(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 June 2015 (11.06.2015)

(10) International Publication Number
WO 2015/085152 A1

(51) International Patent Classification: *G06F 9/44* (2006.01)

(21) International Application Number: PCT/US2014/068754

(22) International Filing Date: 5 December 2014 (05.12.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/912,057 5 December 2013 (05.12.2013) US
62/031,388 31 July 2014 (31.07.2014) US

(71) Applicant: **AB INITIO TECHNOLOGY LLC** [US/US]; 201 Spring Street, Lexington, Massachusetts 02421 (US).

(72) Inventors: **LARSON, Brond**; 49 Lantern Lane, Sharon, Massachusetts 02067 (US). **BAY, Paul**; 7 Gorham Street, Arlington, Massachusetts 02474 (US). **BROMLEY, Mark, H.**; 1194 Center Road, Lyndeborough, New Hampshire 03082 (US). **STEVENS, Daniell, Garrick**; 3 Chestnut St., Belmont, Massachusetts 02478 (US). **TSVETANOV, Tsvetan**; 48 Gilbert Street, Waltham, Massachusetts 02453 (US).

(74) Agent: **MASON, Elliott, J., III**; Occhiuti & Rohlicek LLP, 321 Summer Street, Boston, Massachusetts 02210 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) Title: MANAGING INTERFACES FOR DATAFLOW GRAPHS COMPOSED OF SUB-GRAPHS

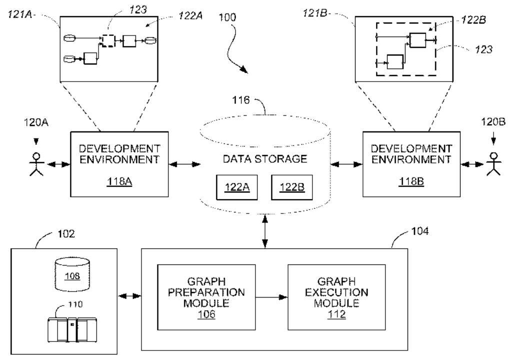

121A
123
122A
100
121B
122B
123
116
120A
120B
DEVELOPMENT ENVIRONMENT 118A
DATA STORAGE
122A
122B
DEVELOPMENT ENVIRONMENT 118B
102
108
110
104
GRAPH PREPARATION MODULE 106
GRAPH EXECUTION MODULE 112

FIG. 1A

(57) Abstract: Specifications of dataflow graphs are generated and/or combined. A sub-graph interface (123) of a dataflow graph includes one or more flow junctions. A flow junction (204D) representing a connection between: a flow of data outside the sub-graph interface, and a flow of data inside the sub-graph interface. In some embodiments, information is processed to generate a combined dataflow graph specification (300), including: identifying an association between the sub-graph interface and the first dataflow graph specification, for at least a first flow junction, determining a direction associated with transferring a value of a descriptor, and transferring a value of a descriptor according to the determined direction. In some embodiments, a link (229) is rendered between the first flow junction and a second flow junction based on user input indicating a relationship between a first descriptor and a second descriptor.

WO 2015/085152 A1

MANAGING INTERFACES FOR DATAFLOW GRAPHS COMPOSED OF SUB-GRAPHS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. Application Serial No. 61/912,057, filed on December 5, 2013, and to U.S. Application Serial No. 62/031,388, filed on July 31, 2014.

BACKGROUND

This description relates to managing interfaces for sub-graphs in a dataflow graph.

Many software applications exist for processing data. Some of these software applications are specified as dataflow graphs. Dataflow graphs typically include a number of data processing components, which are interconnected by links, sometimes referred to as "flows."

When a dataflow graph is being executed, data (e.g., a dataset) is received from a database or from some other data storage or data queueing system. The received data advances through the dataflow graph by propagating through the flows and into the components according to dependencies defined by the interconnection of the components and flows. Each component processes data that it receives according to a predetermined function associated with the component before providing the processed data as output data via a flow. At the output of the dataflow graph the processed data is, for example, stored in another data storage or data queueing system, provided to another downstream system, or presented to a user.

A developer of a dataflow graph generally specifies the graph by dragging blocks representing components onto a graphical working area (or "canvas") provided by a graphical user interface and interconnecting the components with links representing data flows such that the dataflow graph implements a desired functionality. Once the developer is satisfied with his or her implementation of the dataflow graph, he or she can save the dataflow graph to storage for later use. In general, if the developer needs to alter the their implementation of the dataflow graph at a later time, he or she can cause the graphical user interface to read the saved dataflow graph from storage, make changes to the dataflow graph, and then re-save the modified dataflow graph to storage.

In some examples, one or more segments of a dataflow graph are themselves implemented using dataflow graphs, which are referred to as "sub-graphs." In those examples, a sub-graph is part of the dataflow graph. Thus, to alter a sub-graph that is used within a given dataflow graph, the developer requests the system to read the dataflow graph from disk, thereby enabling the developer to open the dataflow graph in the graphical user interface. Then the developer would, within the same graphical user interface, open the sub-graph segment so that the sub-graph can be edited. The developer can make changes to the sub-graph, and then the developer causes the dataflow graph with the modified sub-graph to be together re-saved to storage, thereby embedding the changes to the sub-graph in the saved dataflow graph.

SUMMARY

In one aspect, in general, a method for combining specifications of dataflow graphs includes receiving over an input device or port a plurality of dataflow graph specifications, including: a first dataflow graph specification (for a "sub-graph") that specifies two or more components connected by links representing flows of data between components, including a first component and a second component, with at least one component representing a computation applied to data flowing into a port of the component, and at least one component representing a computation providing data flowing out of a port of the component, and a second dataflow graph specification (for a "container graph") that specifies at least a third component, and at least one sub-graph interface, where the sub-graph interface includes at least one flow junction representing a connection between: (1) a flow of data outside the sub-graph interface to or from a port of the third component, and (2) a flow of data inside the sub-graph interface to or from a port of a component of the first dataflow graph specification. The method includes processing, using at least one processor, information including the first dataflow graph specification and the second dataflow graph specification, to generate a combined dataflow graph specification, the processing including: identifying an association between the sub-graph interface and the first dataflow graph specification, for at least a first flow junction on the sub-graph interface, determining a direction associated with transferring a value of a descriptor of a data or computational characteristic, and transferring a value of a descriptor of a data or computational characteristic from the first flow junction to a component specified by the first dataflow graph specification or a component specified by the second dataflow graph specification, according to the determined direction.

Aspects can include one or more of the following features.

The first dataflow graph specification includes at least one indicator that indicates that a descriptor associated with the first component is identical to a descriptor associated with the second component.

The determined direction corresponds to an inward transfer of a value of a first descriptor of a data or computational characteristic from the first flow junction on the sub-graph interface to the second component.

The first descriptor is provided to the first flow junction from the third component.

The first descriptor is provided to the first component from the second component.

The second dataflow graph specification includes at least one indicator that indicates that a descriptor associated with the third component is identical to a descriptor associated with the sub-graph interface.

The second dataflow graph specification includes at least one indicator that indicates that a descriptor associated with a fourth component is identical to a descriptor associated with the third component.

The determined direction corresponds to an outward transfer of a value of a first descriptor of a data or computational characteristic from the first flow junction on the sub-graph interface to the third component.

The first descriptor is provided to the first flow junction from the first component.

The first descriptor is provided to the fourth component from the third component.

The descriptor describes a data characteristic of data transferred between a port of the first component and a port of the second component.

The data characteristic includes a format of fields of records within the transferred data.

The descriptor describes a computational characteristic of a component of at least one of the first dataflow graph specification or the second dataflow graph specification.

The computational characteristic includes a degree of parallelism of execution of a computation represented by the component.

The first dataflow graph is encrypted.

The second dataflow graph is encrypted.

The two or more components of the first dataflow graph are distributed among a first plurality of sequentially executing phases in the first dataflow graph.

The second dataflow graph includes a plurality of components, the plurality of components and the sub-graph interface distributed among a second plurality of sequentially executing phases in the second dataflow graph.

Preparing the dataflow graph for execution includes determining a number of phases for inclusion in the dataflow graph based on the first plurality of sequentially executing phases and the second plurality of sequentially executing phases.

In another aspect, in general, software is stored in a non-transitory form on a computer-readable medium, for combining specifications of dataflow graphs, the software including instructions for causing a computing system to: receive over an input device or port a plurality of dataflow graph specifications, including: a first dataflow graph specification that specifies two or more components connected by links representing flows of data between components, including a first component and a second component, with at least one component representing a computation applied to data flowing into a port of the component, and at least one component representing a computation providing data flowing out of a port of the component, and a second dataflow graph specification that specifies at least a third component, and at least one sub-graph interface, where the sub-graph interface includes at least one flow junction representing a connection between: (1) a flow of data outside the sub-graph interface to or from a port of the third component, and (2) a flow of data inside the sub-graph interface to or from a port of a component of the first dataflow graph specification; and process, using at least one processor, information including the first dataflow graph specification and the second dataflow graph specification, to generate a combined dataflow graph specification, the processing including: identifying an association between the sub-graph interface and the first dataflow graph specification, for at least a first flow junction on the sub-graph interface, determining a direction associated with transferring a value of a descriptor of a data or computational characteristic, and transferring a value of a descriptor of a data or computational characteristic from the first flow junction to a component specified by the first dataflow graph specification or a component specified by the second dataflow graph specification, according to the determined direction.

In another aspect, in general, a computing system for combining specifications of dataflow graphs includes an input device or port configured to receive a plurality of dataflow graph specifications, including: a first dataflow graph specification that specifies two or more components connected by links representing flows of data between components, including a first component and a second component, with at least one component representing a computation applied to data flowing into a port of the component, and at least one component representing a computation providing data flowing out of a port of the component, and a second dataflow graph specification that specifies at least a third component, and at least one sub-graph interface, where the sub-graph interface includes at least one flow junction representing a connection between: (1) a flow of data outside the sub-graph interface to or from a port of the third component, and (2) a flow of data inside the sub-graph interface to or from a port of a component of the first dataflow graph specification. The system includes at least one processor configured to process information including the first dataflow graph specification and the second dataflow graph specification, to generate a combined dataflow graph specification, the processing including: identifying an association between the sub-graph interface and the first dataflow graph specification, for at least a first flow junction on the sub-graph interface, determining a direction associated with transferring a value of a descriptor of a data or computational characteristic, and transferring a value of a descriptor of a data or computational characteristic from the first flow junction to a component specified by the first dataflow graph specification or a component specified by the second dataflow graph specification, according to the determined direction.

