ...)

storage: Data structures (Bag, Buffer, Memory, ...)

entity: Data items (Concept, Task, Sentence, TruthValue, BudgetValue ...)

inference: Inference rules and functions (SyllogisticRules, BudgetFunctions ...)

language: Narsese language terms and operators

io: Input/Output operations for channels

mental_operator: High level inference mental operators

tests: Collection of tests that verify and test functionality of the system

Additional testing packages can be used to separate the experimental code with the major components of the system.

1.4 Software engineering

The development of the software follows the common practice:

- In the code, *javadoc* comments are present for major parts, however more comments need to be added and are being added per development progress
- Java unit tests to be added to capture common issues.
- New high-level evaluation metrics is always encouraged to allow complete system evaluation
- Wiki documents should be updated per development process.
- Docker will not be used in V3.1.2.

2 Data Structures

Issues and potential changes in the data structures.

2.1 Buffer

Since data items are not repeatedly accessed but only processed once at a time, priority queue was used in Buffer instead of Bag data structure. Elements in buffer have a time limit which is called duration in the code. There are two buffers in the current architecture: Internal Experience and Overall Experience Buffers, both are implemented using corresponding classes within the storage package and both inherent from *EventBuffer* class which has the code responsible for temporal compositions for the participating events which are necessary in both buffers. The elements of the EventBuffer are also in the Buffer, except the elements that have been selected out of the Buffer are still in the EventBuffers until expiring. EventBuffer is implemented using Java ArrayList which is sorted by Occurence time.

Temporal composition in Buffer will be limited to *sequential conjunction* (,) and *predictive implication* ($/\!\!\Rightarrow$) at the current version, though may be extended to include *parallel conjunction* (;) and *concurrent implication* ($|\!\!\Rightarrow$) in a future version. Temporal *equivalence* is not included, because negative evidence cannot be collected in parallel with positive evidence. The same is for *retrospective implication*.

The premises of the two compounds are selected for different purposes:

- $(A/\!\!\Rightarrow B)$ is formed to predict B using A, and it will get a high priority when B is not anticipated, as measured by its achieving-level. A should also be an unusual event with high priority for it to be considered as a predictor of B.

- (A, B) is formed to simplify the description of the experience. Instead of completely depending on composition, the system may depend more on decomposition of the content of the buffer. For example, sequence of events A, B, C, D, E may be treated as a single event if compound (A, B, C, D, E) already exist in memory, or $((A, B, C), (D, E))$ if both sub-compounds are in memory. It is also possible for (A, C, D) to be taken as a compound. An efficient procedure is needed to do the decomposition without exhausting all combinations.

Since every task initially goes through the Overall Experience Buffer once, then is repeatedly processed in the Memory, it may be more natural to do its pre-reprocessing (using local rules and constant time) when it enters the Overall Buffer, because its achieving level and priority can be decided more accurately. We will revisit this idea in the future version.

Besides inference results, the Internal Experience Buffer should collect the execution of the following mental operations:

- Receiving a task from Overall Buffer

- Selecting a concept in Memory
- Selecting a task in concept
- Selecting a belief in concept
- Feeling the value of a status measurement
- Executing an operation

2.2 Memory

Memory retains the same bag data structure and architecture of a hyper-graph, where nodes are concepts connected with multiple types of links: *termlinks* and *tasklinks*. Newer public methods are available to query truth-value, desire-value, concept priority, etc., that directly returns a value without any inference, however such querying, evaluation and influence on overall system's performance will be available in the next versions.

In addition, all the GUI methods are moved to GUI package such that there will be no confusion between the reasoner's and I/O code.

2.3 Concept

For this version concept class remains unchanged, only the GUI methods have been moved to GUI package to simplify the overall codebase. A possible change for future versions is to turn the termLink types into separate termLink tables. This approach was considered for early versions of OpenNARS (such as to have separate link bags for "extension", "intension", "similarity", ...), but the idea was dropped to prefer a pure data-driven (and type-blind) premise selection. Turning link types into tables may improve the efficiency of certain inference by adding a type-bias to the selections. It is still possible to keep the data-driven style by allowing multiple beliefs of different types to be used to process a task in a working cycle.

Whether it is possible to replace or extend the compositional term-links with semantics term-links is an issue to be discussed. In the initial versions of OpenNARS, there were semantic links, so $A \rightarrow B$ was represented by links from A and B to the other. Later, these links were replaced by termLinks between A (or B) and $A \rightarrow B$, where the last one also contains truth-values. In this way, A and B are no longer directly point to each other, but via $A \rightarrow B$.

