CLAIMS

The following is claimed:

1. A ~~distributed system generating~~ rule compiler apparatus for generating a distributed system, comprising:

a memory;

a processor in communication with the memory, and configured to issue a plurality of processing instructions stored in the memory, wherein the processor issues instructions to:

(a) obtain, via the processor, a distributed system execution request data structure;

(b) determine, via the processor, an output rule structured to comprise a declaration implementing polymorphic parameterization and ~~the output rule's~~a set of rule parameters associated with the output rule specified via the distributed system execution request data structure for a requested calculation, wherein a current rule is initialized to the output rule;

(c) query, via the processor, a rule lookup table data structure to determine a set of matching rules, corresponding to the current rule, based on ~~the current rule's~~an identification tuple associated with the current rule;

(d) select, via the processor, ~~the~~ a best matching rule from the set of matching rules, wherein ~~the best matching rule's~~a set of rule parameters associated

with the best matching rule best corresponds to ~~the current rule's~~a set of rule parameters associated with the current rule;

(e) generate, via the processor, a logical dependency graph data structure by adding logical dependency graph nodes and logical dependency graph edges corresponding to the best matching rule, precedent rules of the best matching rule, and precedent rules of each of the precedent rules of the best matching rule;

(f) determine, via the processor, an execution complexity gauge value associated with the generated logical dependency graph data structure;

(g) determine, via the processor, a set of distributed worker processes to be utilized to execute the requested calculation;

(h) divide, via the processor, the generated logical dependency graph data structure into a set of subgraphs based on the execution complexity gauge value, wherein each worker process in the set of distributed worker processes is assigned a subgraph from the set of subgraphs;

(i) initialize, via the processor, each respective worker process in the set of distributed worker processes with the subgraph assigned to the respective worker process;

(j) coordinate, via the processor, execution of the requested calculation by worker processes in the set of distributed worker processes, wherein each respective worker process calculates results for logical dependency graph nodes in the subgraph assigned to the respective worker process; and

(k) obtain, via the processor, a computation result of ~~the~~ a logical dependency graph node from the logical dependency graph nodes corresponding to the output rule, from the worker process in the set of distributed worker processes that is assigned the subgraph that contains the logical dependency graph node corresponding to the output rule.

2. The apparatus of claim 1, wherein the rule lookup table data structure contains a set of identification tuple to rule data structure mappings.

3. The apparatus of claim 2, wherein a rule data structure includes a data field that references an associated concrete atom data structure that implements a forward calculation operation of a generic atom interface to calculate a result associated with the respective rule data structure.

4. The apparatus of claim 3, wherein a rule data structure includes a data field that references a factory function that creates an instance of the associated concrete atom data structure.

5. The apparatus of claim 1, wherein an identification tuple comprises an ordered list that includes a rule's namespace and rule name.

6. The apparatus of claim 1, wherein an identification tuple comprises a hash value calculated based on an ordered list that includes a rule's namespace and rule name.

7. The apparatus of claim 1, wherein a single rule exists in the set of matching rules, and the single rule is selected as the best matching rule.

8. The apparatus of claim 1, wherein multiple rules exist in the set of matching rules, and a rule with ~~the~~ a minimum inheritance distance is selected as the best matching rule.

9. The apparatus of claim 8, wherein multiple rules with the minimum inheritance distance exist, and the best matching rule is selected using a tie breaker technique based on one of: position of rule parameters, types of rule parameters.

10. The apparatus of claim 1, wherein the instructions to (e) generate the logical dependency graph data structure further comprise instructions to:

(e1) determine, via the processor, a logical dependency graph node identifier for the best matching rule;

(e2) either, upon determining, via the processor, that the logical dependency graph node identifier does not yet exist in the logical dependency graph data structure:

add, via the processor, a new logical dependency graph node, corresponding to the best matching rule, to the logical dependency graph data structure, wherein the new logical dependency graph node is assigned the logical dependency graph node identifier;

upon determining, via the processor, that the current rule is not the output rule, add a logical dependency graph edge between the new logical dependency graph node and a logical dependency graph node corresponding to ~~the best matching rule's~~ a dependent rule for the best matching rule;

determine, via the processor, ~~a~~ the set of precedent rules for the best matching rule, wherein the best matching rule is a dependent rule for precedent rules in the set of precedent rules; and

process, via the processor, each respective precedent rule in the set of precedent rules by: setting the current rule to the respective precedent rule, and repeating elements (c)-(e); and

return, via the processor, to processing precedent rules of the ~~best matching rule's~~ dependent rule for the best matching rule until precedent rules for the output rule are processed;

(e3) or, upon determining, via the processor, that the logical dependency graph node identifier already exists in the logical dependency graph data structure:

add, via the processor, a logical dependency graph edge between an existing logical dependency graph node, associated with the logical dependency graph node identifier, and a logical dependency graph node corresponding to the ~~best matching rule's~~ dependent rule for the best matching rule; and

return, via the processor, to processing precedent rules of the ~~best matching rule's~~ dependent rule for the best matching rule.