In another aspect, in general, a method for specifying a dataflow graph includes: rendering, in a first user interface, a representation of a first dataflow graph, the rendering including: rendering a plurality of components of the first dataflow graph, at least one component that represents a computation associated with at least one of data flowing into an input port or data flowing out of an output port, and rendering a link between an output port of a first component of the first dataflow graph and an input port of a second component of the first dataflow graph, based on user input indicating a flow of data from the output port of the first component to the input port of the second component; and rendering, in a second user interface, a representation of a sub-graph of the first dataflow graph, the rendering including: rendering at least one sub-graph interface, the sub-graph interface including one or more flow junctions, where a first flow junction of the sub-graph interface represents a connection between: (1) a flow of data outside the sub-graph interface to or from a port of a third component of the first dataflow graph, and (2) a flow of data inside the sub-graph interface to or from a port of a first component of the sub-graph, rendering a link between the first flow junction and a second flow junction of the sub-graph interface based on user input indicating a relationship between: (1) a first descriptor of a data or computational characteristic associated with the first flow junction of the sub-graph interface, and (2) a second descriptor of a data or computational characteristic associated with the second flow junction of the sub-graph interface, and rendering a portion of the second user interface configured to receive a user input that specifies properties of respective flow junctions of a set of defined flow junctions of the sub-graph interface, wherein the properties include a direction associated with transferring a descriptor of a data or computational characteristic associated with a corresponding flow junction.

In another aspect, in general, a method for specifying a dataflow graph includes: rendering, in a first user interface, a representation of a first dataflow graph, the rendering including: rendering a plurality of components of the first dataflow graph, at least one component that represents a computation associated with at least one of data flowing into an input port or data flowing out of an output port, and rendering a link between an output port of a first component of the first dataflow graph and an input port of a second component of the first dataflow graph, based on user input indicating a flow of data from the output port of the first component to the input port of the second component; and rendering, in a second user interface, a representation of a sub-graph of the first dataflow graph, the rendering including: rendering at least one sub-graph interface, the sub-graph interface including one or more flow junctions, where a first flow junction of the sub-graph interface represents a connection between: (1) a flow of data outside the sub-graph interface to or from a port of a third component of the first dataflow graph, and (2) a flow of data inside the sub-graph interface to or from a port of a first component of the sub-graph, and rendering a link between the first flow junction and a second flow junction of the sub-graph interface based on user input indicating a relationship between: (1) a first descriptor of a data or computational characteristic associated with the first flow junction of the sub-graph interface, and (2) a second descriptor of a data or computational characteristic associated with the second flow junction of the sub-graph interface.

Aspects can include one or more of the following features.

The method further includes generating a data structure storing a representation of the first dataflow graph that includes: data representing a first set of one or more components that include one or more flows of data connected through one or more flow junctions outside of the sub-graph interface; and data representing a second set of one or more components that include one or more flows of data connected through one or more flow junctions inside of the sub-graph interface.

The relationship corresponds to the first descriptor being identical to the second descriptor.

The second user interface includes a portion configured to receive a user input that specifies properties of respective flow junctions of a set of defined flow junctions of the sub-graph interface.

The properties include a direction associated with transferring a descriptor of a data or computational characteristic associated with a corresponding flow junction.

The direction corresponds to an inward transfer of a value of a first descriptor of a data or computational characteristic from the first flow junction to the first component of the sub-graph.

The direction corresponds to an outward transfer of a value of a first descriptor of a data or computational characteristic from the first flow junction to the third component of the first dataflow graph.

The first descriptor describes a data characteristic of data transferred between the port of the third component of the first dataflow graph and the port of the first component of the sub-graph.

The data characteristic includes a format of fields of records within the transferred data.

The descriptor describes a computational characteristic of at least one of the third component of the first dataflow graph or the first component of the sub-graph.

The computational characteristic includes a degree of parallelism of execution of a computation represented by the third component of the first dataflow graph or the first component of the sub-graph.

The first user interface is generated by a first computing system.

The second user interface is generated by a second computing system different from the first computing system.

In another aspect, in general, software is stored in a non-transitory form on a computer-readable medium, for specifying a dataflow graph, the software including instructions for causing a computing system to: render, in a first user interface, a representation of a first dataflow graph, the rendering including: rendering a plurality of components of the first dataflow graph, at least one component that represents a computation associated with at least one of data flowing into an input port or data flowing out of an output port, and rendering a link between an output port of a first component of the first dataflow graph and an input port of a second component of the first dataflow graph, based on user input indicating a flow of data from the output port of the first component to the input port of the second component; and render, in a second user interface, a representation of a sub-graph of the first dataflow graph, the rendering including: rendering at least one sub-graph interface, the sub-graph interface including one or more flow junctions, where a first flow junction of the sub-graph interface represents a connection between: (1) a flow of data outside the sub-graph interface to or from a port of a third component of the first dataflow graph, and (2) a flow of data inside the sub-graph interface to or from a port of a first component of the sub-graph, and rendering a link between the first flow junction and a second flow junction of the sub-graph interface based on user input indicating a relationship between: (1) a first descriptor of a data or computational characteristic associated with the first flow junction of the sub-graph interface, and (2) a second descriptor of a data or computational characteristic associated with the second flow junction of the sub-graph interface.

In another aspect, in general, a computing system for specifying a dataflow graph includes: a first computing device configured to render, in a first user interface, a representation of a first dataflow graph, the rendering including: rendering a plurality of components of the first dataflow graph, at least one component that represents a computation associated with at least one of data flowing into an input port or data flowing out of an output port, and rendering a link between an output port of a first component of the first dataflow graph and an input port of a second component of the first dataflow graph, based on user input indicating a flow of data from the output port of the first component to the input port of the second component; and a second computing device configured to render, in a second user interface, a representation of a sub-graph of the first dataflow graph, the rendering including: rendering at least one sub-graph interface, the sub-graph interface including one or more flow junctions, where a first flow junction of the sub-graph interface represents a connection between: (1) a flow of data outside the sub-graph interface to or from a port of a third component of the first dataflow graph, and (2) a flow of data inside the sub-graph interface to or from a port of a first component of the sub-graph, and rendering a link between the first flow junction and a second flow junction of the sub-graph interface based on user input indicating a relationship between: (1) a first descriptor of a data or computational characteristic associated with the first flow junction of the sub-graph interface, and (2) a second descriptor of a data or computational characteristic associated with the second flow junction of the sub-graph interface.

Aspects can include one or more of the following advantages.

Among other advantages, the approaches to managing sub-graph interfaces, including dynamic linking of sub-graphs, facilitate code abstraction and re-use in dataflow graph development environments. Dynamic linking of a sub-graph into a subgraph interface involves delaying (e.g., until just before execution) the determination of which of multiple possible sub-graphs is to be used to implement the function associated with that sub-graph interface. Thus, a container graph (i.e., a dataflow graph that has one or more sub-graph interfaces) can be used as a template that can be re-used and customized. Template dataflow graphs can be shipped to customers. The customer can then supply graph logic that implements the sub-graph interfaces and customizes template dataflow graphs to the customer's needs and operational environment. Customers may include users of the customized dataflow graphs, and/or developers who provide customized dataflow graphs to other users.

Sub-graph interfaces provide the ability to make a dataflow graph generic across multiple implementations of the interface. Sub-graph interfaces provide the ability to develop graphs against the interface in advance of any implementation of the sub-graph interface. Sub-graph interfaces provide the ability to validate an implementation against the interface without a specific instance of use. Sub-graph interfaces provide the ability to differentiate the shipping location, permissions, encryption, or other attributes of the implementation of a sub-graph.

Some embodiments allow metadata in a container graph including a sub-graph interface to be derived from the implementation sub-graph associated with the sub-graph interface. Metadata may include, for example, a descriptor of data provided to or from a component (e.g., a record format), a computational characteristic of a component, or computing resources associated with a component. Advantageously, deriving metadata for the container graph from the implementation sub-graph makes the container graph configurable with respect to the type(s) of data emerging from or entering the implementation sub-graph. This can be especially useful when the implementation sub-graph is specified by a developer to read data having a specific record format and to pass the data to a predefined container graph for further processing. In some examples, the predefined container graph is specified by some other entity and may be read-only, encrypted or in some other way protected from being viewed and/or altered. In some examples, the implementation sub-graph may be read-only, encrypted or in some other way protected from being viewed and/or altered. Such a graph needs to be able to accommodate different types of record formats from the implementation sub-graph without requiring user intervention. Metadata in an implementation sub-graph associated with the sub-graph interface can also be derived from a container graph including that sub-graph interface, which may allow some flexibility in the use of an implementation sub-graph in different container graphs.

Developers can advantageously create and use libraries of implementations for sub-graph interfaces. Such libraries can serve to reduce development times and encourage code re-use.

Some embodiments allow for dependency analysis and introspection into sub-graph implementations of a sub-graph interface from a container graph.

A graphical user interface for developing container graphs can include graphical indications of multiple phases and how those phases apply to any sub-graph interfaces, which enables a developer to understand how different portions of the dataflow graph will be affected by those phases (e.g., which data flows will cross phase boundaries, causing the data traversing that data flow to be durably stored in a buffer).

A graphical user interface for developing sub-graph interfaces encourages developers to treat dynamically linked sub-graphs as subject to a strong abstraction barrier, requiring a sub-graph interface be developed separately from (and typically prior to) developing one or more sub-graph implementations, and in some examples, separately from (and typically prior to) developing a container graph.

Other features and advantages of the invention will become apparent from the following description, and from the claims.