The recently introduced precondition-list may be handled in this way uniformly.

A related idea is to merge the single-component compound $(\neg\, T)$, $\{T\}$, and $[T]$ into the component concept T, and keep their links in separate tables. The semantic justification is that the meaning of the compound and the component mutually decide each other, which is not the case for the multi-component compounds. This treatment is especially necessary for negation.

The belief/desire tables in a concept keep multiple versions of the value, initially to resolve the problem of overlapping evidence between a task and a

belief. With multiple beliefs, others can be check if the first choice cannot be used. These tables are sorted arrays, not priority queues or Bags. Another possible usage of multiple truth-values is to handle metaphor, analogy, and context-sensitive inference where some previous experience is deliberately ignored to keep context sensitivity, so the most recent truth-value is used even when it does not have the highest confidence.

2.4 Bag

The Bag data structure remains unchanged, one small addition now include forgetting rate that is related to average priority of an elements in Bag. Such forgetting rate might help achieve a desired priority distribution of elements within the Bag.

3 Resource Allocation

The following changes were included:

- Defined *achieving level* in Task (Task.java), and use $h = 1 - |e - e'|$ for Judgment and goal and for questions e' is truth expectation of a best solution.
- Defined *simplicity* in Term (Term.java) as $s = m^{-r}$ where m is the number of atomic components within the term, and r a system parameter defaulted to 0.5.
- Defined *directness* in Stamp (Stamp.java) as $d = b^{-t}$ where b is the length of the evidential base and, and t a system parameter.
- The *fondness* mentioned in [Wang et al., 2020] is not currently implemented, as its usage is unclear.
- Sharpness in *getSharpness()* in TruthValue.java is now defined as $2 * |e - 0.5|$, as the [4] in [Wang et al., 2020] footnote.
- Added *projectedConfidence()* in Sentence.java that returns $c \times (t - t' + 1)^d$, where c is the confidence, t the current time, t' the creationTime of the truth-value, and d a system parameter indicating the decay rate defaulted to 1000.
- In Concept, there is a possibility to rank the beliefs and desires by their projected confidence, so as to favor more recent conclusions.
- *summary()* was renamed to *totalBudget()* in BudgetValue.
- Currently there is no priority defined among the buffers and channels that used to allocate working cycles among them. Instead each of them to be run in every cycle, though how many tasks are handled in each cycle can change.
- In class Reasoner (NAR.java), define the four "global evaluations": *satisfaction, alertness, busyness and well-being*, as specified in [Wang et al., 2020].
- Dividing term *complexity* (positive integer) in *budgetInference()* in (BudgetFunctions.java) is replaced by multiplying by *simplicity* (real number in (0, 1])

4 Narsese

Updates to the Narsese grammar in [Wang, 2013]:

- Because the dependence list of dependent variables seem useless, dependent variables and independent variables are distinguished only by their prefix, '#' and '$', respectively.

- There is a confusion between "quest" and "query" in the documents. The former should be a type of question that asks for the desire-value of an event, while the latter only used in "query variable" that is a variable term in questions (including quests) to be instantiated in the answer.

Tentative changes in the Narsese related code:

- The content of the class *Stamp* may need to be refactored, to separate the attributes of *TruthValue* (*evidentialBase* and *occurrenceTime*) and the others.

There are other issues to be discussed, though they do not demand changes in programs or documents:

- In the future versions, compound-term will probably be extended to include array (for vision) and list (for language). This will not happen in V3.1.2, though should be kept in mind during the process so to be compatible with the changes.

- The current grammar is less restrictive to the actual language supported by NARS, in the sense that some sentences can be accepted, but not processed, such as those that contains multiple statement-level copulas, or those where variable terms appear in non-canonical locations. There are may still be some derived compound term that have this nature, though should be detected testing, so as to modify the rule that produce them.

- The "tense" of a question can be confusing, as a question of a "future" tense may get an answer with a "past" tense, as the exact moment referred to has changed.

- "Negative sentences" usually should be expressed using explicit negation sign, rather than a low frequency value, because some inference rules (like deduction) cannot handle a premise of low-frequency, and a goal with low-frequency will not effectively derive new goals. However, low-frequency judgments and goals work as expected in (truth-value or desire-value) revision, as an opposition of a (positively stated) judgment or goal, rather than as a stand-alone judgment or goal itself.

- It may need to be enforced that executable operations only use sets as arguments, because sensorimotor only directly works on the boundary of the conceptual hierarchy.