11. The apparatus of claim 1, wherein a logical dependency graph node is a data structure that includes a data field that that references a corresponding rule's identification tuple.

12. The apparatus of claim 1, wherein a logical dependency graph edge is a data field in a logical dependency graph node that refers to another logical dependency graph node.

13. The apparatus of claim 1, wherein the execution complexity gauge value is determined based on at least one of: ~~the~~ number of nodes in the logical dependency graph, ~~the~~ sum of output tensor sizes of logical dependency graph nodes, network data transfer size associated with the logical dependency graph, computation time of the requested calculation.

14. The apparatus of claim 1, wherein ~~the~~ cardinality of the set of distributed worker processes is determined based on a specified fixed number of available worker processes.

15. The apparatus of claim 1, wherein ~~the~~ cardinality of the set of distributed worker processes is determined based on the execution complexity gauge value and a scaling parameter value, wherein the scaling parameter value is determined based on at least one of: target processing time, target available memory, target cost, target data transfer size.

16. The apparatus of claim 15, wherein the cardinality of the set of distributed worker processes is further determined based on an average worker execution capability.

17. The apparatus of claim 1, wherein ~~the~~ number of nodes in a subgraph assigned to each respective worker process is determined based on ~~the respective worker's~~

worker execution capability associated with the respective worker process, wherein the worker execution capability is determined based on at least one of: processor speed, memory size, network speed.

18. The apparatus of claim 1, wherein at least some of logical dependency graph nodes in the generated logical dependency graph data structure are duplicated in multiple subgraphs that are assigned to different worker processes.

19. The apparatus of claim 1, wherein the generated logical dependency graph data structure includes a packaged logical dependency graph node that utilizes an internal logical dependency graph data structure, wherein the internal logical dependency graph data structure is constructed using an associated packaged atom data structure.

20. The apparatus of claim 19, wherein the internal logical dependency graph data structure is in ~~the~~ same memory space as the worker process in the set of distributed worker processes that is assigned the subgraph that contains the packaged logical dependency graph node.

21. The apparatus of claim 19, wherein the internal logical dependency graph data structure is distributed across multiple worker processes in the set of distributed worker processes.

22. The apparatus of claim 1, further comprising:

the processor issues instructions to:

provide a graph representation of the logical dependency graph data structure;

obtain a user selection of a target node in the graph representation;

determine a target logical dependency graph node in the logical dependency graph data structure corresponding to the user selected target node in the graph representation;

obtain a computation result of the target logical dependency graph node from the worker process in the set of distributed worker processes that is assigned the subgraph that contains the target logical dependency graph node; and

provide the computation result for display of the target logical dependency graph node.

23. The apparatus of claim 22, wherein the target node in the graph representation includes a visual indicator that specifies the worker process in the set of distributed worker processes that is assigned the subgraph that contains the target logical dependency graph node.

24. The apparatus of claim 23, wherein the visual indicator is at least one of: color, color pattern, shape, or border style of the target node.

25. The apparatus of claim 22, wherein each respective node in the graph representation includes a visual indicator that specifies whether ~~the respective node's~~ precedent nodes associated with the respective node are displayed.

26. The apparatus of claim 25, wherein the visual indicator is at least one of: color, color pattern, shape, or border style of the respective node.

27. The apparatus of claim 22, further comprising:

the processor issues instructions to:

obtain a user command to view state replication code, wherein execution of the state replication code on a remote client replicates state of an associated logical dependency graph node on the remote client;

determine state replication code associated with the target logical dependency graph node; and

generate a display containing the determined state replication code.

28. The apparatus of claim 22, further comprising:

the processor issues instructions to:

obtain a user command to switch graph display mode;

provide a second graph representation of a rule dependency graph data structure corresponding to the logical dependency graph data structure;

obtain a user selection of a second target node in the second graph representation; and

provide display information associated with the second target node, wherein the display information includes a list of logical dependency graph nodes

associated with the second target node and metadata associated with the second target node.

29. The apparatus of claim 1, further comprising:

the processor issues instructions to:

provide a graph representation of the logical dependency graph data structure;

obtain a user selection of a target node in the graph representation;

determine a target logical dependency graph node in the logical dependency graph data structure corresponding to the user selected target node in the graph representation; and

provide a second graph representation of ~~the target logical dependency graph node's~~ internal logical dependency graph data structure associated with the target logical dependency graph node upon determining that the target logical dependency graph node is a packaged atom.

30. The apparatus of claim 29, further comprising:

the processor issues instructions to:

obtain a user command to switch graph display mode; and

provide a third graph representation of a rule dependency graph data structure corresponding to the target logical dependency graph node's internal logical dependency graph data structure.

31. The apparatus of claim 30, wherein the third graph representation further includes a graph representation of a rule dependency graph data structure corresponding to the logical dependency graph data structure.

32. The apparatus of claim 30, wherein nested graph representations of rule dependency graph data structures include navigation links that facilitate navigation among the nested graph representations.

33. The apparatus of claim 1, wherein at least one of: the best matching rule, a precedent rule of the best matching rule, and a precedent rule of another precedent rule utilizes dynamic precedents.