DESCRIPTION OF DRAWINGS

FIG. 1A is a block diagram of a system configured to use dynamically linked sub-graphs.

FIG. 1B is a flowchart of different stages of preparing a dataflow graph for execution.

FIG. 2A is a dataflow graph including a sub-graph interface.

FIG. 2B is an implementation of the sub-graph interface.

FIG. 2C is a combined dataflow graph.

FIG. 2D is a sub-graph interface development user interface.

FIG. 2E is a port configuration tab of a sub-graph interface properties user interface.

FIG. 3A illustrates edit-time record format metadata propagation in the dataflow graph of FIG. 2A.

FIG. 3B illustrates edit-time record format metadata propagation in the implementation of the sub-graph interface of FIG. 2B.

FIG. 3C illustrates link-time record format metadata propagation in the combined dataflow graph of FIG. 2C.

FIG. 4A illustrates edit-time layout metadata propagation in the dataflow graph of FIG. 2A.

FIG. 4B illustrates edit-time layout metadata propagation in the implementation of the sub-graph interface of FIG. 2B.

FIG. 4C illustrates link-time layout metadata propagation in the combined dataflow graph of FIG. 2C.

DESCRIPTION

FIG. 1A shows an example of a data processing system 100 in which dynamically linked sub-graphs can be used. The system 100 includes a data source 102 that may include one or more sources of data such as storage devices or connections to online data streams, each of which may store or provide data in any of a variety of formats (e.g., database tables, spreadsheet files, flat text files, or a native format used by a mainframe). An execution environment 104 includes a graph preparation module 106 and a graph execution module 112. Very generally, the graph preparation module 106 assembles and links a specification of a dataflow graph (described in greater detail below) into a representation that is executable by the graph execution module 112. The execution environment 104 may be hosted, for example, on one or more general-purpose computers under the control of a suitable operating system, such as a version of the UNIX operating system. For example, the execution environment 104 can include a multiple-node parallel computing environment including a configuration of computer systems using multiple central processing units (CPUs) or processor cores, either local (e.g., multiprocessor systems such as symmetric multi-processing (SMP) computers), or locally distributed (e.g., multiple processors coupled as clusters or massively parallel processing (MPP) systems, or remote, or remotely distributed (e.g., multiple processors coupled via a local area network (LAN) and/or wide-area network (WAN)), or any combination thereof.

In some examples, the execution environment 104 reads data from the data source 102, processes the data by executing a dataflow graph on the data (e.g., by the graph execution module 112), and stores the processed data in a data storage system. Storage devices providing the data source 102 may be local to the execution environment 104, for example, being stored on a storage medium connected to a computer hosting the execution environment 104 (e.g., hard drive 108), or may be remote to the execution environment 104, for example, being hosted on a remote system (e.g., mainframe 110) in communication with a computer hosting the execution environment 104, over a remote connection (e.g., provided by a cloud computing infrastructure).

The graph execution module 112 uses the representation of the dataflow graph generated by the graph preparation module 106 to process the data provided by the data source 102. The output data may be stored back in the data source 102 or in a data storage system 116 accessible to the execution environment 104, or otherwise used. The data storage system 116 is also accessible to a development environment 118A in which a developer 120A is able to make changes to the specification of a dataflow graph within a user interface 121A. In this example there are multiple separate development environments for developing different dataflow graph specifications within separate user interfaces. For example, a first developer 120A develops a container graph 122A including a sub-graph interface 123 using a first user interface 121A of the development environment 118A. A second, possibly different developer 120B uses a second user interface 121B of a development environment 118B to develop an implementation sub-graph 122B to be loaded in a sub-graph interface 123 of the container graph 112A, the implementation sub-graph conforming to the sub-graph interface 123. In some examples, the development environment 118A or 118B is a system for developing applications as dataflow graphs that include vertices (representing data processing components or datasets) connected by directed links (representing flows of work elements, i.e., data) between the vertices. For example, such an environment is described in more detail in U.S. Publication No. 2007/0011668, titled "Managing Parameters for Graph-Based Applications," incorporated herein by reference. A system for executing such graph-based computations is described in U.S. Patent 5,966,072, titled "EXECUTING COMPUTATIONS EXPRESSED AS GRAPHS," incorporated herein by reference. Dataflow graphs made in accordance with this system provide methods for getting information into and out of individual processes represented by graph components, for moving information between the processes, and for defining a running order for the processes. This system includes algorithms that choose interprocess communication methods from any available methods (for example, communication paths according to the links of the graph can use TCP/IP or UNIX domain sockets, or use shared memory to pass data between the processes).

The execution module 104 can receive data from a variety of types of systems that may embody the data source 102, including different forms of database systems. The data may be organized as records having values for respective fields (also called "attributes" or "columns"), including possibly null values. When first reading data from a data source, the execution module 104 typically starts with some initial format information about records in that data source. In some circumstances, the record structure of the data source may not be known initially and may instead be determined after analysis of the data source or the data. The initial information about records can include, for example, the number of bits that represent a distinct value, the order of fields within a record, and the type of value (e.g., string, signed/unsigned integer) represented by the bits.

FIG. 1B shows an example of different stages of preparing dataflow graphs for execution and executing the dataflow graphs using the data processing system 100. During edit-time, any number of developers edit (150) different dataflow graphs, which may include one developer 120A editing a container graph, and another developer 120B editing an implementation sub-graph that implements a sub-graph interface included in that dataflow graph. In some cases, an implementation sub-graph may itself include a sub-graph interface that will be implemented by its own nested implementation sub-graph. A graph developer or graph user may then initiate a process carried out by the graph preparation module 106 of preparing dataflow graphs for execution using dynamic linking of sub-graphs. The graph preparation module 106 determines (152) if there are any unlinked sub-graph interfaces in a dataflow graph being prepared. If so, the module 106 links (154) the appropriate sub-graph. During this potentially recursive process of linking (called "link-time"), there may be various parameters associated with dataflow graphs that need to be evaluated, including parameters indicating which particular implementation sub-graph should be linked to a given sub-graph interface. After dynamic linking is complete, the module 106 compiles (156) the fully assembled dataflow graph into an executable form (at "compile-time"), and the execution module 104 executes (158) the compiled dataflow graph (at "run-time"). There may be certain parameters associated with a dataflow graph that are evaluated at compile-time or at run-time.

1 Container Graphs, Sub-Graphs, and Sub-Graph Interfaces

Referring to FIG. 2A a block diagram of a first dataflow graph 200 is configured to process data from a first input dataset 202A and a second input dataset 202B using a number of components and to store the resulting processed data in an output dataset 202C. The first dataflow graph 200 includes a first component 208A, a second component 208B, and a sub-graph interface 210. Very generally, the sub-graph interface 210 allows for a sub-graph to be dynamically loaded into the first dataflow graph. In some examples, the first dataflow graph 200 is referred to as a "container graph" due to its inclusion of a sub-graph interface.

Each of the components has one or more input ports for receiving input data and one or more output ports for providing output data. In general, each component applies a computation to the input data flowing into its input port(s) and provides the result of the computation as output via its output port(s). It is noted that in some examples, certain types of components may include only input ports or only output ports. The sub-graph interface includes one or more flow junctions, which define a point of connection between a flow in the container graph and the sub-graph associated with the sub-graph interface (as is described in further detail below). Each flow junction represents a connection (or "junction") between a flow of data to or from a port on a component of the first dataflow graph and a flow of data to or from a port on a component of the second dataflow graph. The ports of the datasets and components and the flow junctions of the sub-graph interface are interconnected by flows 206A-206E, which define how data propagates between the datasets, components, and the sub-graph interface of the first dataflow graph 200.

Specifically, for the first dataflow graph 200 of FIG. 2A, a first input port 204B included on the first component 208A is connected to a first output port 204A included on the second input dataset 202B using a first flow 206A. A first flow junction 204D included on the sub-graph interface 210 is connected to a second output port 204C included on the first input dataset 202A using a second flow 206B. A second flow junction 204F included on the sub-graph interface 210 is connected to a third output port 204E included on the first component 208A using a third flow 206C. A second input port 204H included on the second component 208B is connected to a third flow junction 204G included on the sub-graph interface 210 using a fourth flow 206D. Finally, a third input port 204J included on the output dataset 202C is connected to a fourth output port 204I included on the sub-graph interface 210 using a fifth flow 206E.

In the first dataflow graph 200, the first component 208A and the second component 208B are conventionally known dataflow graph components which implement functions such as sorting, joining, various data transformations, and so on.

The sub-graph interface 210 is a special type of node in a dataflow graph that allows for the management of the specification of a portion of the first dataflow graph 200 using dynamic linking of sub-graphs. In some examples, the sub-graph interface 210 receives a parameter input P_S 212 which includes a path to a second dataflow graph specification on disk. Just prior to execution of the first dataflow graph 200, the second dataflow graph is dynamically linked into the first dataflow graph 200, essentially taking the place of the sub-graph interface 210.

In general, in order for the dynamic linking of the second dataflow graph into the first dataflow graph 200 to be possible, the second dataflow graph must conform to an interface defined by the sub-graph interface 210. That is, the second dataflow graph must have ports that are connected to the flow junctions of the sub-graph interface. By forcing the second dataflow graph to conform to the sub-graph interface 210, it is known, without inspection of the second dataflow graph, that flows connected to the ports of the sub-graph interface 210 in the first dataflow graph 200 can be directly connected to the ports of the sub-graph interface 210 in the second dataflow graph, forming a single flow between the ports in the two dataflow graphs.

Referring to FIG. 2B, one example of a second dataflow graph 201 conforms to the sub-graph interface 210 of FIG. 2A. In some examples, the second dataflow graph 201 is referred to as an "implementation sub-graph" since it conforms to and implements functionality for a sub-graph interface 210.

The second dataflow graph 201 is configured to process data from the first and second flow junctions 204D, 204F specified by the sub-graph interface 210 using a number of components and to provide the resulting processed data as output to a third flow junction 204G defined by the sub-graph interface 210. Each of the components has one or more input ports for receiving input data and one or more output ports for providing output data. The ports of the components are interconnected by flows 206F-206I, which define how data propagates between the sub-graph interface 210 and the components of the second dataflow graph 201.