- The two virtual terms, E (for "current evidence" that is used to transfer the truth-value between a statement T and an implication $E \Rightarrow T$) and D (for "current desire" that is used to transfer between the desire-value of a statement T and the truth-value of an implication $T \Rightarrow D$) are both *events* whose content depend on time.

- In temporal inference, the related term connectors (e.g., *sequential conjunction* ',') and copulas (e.g., *predictive implication* '$/\!\!\Rightarrow$') are interpreted as a *conjunction* of a temporal relation that is binary and a semantic relation that is multi-valued. For an inference step, the semantic relation decides the truth-value calculation (as in the non-temporal situation), while the temporal relation decide the occurrence order in the conclusion (if possible). The weak inference rules are no longer the inverse of the strong rules anymore, as the temporal order may be violated in that way. For example, $\{A /\!\!\Rightarrow B, A /\!\!\Rightarrow C\} \vdash B /\!\!\Rightarrow C$ is not a valid inference rule, while $\{A \backslash\!\!\Rightarrow B, A /\!\!\Rightarrow C\} \vdash B /\!\!\Rightarrow C$ is.

5 NAL

The inference rules of OpenNARS 3.1.2 are exactly the same as the previous versions, following Non-Axiomatic Logic (NAL) as specified in [Wang, 2013]. The planned changes are discussed in the following.

5.1 Temporal inference

In 3.1.2, temporal inference will happen in two places:

1. In each EventBuffer, limited forms of temporal compositions are carried out among events observed within a certain period, as described in Section 2.1.

2. In the inference engine, whenever an Implication, Equivalence, or Conjunction is formed and all the components are events, a temporal version of the connector will be used, and the conclusion will be temporal.

In both cases, the time interval is the difference of the occurrence time of the two events, represented as an attribute of the compound. As the interval is not part of compound term, revision may happen between two statements with different time intervals, where the interval of the belief (or second event) will be projected as before, with a confidence discount.

These two types do not include direct temporal compositions between two events that are far from each other in time and without common term. Whether such cases should be considered is left for future study.

The above inference rules should have been implemented already in version 3.1.1 and 3.0.4, though testings are needed to make sure they are done properly and completely.

5.2 Variable-related rules

Issues to be resolved:

- Whether there are more variable introduction rules beside the four in Table 10.6 in [Wang, 2013],

- Whether to support more than two occurrences of a variable in a sentence.

5.3 Mental operator

As a step toward self-awareness and self-control, in 3.1.1 major events in the inference cycle will be "operationalized", meaning an existing method will be associated with a mental operation that can be invoked by the system, and the execution of an operation will enter the internal experience buffer, together with the derived tasks.

The candidate events include:

- The insertion of a new task inserts into memory (different operations for different task type),
- The selection of a concept,
- The selection of a task,
- The Implication formed between the selected task and a derived task,
- A feeling reporting one of the four global measurements.

The list may change.

Two ways to realize this operationalization:

- Rewrite the related code, so the same code can be invoked both as an operation and as a method,
- Keep the existing code, and use additional code to turn an operation execution to a method call, as well as the other way around.

While each I/O operation is registered in the corresponding channel, the mental operators are registered in the Memory class or the Reasoner class.

Special mental operations that needs discussions:

- *compile*: to give an atomic name to a useful compound term by creating a special Similarity statement between the two. Special case: compound operations with arguments.
- *assume*: to create a local context with implicit assumptions, by using special region or higher-order statements.

5.4 Analytic truth

As explained in Section 9.5 of [Wang, 2013], analytical truths are only used as premises in strong inference. This policy may be too restrictive for the following case:

```
  <swan --> (|,bird,swimmer)>.
  <swan --> swimmer>?
  20
```

In 3.0.4, the above question gets no answer, but the following input

```
  <swan --> bird>. %0.90%
  <swan --> (|,bird,swimmer)>?
20
  <swan --> swimmer>?
...
  <swan --> swimmer>. %0.09;0.06%
```

It does not look right, though the conclusion has very low confidence.

The issue is not found in 3.1.1, though it does not mean that similar issues do not exist there.

References

[Wang, 2013] Wang, P. (2013). *Non-Axiomatic Logic: A Model of Intelligent Reasoning*. World Scientific, Singapore.

[Wang et al., 2020] Wang, P., Hammer, P., Isaev, P., and Li, X. (2020). The conceptual design of opennars 3.1.0. Technical Report 11, AGI Team, Temple University, Philadelphia, USA.