34. The apparatus of claim 1, wherein at least one of: the best matching rule, a precedent rule of the best matching rule, and a precedent rule of another precedent rule corresponds to a rule entry in the rule lookup table data structure for a cloned namespace.

35. The apparatus of claim 1, wherein at least one of: the best matching rule, a precedent rule of the best matching rule, and a precedent rule of another precedent rule is an overriding rule in a cloned namespace determined via a namespace lookup table data structure.

36. A ~~distributed system generating~~ rule compiler processor-readable non-transitory physical medium for generating a distributed system storing processor-issuable instructions to:

(a) obtain, via the processor, a distributed system execution request data structure;

(b) determine, via the processor, an output rule structured to comprise a declaration implementing polymorphic parameterization and ~~the output rule's~~a set of rule parameters associated with the output rule specified via the distributed system execution request data structure for a requested calculation, wherein a current rule is initialized to the output rule;

(c) query, via the processor, a rule lookup table data structure to determine a set of matching rules, corresponding to the current rule, based on ~~the current rule's~~an identification tuple associated with the current rule;

(d) select, via the processor, the best matching rule from the set of matching rules, wherein ~~the best matching rule's~~a set of rule parameters associated with the best matching rule best corresponds to ~~the current rule's~~a set of rule parameters associated with the current rule;

(e) generate, via the processor, a logical dependency graph data structure by adding logical dependency graph nodes and logical dependency graph edges corresponding to the best matching rule, precedent rules of the best matching rule, and precedent rules of each of the precedent rules of the best matching rule~~precedent rule~~;

(f) determine, via the processor, an execution complexity gauge value associated with the generated logical dependency graph data structure;

(g) determine, via the processor, a set of distributed worker processes to be utilized to execute the requested calculation;

(h) divide, via the processor, the generated logical dependency graph data structure into a set of subgraphs based on the execution complexity gauge value, wherein each worker process in the set of distributed worker processes is assigned a subgraph from the set of subgraphs;

(i) initialize, via the processor, each respective worker process in the set of distributed worker processes with the subgraph assigned to the respective worker process;

(j) coordinate, via the processor, execution of the requested calculation by worker processes in the set of distributed worker processes, wherein each respective worker process calculates results for logical dependency graph nodes in the subgraph assigned to the respective worker process; and

(k) obtain, via the processor, a computation result of ~~the~~ a logical dependency graph node from the logical dependency graph nodes corresponding to the output rule, from the worker process in the set of distributed worker processes that is assigned the subgraph that contains the logical dependency graph node corresponding to the output rule.

37. A processor-implemented ~~distributed system generating~~ rule compiler method for generating a distributed system, comprising executing processor-implemented instructions to:

(a) obtain, via the processor, a distributed system execution request data structure;

(b) determine, via the processor, an output rule structured to comprise a declaration implementing polymorphic parameterization and ~~the output rule's~~ a set of rule parameters associated with the output rule specified via the distributed system execution request data structure for a requested calculation, wherein a current rule is initialized to the output rule;

(c) query, via the processor, a rule lookup table data structure to determine a set of matching rules, corresponding to the current rule, based on ~~the current rule's~~an identification tuple associated with the current rule;

(d) select, via the processor, the best matching rule from the set of matching rules, wherein ~~the best matching rule's~~a set of rule parameters associated with the best matching rule best corresponds to ~~the current rule's~~a set of rule parameters associated with the current rule;

(e) generate, via the processor, a logical dependency graph data structure by adding logical dependency graph nodes and logical dependency graph edges corresponding to the best matching rule, precedent rules of the best matching rule, and precedent rules of each of the precedent rules of the best matching rule;

(f) determine, via the processor, an execution complexity gauge value associated with the generated logical dependency graph data structure;

(g) determine, via the processor, a set of distributed worker processes to be utilized to execute the requested calculation;

(h) divide, via the processor, the generated logical dependency graph data structure into a set of subgraphs based on the execution complexity gauge value, wherein each worker process in the set of distributed worker processes is assigned a subgraph from the set of subgraphs;

(i) initialize, via the processor, each respective worker process in the set of distributed worker processes with the subgraph assigned to the respective worker process;

(j) coordinate, via the processor, execution of the requested calculation by worker processes in the set of distributed worker processes, wherein each respective worker process calculates results for logical dependency graph nodes in the subgraph assigned to the respective worker process; and

(k) obtain, via the processor, a computation result of ~~the~~ a logical dependency graph node from the logical dependency graph nodes corresponding to the output rule, from the worker process in the set of distributed worker processes that is assigned the subgraph that contains the logical dependency graph node corresponding to the output rule.

38. The apparatus of claim 1, wherein the declaration of the output rule implements type polymorphism.

39. The apparatus of claim 1, wherein the declaration of the output rule implements value polymorphism.

40. The apparatus of claim 1, wherein the declaration of the output rule implements type and value polymorphism.

41. The apparatus of claim 1, wherein each rule in the set of matching rules is structured to comprise a declaration implementing polymorphic parameterization.