Specifically, for the second dataflow graph 201 of FIG. 2B, a fourth input port 204K included on the third component 208C is connected to the second flow junction 204F included on the graph interface 210 using a sixth flow 206F. An fifth input port 204M included on the fourth component 208D is connected to the first flow junction 204D included on the graph interface 210 using a seventh flow 206G. A sixth input port 204N included on the fourth component 208D is connected to a fifth output port 204L included on the third component 208C using an eighth flow 206H. A third flow junction 204G included on the graph interface 210 is connected to a sixth output port 204O included on the fourth component 208D using a ninth flow 206I.

It is noted that in some examples, the connections between the ports of the components in the second dataflow graph 201 and the flow junctions (i.e., 206F, 206G, 206I) are not conventional flows but are instead bindings (i.e., associations) or terminal connectors between the flow junctions and the ports. When the second dataflow graph 201 is linked into the first dataflow graph 200, the bindings or terminal connectors are stripped and the ports of the components in the second dataflow graph 201 are directly connected to the flows of the first dataflow graph 200.

2 Dynamic Linking

Referring again to FIG. 2A, immediately before execution of the dataflow graph by the graph execution module 112, the graph preparation module 106 processes portions of the first dataflow graph 200 and the second dataflow graph 201 to prepare the first dataflow graph 200 for execution. Among other steps, the processing includes dynamic linking (i.e., binding) of the second dataflow graph 201 into the first dataflow graph 200 at the location of the sub-graph interface 210 in the first dataflow graph 200.

To dynamically link the second dataflow graph 201 into the first dataflow graph 200, the graph preparation module 106 analyzes the parameter input P_S 212 provided to the sub-graph interface 210 to determine which implementation sub-graph stored on disk is associated with the sub-graph interface 210. The determined implementation sub-graph (e.g., the second dataflow graph 201) is loaded and instantiated and its parameters and ports are bound into the first dataflow graph 200 to form a combined dataflow graph. At least some of the flow junctions of the sub-graph interface are then analyzed to determine a direction of metadata propagation, which corresponds to a direction of transfer of a descriptor of data or a computational characteristic (i.e., metadata) associated with the flow junctions. In the process of metadata propagation, for the at least some flow junctions, a descriptor of data or a computational characteristic is transferred from that flow junction to a component or a port on a component in the first dataflow graph 200 or a component or a port on a component in the second dataflow graph 201, according to the determined direction. This process of metadata propagation is described in detail below.

Referring to FIG. 2C, a combined dataflow graph 300 includes the first dataflow graph 200 of FIG. 2A with the second dataflow graph 201 of FIG. 2B linked in place of the sub-graph interface 210. Since the second dataflow graph 201 is an implementation of the sub-graph interface 210 and conforms to the sub-graph interface 210, all of the input and output ports of the second dataflow graph 201 are connected to the components of the first dataflow graph 200 via flows.

The combined dataflow graph 300 is executable by the graph execution module 112.

3 Metadata Propagation

In dataflow graphs, in general, it is important that metadata associated with the ports of components in the dataflow graph and/or metadata associated with the components themselves is managed. As described above, in some examples, metadata includes a descriptor of data (e.g., a record format for a port including a sequence of fields and data types of records flowing into or out of a port) or a computational characteristic (e.g., a partitioning or a layout for a component). In some examples, metadata may include an amount of memory a component may use, which computing resources a component may use, sortedness, compression method, character set, binary representation (e.g., big-endian, little-endian), or data transformations.

Metadata management can be accomplished manually, automatically, or by using a combination of manual and automatic metadata management. For manual metadata management, metadata is supplied, for example, by a graph developer or by a graph user. For automatic metadata management, metadata is propagated from portions of the graph for which metadata has been explicitly defined (e.g., by a graph developer or by a graph user) metadata to portions of the graph for which metadata has not been explicitly defined. Metadata propagation is used when metadata for a given port or component is not directly supplied by a graph user or developer. In such a case, the metadata for the given port or component is derived from other ports or components in the graph. The term metadata propagation as is used herein refers to this derivation process.

In the first dataflow graph 200 of FIG. 2A, one simple example of conventional metadata propagation occurs when no record format metadata is explicitly defined for the first input port 204B of the first component 208A. The first input port 204B is connected to the first output port 204A of the second input dataset 202B via the first flow 206A. In general, the record format of the data supplied by the second input dataset 202B is always explicitly known and the metadata associated with the first output port 204A is therefore explicitly defined. The explicitly defined metadata associated with the first output port 204A is propagated over the first flow 206A to the first input port 204B where it is associated with the first input port 204B.

The above described metadata propagation between the first output port 204A and the first input port 204B occurred over the first flow 206A in the same direction that data flows through the dataflow graph (i.e., from left to right). However, in some examples, metadata propagates over flows in a direction opposite to the direction that data flows through the dataflow graph. For example, the explicitly defined metadata associated with the third input port 204J of the output dataset 202C propagates over the fifth flow 206E to the fourth output port 204I of the second component 208B in a direction opposite to the direction that data flows over the fifth flow 206E.

In a conventional dataflow graph including conventional components and datasets this propagation of explicitly defined metadata through the dataflow graph results in all ports and components in the dataflow graph being associated with metadata. Any conflicts arising in metadata propagation are generally flagged for developer intervention. However, metadata propagation for dataflow graphs including a sub-graph interface is generally handled differently from metadata propagation for dataflow graphs including only conventional components. In particular, metadata may be propagated in two stages: an edit-time metadata propagation stage and a link-time metadata resolution stage.

Using this two stage approach, at edit-time, the developer of the container graph (i.e., the graph including a sub-graph interface) and the developer of the implementation sub-graph (i.e., the sub-graph which conforms to the sub-graph interface) that will be linked in place of the sub-graph interface do not need to be aware of each other's metadata. Without access to the propagation information, conventional metadata propagation would have no way of knowing whether metadata should be propagated in a direction "inward" into the implementation sub-graph (i.e., the container graph acts as a source of metadata for the implementation sub-graph) or in a direction "outward" from the implementation sub-graph (i.e., the container graph acts as a sink for metadata from the implementation sub-graph).

To facilitate metadata propagation in a dataflow graph including a sub-graph interface, each flow junction of the sub-graph interface specifies a direction of metadata propagation. Metadata is associated with the flow junction during the edit-time metadata propagation, and then the associated metadata is transferred from the flow junction at link-time according to the direction of metadata propagation. In some examples, the set of possible directions of metadata propagation includes "inward" propagation and "outward" propagation. The particular inward or outward value of the direction can be stored in the form of a Boolean variable that can be retrieved, for example, for such propagation or otherwise determining the indicated direction of metadata propagation.

3.1.1 Inward Metadata Propagation

When a flow junction on the sub-graph interface is declared as having a metadata propagation direction of "inward," metadata propagation in the container graph supplies a metadata definition via the flow connected to the flow junction (and eventually to a port connected to a flow (or binding) in the implementation sub-graph).

That is, in the container graph, edit-time metadata propagation treats the flow junction as a metadata sink. In particular, edit-time metadata propagation in the container graph propagates metadata between conventional components in the container graph using conventional metadata propagation as is described above. When metadata is propagated to a flow junction on a sub-graph interface with an "inward" metadata propagation direction, the metadata can propagate no further. This metadata is maintained at the flow junction until the implementation sub-graph is linked into the container graph and link-time metadata resolution can be performed (as is described below).

In the implementation sub-graph, edit-time metadata propagation treats the flow junction having an "inward" metadata propagation direction as a metadata source. However, in the case of the implementation sub-graph, the actual metadata value at the flow junction on the sub-graph interface is unknown (since it is only available from the container graph at link-time). Thus, in order to perform edit-time propagation for the implementation sub-graph, a placeholder value is assigned for the metadata at the flow junction and is propagated through the implementation sub-graph using conventional metadata propagation as is described above. The placeholder value is resolved when the implementation sub-graph is linked into the container graph and link-time metadata resolution is performed.

During the link-time metadata resolution stage, with the implementation sub-graph linked into the container graph, the metadata that was maintained at the flow junction with an "inward" metadata propagation direction is transferred into the implementation sub-graph and to the appropriate ports of the components of the implementation sub-graph. In some examples, link-time metadata resolution resolves the placeholder value at the flow junction in the implementation sub-graph by simply replacing the placeholder value with the metadata value that was maintained at the corresponding flow junction in the container graph.

In some examples, “inward” metadata propagation is the default, and is appropriate even when the graph container the sub-graph interface will always be supplying an explicit metadata definition.

3.1.2 Outward Metadata Propagation

When a flow junction on the sub-graph interface is declared as having a metadata propagation direction of “outward,” metadata propagation in the implementation sub-graph supplies a metadata definition for the flow junction to the container graph.

That is, in the container graph, edit-time metadata propagation treats the flow junction as a metadata source even though an edit-time definition for the metadata is not present (since the definition is only available from the implementation sub-graph at link-time). In particular, edit-time metadata propagation in the container graph propagates metadata between conventional components in the container graph using conventional metadata propagation as is described above. When metadata is propagation is performed on a flow junction on the sub-graph interface having a metadata propagation direction of outward, a placeholder value is assigned for the metadata at the flow junction and the placeholder value is propagated through the container graph using conventional metadata propagation.

In the implementation sub-graph, edit-time metadata propagation treats the flow junction having an “outward” metadata propagation direction as a metadata sink. In particular, edit-time metadata propagation in the implementation sub-graph propagates metadata between conventional components in the container graph using conventional metadata propagation as is described above. When metadata is propagated to a flow junction on the sub-graph interface having an “outward” metadata propagation direction, the metadata can propagate no further. This metadata is maintained at the flow junction until the implementation sub-graph is linked into the container graph and link-time metadata resolution can be performed (as is described below).

During the link-time metadata resolution stage, with the implementation sub-graph linked into the container graph, the metadata that was maintained at the flow junction with an “outward” metadata propagation direction is transferred out to the container graph and to the appropriate ports of the components of the container graph. In some examples, link-time metadata resolution identifies the placeholder value at the flow junction in the container sub-graph and simply replaces the placeholder value with the metadata value that was maintained at the flow junction in the implementation sub-graph.

In some examples, the developer is not allowed to explicitly define the metadata for a flow junction with an “outward” metadata propagation direction.

4 Sub-Graph Interface Development Graphical User Interface

Referring to FIG. 2D, a sub-graph interface development graphical user interface 220 facilitates creation, inspection, and modification of a sub-graph interface 224. The sub-graph user interface development graphical user interface 220 includes a display window 222 which is used to present a graphical representation of the sub-graph interface 224 including a number of input flow junctions 226, output flow junctions 228, and one or more links 229 between the input and output flow junctions 226, 228 to a developer.

Very generally, the sub-graph interface 224 acts as a strict prototype to which any sub-graph implementations of the sub-graph interface 224 and any container graphs that include the sub-graph interface 224 must conform. As such, the outside of the sub-graph interface 224 in the graphical user interface 220 can be thought of as a placeholder for a separately developed container graph and the inside of the sub-graph interface 224 in the graphical user interface can be thought of as a placeholder for a separately developed sub-graph implementation.

The flow junctions 226, 228 are disposed on the boundary of the sub-graph interface 224 and act as a bridge, connecting a flow of data to or from a port of a component outside of the sub-graph interface 224 (e.g., a flow of data from a port of a component in a container graph) and a flow of data to or from a port of a component inside of the sub-graph interface 224 (i.e., a flow of data to a port of a component in a sub-graph implementation). In FIG. 2D, the sub-graph interface 224 includes two input flow junctions 226 (i.e., in0 and in1) and two output flow junctions 228 (i.e., out0, out1). However, any number of input flow junctions 226 and output flow junctions 228 can be included on the sub-graph interface 224.

Each of the input flow junctions 226 is associated with a direction (i.e., inward or outward) of propagation of a descriptor of data and/or a computational characteristic (i.e., metadata). As is described above, the direction of propagation determines whether the metadata propagated through the input flow junction 226 is provided by a container graph or a sub-graph implementation. If the direction of propagation for the input flow junction 226 is outward, the input flow junction 226 defines the record format of data passing through the input flow junction 226 and the layout of components attached to the input flow junction 226.

Similarly, each of the output flow junctions 228 is associated with a direction (i.e., inward or outward) of propagation of a descriptor of data and/or a computational characteristic (i.e., metadata). As is described above, the direction of propagation determines whether the metadata propagated through the output flow junction 228 is provided by a container graph or a sub-graph implementation. If the direction of propagation for the output flow junction 228 is outward, the output flow junction 228 defines the record format of data passing through the output flow junction 228 and the layout of components attached to the output flow junction 228.

In some examples, a developer can add flow junctions 226, 228 by right clicking on the boundary of the sub-graph interface 224 and selecting an "Add Input Flow Junction" or an "Add Output Flow Junction" menu item from a right click menu. In other examples, the graphical user interface 220 includes a special tool for adding flow junctions to the sub-graph interface 224. Similarly, flow junctions can be removed from the sub-graph interface 224 using a right click menu option or a specialized tool.

The one or more links 229 between the input and output flow junctions 226, 228 are indicative of relationships between data descriptors of data or computational characteristics (i.e., metadata) associated with the input flow junctions 226 and data descriptors of a data or computational characteristics (i.e., metadata) associated with the output flow junctions 226. In FIG. 2D, a single link 229 connects a first input flow junction, in0, to a first output flow junction, out0. The link 229 indicates that there is a relationship between the data or computational characteristic (i.e., metadata) associated with the first input flow junction, in0, and the data or computational characteristic (i.e., metadata) of the first output flow junction, out0. In some examples, the link 229 indicates that any data or computational characteristics (i.e., metadata) associated with the first input flow junction, in0, and the first output flow junction, out0, must be the same. This can be accomplished, for example, by the link 229 indicating that one characteristic is derived from the other.

In some examples, a developer can create links 229 between the input and output flow junctions 226, 228 by clicking on a first flow junction of a first type (e.g., an input flow junction) and drawing a line on the inside of the sub-graph interface 224 from the first flow junction to a second flow junction of a second type (e.g., an output flow junction), thereby connecting the two flow junctions using the line.

Referring to FIG. 2E, a flow junction configuration tab 230 of a sub-graph interface properties user interface 232 allows a user to configure one or more of the flow junctions of the sub-graph interface. The flow junction configuration tab includes a junction list control 234 , and a propagation control section 236 including a record format metadata propagation direction control 238, a layout metadata propagation direction control 240, a layout association control 242, a ‘can fan’ control 244, and a required control 246.

The flow junctions list control 234 displays a list of all flow junctions, categorized into an input flow junction category 248 and an output flow junction category 250. A developer can select one or more of the flow junctions from the flow junctions list control 234 for configuration. The developer can then configure the selected flow junctions using the controls 238, 240, 242, 244, 246 in the propagation control section 236. In particular, the developer can select whether the direction of record format metadata propagation for the selected flow junctions is inward or outward using the record format metadata propagation direction control 238. The developer can select whether the direction of layout metadata propagation for the selected flow junctions is inward or outward using the layout metadata propagation direction control 240. The developer can select a name of a layout parameter associated with the selected flow junctions using the layout association control 242. The developer can specify whether the selected flow junctions are allowed to fan-in or fan-out using the ‘can fan’ control 244. The developer can specify whether the selected flow junctions are required by the sub-graph interface using the required control 246.

In some examples, the flow junction configuration tab 230 of the sub-graph interface properties user interface 232 can be accessed by the developer right clicking on the sub-graph interface and selecting a properties item from a right click menu.

In some examples, the graphical user interface 220 of FIG. 2D can also be utilized to define a number of phases allowed in sub-graph interfaces of the sub-graph interface 224.

5 Examples

The following sections provide examples of metadata propagation for the dataflow graphs of FIGs. 2A and 2B. The dashed bold lines in FIGs. 3A-3C and FIGs. 4A-4C indicate metadata propagation with the arrowhead on the lines indicating a direction of metadata propagation.

5.1 Record Format Metadata Propagation

Referring to FIGs. 3A-3C, an example of record format metadata propagation in the first dataflow graph 200 of FIG. 2A and the second dataflow graph 201 of FIG. 2B is illustrated. Referring now to FIG. 3A, in the present example, it is assumed that the first flow junction 204D of the sub-graph interface 210 and the second flow junction 204F of the sub-graph interface 210 both have a metadata propagation direction of “inward” and the third flow junction 204G of the sub-graph interface has a metadata propagation direction of “outward.” It is also assumed that the only explicitly defined metadata in the first dataflow graph 200 is the record format **A** associated with the second output port 204C of the first input dataset 202A, the record format **B** associated with the first output port 204A of the second input dataset 202B, and the record format **C** associated with the third input port 204J of the output dataset 202C.

While the graph developer is editing the first dataflow graph 200, edit-time record format metadata propagation is performed on the first dataflow graph 200. In particular, the record format **A** is propagated from the second output port 204C of the first input dataset 202A to the first flow junction 204D of the sub-graph interface 210 via the second flow 206B. Since the first flow junction 204D has a metadata propagation direction of “inward,” the record format **A** is not propagated any further and is maintained at the first flow junction 204D for later use in the link-time metadata resolution stage.

The record format **B** is propagated from the first output port 204A of the second input dataset 202B to the first input port 204B of the first component 208A via the first flow 206A where the record format **B** is associated with the first input port 204B. The edit-time metadata propagation process determines that the first component 208A does not apply any transformation to the metadata and therefore propagates the record format **B** through the component and associates it with the third output port 204E of the first component 208A.

The record format **B** is then propagated from the third output port 204E of the first component 208A to the second flow junction 204F of the sub-graph interface 210 via the third flow 206C. Since the second flow junction 204F has a metadata propagation direction of "inward," the record format **B** is not propagated any further and is maintained at the second flow junction 204F for later use in link-time metadata resolution.

The record format **C** is then propagated from the third input port 204J of the output dataset 202C to the fourth output port 204I of the second component 208B via the fifth flow 206E where the record format **C** is associated with the fourth output port 204I.

Since the third flow junction 204G of the sub-graph interface 210 has a metadata propagation direction of "outward," the metadata associated with the flow junction is unknown at edit-time in the first dataflow graph 200 and is expected to be provided by the implementation sub-graph at link-time. For this reason, a placeholder record format $\mathbf{TBD_1}$ is temporarily associated with the third flow junction 204G. The placeholder record format $\mathbf{TBD_1}$ is propagated to the second input port 204H of the second component 208B via the fourth flow 206D where it is temporarily associated with the second input port 204H. The placeholder is maintained at the third flow junction 204G and the second input port 204H until link-time metadata resolution resolves the actual value of $\mathbf{TBD_1}$ and associates the actual value as the metadata for the third flow junction 204G and the second input port 204H.

Referring to FIG. 3B, while the graph developer is editing the second dataflow graph 201, edit-time metadata propagation is performed on the second dataflow graph 201. The second dataflow graph 201 has a single port, the sixth output port 204O of the fourth component 208D, with metadata explicitly defined as record format **E**.

As is described above, the second dataflow graph 201 conforms to the sub-graph interface 210 (i.e., the sub-graph interface included in the first dataflow graph 200). For this reason, the metadata propagation directions of the ports of the sub-graph interface 210 are the same in the second dataflow graph 201 as they were in the first dataflow graph 200. That is, the first flow junction 204D of the sub-graph interface 210 and the second flow junction 204F of the sub-graph interface 210 both have a metadata propagation direction of "inward" and the third flow junction 204G of the sub-graph interface has a metadata propagation direction of "outward."

Since the first flow junction 204D has a metadata propagation direction of "inward," the metadata associated with the port is unknown at edit-time in the second dataflow graph 201 and is expected to be provided by the container graph (i.e., the first dataflow graph 200) at link-time. For this reason, a placeholder record format **TBD_2** is temporarily associated with the first flow junction 204D. The placeholder record format **TBD_2** is propagated to the fifth input port 204M of the fourth component 208D via the seventh flow 206G where it is temporarily associated with the fifth input port 204M. The placeholder is maintained at the first flow junction 204D and the fifth input port 204M until link-time metadata resolution resolves the actual value of **TBD_2** and associates the actual value as the metadata for the fifth input 204M.

Similarly, since the second flow junction 204F has a metadata propagation direction of "inward," the metadata associated with the port is unknown at edit-time in the second dataflow graph 201 and is expected to be provided by the container graph (i.e., the first dataflow graph 200) at link-time. For this reason, a placeholder record format **TBD_3** is temporarily associated with the second flow junction 204F. The placeholder record format **TBD_3** is propagated to the fourth input port 204K of the third component 208C via the sixth flow 206F where it is temporarily associated with the fourth input port 204K. The edit-time metadata propagation process then determines that the third component 208C does not apply any transformation to the metadata and therefore propagates the record format **TBD_3** through the component and associates it with the fifth output port 204L of the third component 208C. **TBD_3** is then propagated to the sixth input port 204N of the fourth component 208D via the eighth flow 206H where it is temporarily associated with the sixth input port 204N.

The placeholder **TBD_3** is maintained at the second flow junction 204F and at the three ports 204K, 204L, 204N until link-time metadata resolution resolves the actual value of **TBD_3** and associates the actual value as the metadata for the ports.

The record format **E** is propagated from the sixth output port 204O of the fourth component 208D to the third flow junction 204G of the sub-graph interface 210 via the ninth flow 206I. Since the third flow junction 204G has a metadata propagation direction of "outward," the record format **F** is not propagated any further and is maintained at the third flow junction port 204G for later use in link-time metadata resolution.

Referring to FIG. 3C, just before run-time, the second dataflow graph 201 is linked into the first dataflow graph 200 resulting in the combined dataflow graph 300 and link-time metadata resolution is performed. To perform link-time metadata resolution, the properties of each of the flow junctions of the sub-graph interface 210 are analyzed to determine their associated direction of metadata propagation. For flow junctions of the sub-graph interface 210 having a metadata propagation direction of inward, record format metadata which is maintained at the flow junctions in the first dataflow graph 200 is transferred "inward" to the appropriate ports of the components of the second dataflow graph 201. Similarly, for flow junctions of the sub-graph interface 210 having a metadata propagation direction of outward, record format metadata which is maintained at the flow junctions in the second dataflow graph 201 is transferred outward to the appropriate ports of the components of the first dataflow graph 200.

In particular, the first flow junction 204D is analyzed and it is determined that the first flow junction 204D has a metadata propagation direction of "inward." It is then determined that the first flow junction 204D is associated with defined metadata record format **A** in the first dataflow graph 200 and with placeholder metadata $\mathbf{TBD_2}$ in the second dataflow graph 201. The link-time metadata resolution stage associates all ports associated with $\mathbf{TBD_2}$ in the second dataflow graph 201 (i.e., the fifth input port 204M) with record format **A**.

The second flow junction 204F is analyzed and it is determined that the second flow junction 204F has a metadata propagation direction of "inward." It is then determined that the second flow junction 204F is associated with defined metadata record format **B** in the first dataflow graph 200 and with placeholder metadata $\mathbf{TBD_3}$ in the second dataflow graph 201. The link-time metadata resolution stage associates all ports associated with $\mathbf{TBD_3}$ in the second dataflow graph 201 (i.e., the fourth input port 204K, the fifth output port 204L, and the sixth input port 204N) with record format **B**.

The third flow junction 204G is analyzed and it is determined that the third flow junction 204G has a metadata propagation direction of "outward." It is then determined that the third flow junction 204G is associated with defined metadata record format **E** in the second dataflow graph 201 and with placeholder metadata $\mathbf{TBD_1}$ in the first dataflow graph 200. The link-time metadata resolution stage associates all ports associated with **TBD$_1$** in the first dataflow graph 200 (i.e., the second input port 204H) with record format **E**.

As a result of the above-described edit-time metadata propagation and link-time metadata resolution processes, all of the ports in the combined dataflow graph 300 are associated with valid record format metadata.

5.2 Layout Metadata Propagation

Referring to FIGs. 4A-4C, an example of layout metadata propagation in the first dataflow graph 200 of FIG. 2A and the second dataflow graph 201 of FIG. 2B is illustrated. To facilitate layout metadata propagation, a designer of the sub-graph interface 212 specifies one or more layout metadata parameters which are associated with the sub-graph interface 212. For each of the one or more layout metadata parameters, the sub-graph interface designer specifies a direction of metadata propagation. With the layout metadata parameters defined, the designer of the sub-graph interface 212 associates each of the flow junctions of the sub-graph interface with one of the layout metadata parameters. In some examples, each flow junction has its own unique layout metadata parameter specified. In other examples, there are fewer layout metadata parameters than there are flow junctions and certain layout metadata parameters are associated with more than one flow junction. In such examples, flow junctions which share a common layout metadata parameter must all comply with the direction of metadata propagation and the propagated layout value associated with the common layout metadata parameter.

Referring now to FIG. 4A, in the present example, it is assumed that the sub-graph interface 212 has two layout metadata parameters associated with it: a first layout metadata parameter having an “outward” metadata propagation direction and a second layout metadata parameter having an “inward” metadata propagation direction. The first layout metadata parameter is associated with the first flow junction 204D and the second flow junction 204F. The a second layout metadata parameter is associated with the third flow junction 204G. It is also assumed that the layout metadata associated with the first component 208A and the second component 208B is not explicitly defined and is therefore defined by metadata propagation.

While the graph developer is editing the first dataflow graph 200, edit-time layout metadata propagation is performed on the first dataflow graph 200. Since layout metadata for the first component 208A is not explicitly defined, it is expected that its layout metadata will be propagated from the second flow junction 204F which is associated with the first layout metadata parameter having a metadata propagation direction of "outward." However, the layout metadata associated with the second flow junction 204F is unknown at edit-time in the first dataflow graph 200 and is expected to be provided by the implementation sub-graph at link-time. For this reason, placeholder layout metadata $\mathbf{TBD_1}$ is propagated to and temporarily associated with the first component 208A. The placeholder layout metadata is maintained at the first component 208A until link-time metadata resolution resolves the actual value of $\mathbf{TBD_1}$ and associates that actual value as the layout metadata for the first component 208A.

Similarly, layout metadata for the second component 208B is not explicitly defined. However, edit-time layout metadata propagation can analyze the configuration of the output dataset 202C which is connected to the fourth output port 204I of the second component 208B to infer the layout metadata for the second component 208B. In the example of FIG. 4A, metadata propagation analyzes the configuration of the output dataset 202C and determines that it is configured receive data from two parallel-executing instances of its upstream component (i.e., the second component 208B). Based on the configuration of the output dataset 202C, metadata propagation infers that two instances of the second component 208B execute in parallel (i.e., the second component 208C runs "two ways parallel," denoted as **2x** in the figure) during execution of the first dataflow graph 200. Based on this determination, metadata propagation associates the **2x** layout metadata with the second component 208B.

Metadata propagation then propagates the **2x** layout metadata from the second component 208B to the third flow junction 204G. Since the third flow junction 204G is associated with the second layout metadata parameter having a metadata propagation direction of "inward," the layout metadata **2x** is not propagated any further and is maintained at the third flow junction 204G for later use in link-time metadata resolution.

It is noted that since a file, which already has layout metadata, is connected to the first flow junction 204D in the first dataflow graph 200, no layout metadata propagation occurs through the first flow junction 204D.

Referring to FIG. 4B, while the graph developer is editing the second dataflow graph 201, edit-time layout metadata propagation is performed on the second dataflow graph 201. The third component 208C included in the second dataflow graph 201 has explicitly defined layout metadata indicating that, during execution of the second dataflow graph 201, one instance of the third component 208C runs (denoted as **1x** in the figure). The layout metadata for the fourth component 208D included in the second dataflow graph 201 is not explicitly defined and is therefore defined by metadata propagation.

Since the second flow junction 204F is associated with the first layout metadata parameter having a metadata propagation direction of "outward," the **1x** layout metadata for the third component 208C is propagated to the second flow junction 204F where it is maintained for later use in the link-time metadata resolution stage.

Since the layout metadata for the fourth component 208D is not explicitly defined, it is expected that its layout metadata will be propagated from the third flow junction 204G which is associated with the second layout metadata parameter having a metadata propagation direction of "inward." However, the layout metadata associated with the third flow junction 204G is unknown at edit time in the second dataflow graph 201 and is expected to be provided by the container graph at link-time. For this reason, placeholder layout metadata $\mathbf{TBD_2}$ is propagated to and temporarily associated with the fourth component 208D. The placeholder layout metadata is maintained at the fourth component 208D until link-time metadata resolution resolves the actual value of $\mathbf{TBD_2}$ and associates that actual value as the layout metadata for the fourth component 208D.

Referring to FIG. 4C, just before run-time, the second dataflow graph 201 is linked into the first dataflow graph 200 resulting in the combined dataflow graph 300 and link-time metadata resolution is performed. To perform link-time metadata resolution, the layout metadata parameters associated with each of the flow junctions of the sub-graph interface 210 are analyzed to determine the direction of metadata propagation associated with each of the flow junctions. For flow junctions of the sub-graph interface 210 having a metadata propagation direction of "inward," layout metadata which is maintained at the flow junctions in the first dataflow graph 200 is transferred inward to the appropriate components of the second dataflow graph 201. Similarly, for ports of the sub-graph interface 210 having a metadata propagation direction of "outward," layout metadata which is maintained at the flow junctions in the second dataflow graph 201 is transferred outward to the appropriate components of the first dataflow graph 200.

In particular, the first layout metadata parameter associated with the second flow junction 204F is analyzed and it is determined that the second flow junction 204F has a metadata propagation direction of "outward." It is then determined that the second flow junction 204F is associated with explicitly defined layout metadata (i.e., **1x**) in the second dataflow graph 201 and with placeholder layout metadata $\mathbf{TBD_1}$ in the first dataflow graph 200. The link-time metadata resolution stage associates all components associated with $\mathbf{TBD_1}$ in the first dataflow graph 200 (i.e., the first component 208A) with layout metadata **1x**.

The second layout metadata parameter associated with the third flow junction 204G is analyzed and it is determined that the third flow junction 204G has a metadata propagation direction of "inward." It is then determined that the third flow junction 204G is associated with placeholder layout metadata $\mathbf{TBD_2}$ in the second dataflow graph 201 and with explicitly defined layout metadata (i.e., **2x**) in the first dataflow graph 200. Link-time metadata resolution associates all components associated with $\mathbf{TBD_2}$ in the second dataflow graph 201 (i.e., the fourth component 208D) with layout metadata **2x**.

As a result of the above-described edit-time metadata propagation and link-time metadata resolution processes, all of the components in the combined dataflow graph 300 are associated with valid layout metadata.

In some of the examples described above, metadata propagation is described as an operation which copies explicitly defined metadata associated with a given port, terminal, or component to another port, terminal, or component which is has undefined metadata. The copied metadata is then associated with the other port, terminal, or component.

However, in some examples, rather than copying metadata, metadata propagation uses pointers to indicate associations between ports, terminals, or components with explicitly defined metadata and those with undefined metadata. For example, a given port with undefined metadata may have a metadata pointer which metadata propagation assigns to explicitly defined metadata associated with another, different port. At link-time no further propagation occurs. Instead, the pointer resolves causing the explicitly defined metadata to be associated with the other, different port.

6 Applications

In some examples, a given sub-graph interface can be associated with a library of implementation sub-graphs which conform to the sub-graph interface. A graph developer who places the sub-graph interface into a container graph can then easily choose from any of the implementation sub-graphs in the library during development.

In some examples, use of the sub-graph interfaces described above facilitates abstraction of code. For example, a given implementation of a sub-graph interface may exist in only one place on disk but may be used in many container graphs. Changes made to the implementation on disk affect the functionality of all of the container graphs where the implementation is used without requiring any modification to the container graphs.

7 Alternatives

In some examples, certain implementation sub-graphs are encrypted such that unauthorized users can not inspect the contents of the sub-graphs. In such examples, the sub-graph interface is capable of linking the encrypted implementation sub-graphs. In some examples, certain container graphs are encrypted such that unauthorized users can not inspect the contents of the container graphs.

In some examples, each flow junction of a graph interface includes one or more of the following properties: a label, a dataflow direction, an indication of whether the flow junction fan-in or fan-out, an indication of whether the flow junction is required, a metadata propagation direction, and a name of an associated layout metadata parameter.

In some examples, a designer of a sub-graph interface can specify a rule defining a metadata relationship between two or more flow junctions of the sub-graph interface. For example, the sub-graph interface designer can specify that an input flow junction and an output flow junction have the same metadata.

In some examples, the sub-graph interface can be implemented such that it supports multi-phase sub-graph implementations. Very-generally, a multi-phase graph is a graph which has its components separated into two or more “phases” which execute sequentially in a predefined order. For example, a multi-phase graph may include three phases: a first phase, a second phase, and a third phase, each including one or more components. In operation, the components of the second phase do not begin processing data until the components of the first phase complete their processing. Similarly, the components of the third phase do not begin processing data until the components of the second phase complete their processing.

To accommodate dynamic sub-graphs with multiple phases, the sub-graph interface includes a parameter that allows the designer of the sub-graph interface to specify whether the sub-graph interface is single phase or multi-phase. In the case that the sub-graph interface is specified as multi-phase, the designer does not need to specify a fixed number of phases for the sub-graph interface.

An example of requirements for implementations of a sub-graph interface that has been specified as multi-phase include the following requirements. Any implementation of the sub-graph interface must have all components connected to the input ports (or flow junctions) of the sub-graph interface in a single phase, φi. Furthermore, any implementation of the sub-graph interface must have all components connected to the output ports (or flow junctions) of the sub-graph interface in a single phase, $\varphi o > \varphi i$, where φo is the maximum phase in the implementation. Implementations of the sub-graph interface may have any number of phases that are purely internal to the sub-graph implementation.

When a container graph including a multi-phase sub-graph interface is being edited, the phase at the output of the sub-graph implementation is temporarily assumed to be its input phase + 1. This is sufficient for graph developers to determine when downstream components will be in the same phase. However the edit-time phases are not necessarily the same as the run-time phases.

When the implementation sub-graph is bound in place of the sub-graph interface, the overall phase information for the combined graph is updated as follows:

1) For a given sub-graph implementation, let I be the maximum of the phases of components writing to its input flow junctions. For all multi-phase sub-graph implementations at a given input phase value I, compute M, which is the maximum over the number of purely internal phases in the multi-phase sub-graph implementations.

2) The phase of all components in the graph with phase greater than I is incremented by M, and the phase deltas are propagated downstream.

3) The output phase O of a sub-graph implementation is set to the minimum of the adjusted phases of the set of downstream components of the sub-graph implementation's output ports.

4) This process is repeated for each phase in the combined graph which contains one or more multi-phase sub-graph implementations.

If φi, the input phase in the sub-graph implementation, is not 0 then some number m > 0 of the internal phases occur before the first input phase in the sub-graph implementation. These "pre-phases" of the sub-graph implementation are useful, for example, for creating a lookup file to be used in the main input processing. In this case, if I < φi, the global phase of all components may need to be incremented so that the m pre-phases can be run at positive phase numbers in the combined dataflow graph. This can carried out in a phase by phase manner, incrementing by the maximum of the pre-phases of all sub-graph implementations in the starting phase I (before the adjustment), propagating the phase deltas downstream, and iterating for the next phase.

In some examples, the above adjustment algorithm may result in gaps in the phase numbering of components in a sub-graph implementation (e.g., if there were other sub-graph implementations with internal phases in the same input phase as the sub-graph implementation). For example, consider two sub-graph implementations in an input phase 0: A, with 2 internal phases; and B, with one internal phase. The phase of the next downstream components of A and B is at least 1, since they are multi-phase graph implementations. The max of the internal phase counts is 2, and so the adjusted global output phase of both A and B is be 3. This means B will include a phase gap – its input phase will be 0, its only purely internal phase will be 1, but its output phase will be 3. Such a phase gap is not detrimental to the operation of the dataflow graph.

In some examples, purely internal phases of multiple sub-graph implementations in the same input phase, I, will overlap with each other. This can be problematic due to resource constraints and it is preferable that the sub-graph implementation has a private space of internal phases. For this reason, it is possible to allow a sub-graph implementation to opt out of sharing its internal phases with other sub-graph implementations. This can be done, for example, using a local parameter on the on the sub-graph implementation (e.g., named **private_internal_phasing)** that can be resolved to a Boolean value. When true, the computation of M in step 1 above is altered to be the sum of the internal phases of the sub-graph implementations at input phase I with **private_internal_phasing** set to True, added to the maximum of the internal phase count of the remaining sub-graph implementations at input phase I.

In some examples, the sub-graph interface may not have any flow terminals, but may still be useful for allowing a user to define different sets of resources (e.g., lookup files) to be used by a particular container graph, depending on which sub-graph implementation is loaded.

In some examples, the sub-graph interface is implemented by using a simple textual specification of the interface rather than by using a graphical user interface.

In some examples, the implementation of a sub-graph interface is simplified using a "wizard" style graphical user interface. For example, the wizard style graphical user interface would lead a user through the implementation of the sub-graph interface by asking a series of questions and automatically generating the sub-graph interface based on the user's answers. In one example, the wizard style graphical user interface includes a number of pages including but not limited to a pre-flight check page (i.e., an introduction page), a file names and locations page, a parameter definition page, a flow junction definition and metadata propagation page, a layout metadata page, and a summary page.

8 System Configurations

The approaches for managing sub-graphs and sub-graph interfaces described above can be implemented, for example, using a programmable computing system executing suitable software instructions or it can be implemented in suitable hardware such as a field-programmable gate array (FPGA) or in some hybrid form. For example, in a programmed approach the software may include procedures in one or more computer programs that execute on one or more programmed or programmable computing system (which may be of various architectures such as distributed, client/server, or grid) each including at least one processor, at least one data storage system (including volatile and/or non-volatile memory and/or storage elements), at least one user interface (for receiving input using at least one input device or port, and for providing output using at least one output device or port). The software may include one or more modules of a larger program, for example, that provides services related to the design, configuration, and execution of dataflow graphs. The modules of the program (e.g., elements of a dataflow graph) can be implemented as data structures or other organized data conforming to a data model stored in a data repository.

The software may be provided on a tangible, non-transitory medium, such as a CD-ROM or other computer-readable medium (e.g., readable by a general or special purpose computing system or device), or delivered (e.g., encoded in a propagated signal) over a communication medium of a network to a tangible, non-transitory medium of a computing system where it is executed. Some or all of the processing may be performed on a special purpose computer, or using special-purpose hardware, such as coprocessors or field-programmable gate arrays (FPGAs) or dedicated, application-specific integrated circuits (ASICs). The processing may be implemented in a distributed manner in which different parts of the computation specified by the software are performed by different computing elements. Each such computer program is preferably stored on or downloaded to a computer-readable storage medium (e.g., solid state memory or media, or magnetic or optical media) of a storage device accessible by a general or special purpose programmable computer, for configuring and operating the computer when the storage device medium is read by the computer to perform the processing described herein. The inventive system may also be considered to be implemented as a tangible, non-transitory medium, configured with a computer program, where the medium so configured causes a computer to operate in a specific and predefined manner to perform one or more of the processing steps described herein.

A number of embodiments of the invention have been described. Nevertheless, it is to be understood that the foregoing description is intended to illustrate and not to limit the scope of the invention, which is defined by the scope of the following claims. Accordingly, other embodiments are also within the scope of the following claims. For example, various modifications may be made without departing from the scope of the invention. Additionally, some of the steps described above may be order independent, and thus can be performed in an order different from that described.

What is claimed is:

1. A method for combining specifications of dataflow graphs, the method including:

receiving over an input device or port a plurality of dataflow graph specifications, including:

a first dataflow graph specification that specifies two or more components connected by links representing flows of data between components, including a first component and a second component, with at least one component representing a computation applied to data flowing into a port of the component, and at least one component representing a computation providing data flowing out of a port of the component, and

a second dataflow graph specification that specifies at least a third component, and at least one sub-graph interface, where the sub-graph interface includes at least one flow junction representing a connection between: (1) a flow of data outside the sub-graph interface to or from a port of the third component, and (2) a flow of data inside the sub-graph interface to or from a port of a component of the first dataflow graph specification; and

processing, using at least one processor, information including the first dataflow graph specification and the second dataflow graph specification, to generate a combined dataflow graph specification, the processing including:

identifying an association between the sub-graph interface and the first dataflow graph specification,

for at least a first flow junction on the sub-graph interface, determining a direction associated with transferring a value of a descriptor of a data or computational characteristic, and transferring a value of a descriptor of a data or computational characteristic from the first flow junction to a component specified by the first dataflow graph specification or a component specified by the second dataflow graph specification, according to the determined direction.

2. The method of claim 1, wherein the first dataflow graph specification includes at least one indicator that indicates that a descriptor associated with the first component is identical to a descriptor associated with the second component.

3. The method of claim 2, wherein the determined direction corresponds to an inward transfer of a value of a first descriptor of a data or computational characteristic from the first flow junction on the sub-graph interface to the second component.

4. The method of claim 3, wherein the first descriptor is provided to the first flow junction from the third component.

5. The method of claim 3 or 4, wherein the first descriptor is provided to the first component from the second component.

6. The method of claim 1, wherein the second dataflow graph specification includes at least one indicator that indicates that a descriptor associated with the third component is identical to a descriptor associated with the sub-graph interface.

7. The method of claim 6, wherein the second dataflow graph specification includes at least one indicator that indicates that a descriptor associated with a fourth component is identical to a descriptor associated with the third component.

8. The method of claim 7, wherein the determined direction corresponds to an outward transfer of a value of a first descriptor of a data or computational characteristic from the first flow junction on the sub-graph interface to the third component.

9. The method of claim 8, wherein the first descriptor is provided to the first flow junction from the first component.

10. The method of claim 8 or 9, wherein the first descriptor is provided to the fourth component from the third component.

11. The method of any of claims 1 to 10, wherein the descriptor describes a data characteristic of data transferred between a port of the first component and a port of the second component.

12. The method of any of claims 1 to 11, wherein the data characteristic includes a format of fields of records within the transferred data.

13. The method of any of claims 1 to 10, wherein the descriptor describes a computational characteristic of a component of at least one of the first dataflow graph specification or the second dataflow graph specification.

14. The method of any of claims 1 to 10 and 13, wherein the computational characteristic includes a degree of parallelism of execution of a computation represented by the component.

15. The method of any of claims 1 to 14, wherein the first dataflow graph is encrypted.

16. The method of any of claims 1 to 15, wherein the second dataflow graph is encrypted.

17. The method of any of claims 1 to 16, wherein the two or more components of the first dataflow graph are distributed among a first plurality of sequentially executing phases in the first dataflow graph.

18. The method of claim 17, wherein the second dataflow graph includes a plurality of components, the plurality of components and the sub-graph interface distributed among a second plurality of sequentially executing phases in the second dataflow graph.

19. The method of claim 17, wherein preparing the dataflow graph for execution includes determining a number of phases for inclusion in the dataflow graph based on the first plurality of sequentially executing phases and the second plurality of sequentially executing phases.

20. Software stored in a non-transitory form on a computer-readable medium, for combining specifications of dataflow graphs, the software including instructions for causing a computing system to:

receive over an input device or port a plurality of dataflow graph specifications, including:

a first dataflow graph specification that specifies two or more components connected by links representing flows of data between components, including a first component and a second component, with at least one component representing a computation applied to data flowing into a port of the component, and at least one component representing a computation providing data flowing out of a port of the component, and a second dataflow graph specification that specifies at least a third component, and at least one sub-graph interface, where the sub-graph interface includes at least one flow junction representing a connection between: (1) a flow of data outside the sub-graph interface to or from a port of the third component, and (2) a flow of data inside the sub-graph interface to or from a port of a component of the first dataflow graph specification; and

process, using at least one processor, information including the first dataflow graph specification and the second dataflow graph specification, to generate a combined dataflow graph specification, the processing including:

identifying an association between the sub-graph interface and the first dataflow graph specification,

for at least a first flow junction on the sub-graph interface, determining a direction associated with transferring a value of a descriptor of a data or computational characteristic, and

transferring a value of a descriptor of a data or computational characteristic from the first flow junction to a component specified by the first dataflow graph specification or a component specified by the second dataflow graph specification, according to the determined direction.

21. A computing system for combining specifications of dataflow graphs, the computing system including:

an input device or port configured to receive a plurality of dataflow graph specifications, including:

a first dataflow graph specification that specifies two or more components connected by links representing flows of data between components, including a first component and a second component, with at least one component representing a computation applied to data flowing into a port of the component, and at least one component representing a computation providing data flowing out of a port of the component, and

a second dataflow graph specification that specifies at least a third component, and at least one sub-graph interface, where the sub-graph interface includes at least one flow junction representing a connection between: (1) a flow of data outside the sub-graph interface to or from a port of the third component, and (2) a flow of data inside the sub-graph interface to or from a port of a component of the first dataflow graph specification; and

at least one processor configured to process information including the first dataflow graph specification and the second dataflow graph specification, to generate a combined dataflow graph specification, the processing including:

identifying an association between the sub-graph interface and the first dataflow graph specification,

for at least a first flow junction on the sub-graph interface, determining a direction associated with transferring a value of a descriptor of a data or computational characteristic, and

transferring a value of a descriptor of a data or computational characteristic from the first flow junction to a component specified by the first dataflow graph specification or a component specified by the second dataflow graph specification, according to the determined direction.

22. A method for specifying a dataflow graph, the method including:

rendering, in a first user interface, a representation of a first dataflow graph, the rendering including:

rendering a plurality of components of the first dataflow graph, at least one component that represents a computation associated with at least one of data flowing into an input port or data flowing out of an output port, and

rendering a link between an output port of a first component of the first dataflow graph and an input port of a second component of the first dataflow graph, based on user input indicating a flow of data from the output port of the first component to the input port of the second component; and

rendering, in a second user interface, a representation of a sub-graph of the first dataflow graph, the rendering including:

rendering at least one sub-graph interface, the sub-graph interface including one or more flow junctions, where a first flow junction of the sub-graph interface represents a connection between: (1) a flow of data outside the sub-graph interface to or from a port of a third component of the first dataflow graph, and (2) a flow of data inside the sub-graph interface to or from a port of a first component of the sub-graph,

rendering a link between the first flow junction and a second flow junction of the sub-graph interface based on user input indicating a relationship between: (1) a first descriptor of a data or computational characteristic associated with the first flow junction of the sub-graph interface, and (2) a second descriptor of a data or computational characteristic associated with the second flow junction of the sub-graph interface, and rendering a portion of the second user interface configured to receive a user input that specifies properties of respective flow junctions of a set of defined flow junctions of the sub-graph interface, wherein the properties include a direction associated with transferring a descriptor of a data or computational characteristic associated with a corresponding flow junction.

23. A method for specifying a dataflow graph, the method including:

rendering, in a first user interface, a representation of a first dataflow graph, the rendering including:

rendering a plurality of components of the first dataflow graph, at least one component that represents a computation associated with at least one of data flowing into an input port or data flowing out of an output port, and

rendering a link between an output port of a first component of the first dataflow graph and an input port of a second component of the first dataflow graph, based on user input indicating a flow of data from the output port of the first component to the input port of the second component; and

rendering, in a second user interface, a representation of a sub-graph of the first dataflow graph, the rendering including:

rendering at least one sub-graph interface, the sub-graph interface including one or more flow junctions, where a first flow junction of the sub-graph interface represents a connection between: (1) a flow of data outside the sub-graph interface to or from a port of a third component of the first dataflow graph, and (2) a flow of data inside the sub-graph interface to or from a port of a first component of the sub-graph, and rendering a link between the first flow junction and a second flow junction of the sub-graph interface based on user input indicating a relationship between: (1) a first descriptor of a data or computational characteristic associated with the first flow junction of the sub-graph interface, and (2) a second descriptor of a data or computational characteristic associated with the second flow junction of the sub-graph interface.

24. The method of claim 23, further including generating a data structure storing a representation of the first dataflow graph that includes:

data representing a first set of one or more components that include one or more flows of data connected through one or more flow junctions outside of the sub-graph interface; and

data representing a second set of one or more components that include one or more flows of data connected through one or more flow junctions inside of the sub-graph interface.

25. The method of claim 23 or 24, wherein the relationship corresponds to the first descriptor being identical to the second descriptor.

26. The method of any of claims 23 to 25, wherein the second user interface includes a portion configured to receive a user input that specifies properties of respective flow junctions of a set of defined flow junctions of the sub-graph interface.

27. The method of claim 26, wherein the properties include a direction associated with transferring a descriptor of a data or computational characteristic associated with a corresponding flow junction.

28. The method of claim 27, wherein the direction corresponds to an inward transfer of a value of a first descriptor of a data or computational characteristic from the first flow junction to the first component of the sub-graph.

29. The method of claim 27, wherein the direction corresponds to an outward transfer of a value of a first descriptor of a data or computational characteristic from the first flow junction to the third component of the first dataflow graph.

30. The method of any of claims 23 to 29, wherein the first descriptor describes a data characteristic of data transferred between the port of the third component of the first dataflow graph and the port of the first component of the sub-graph.

31. The method of claim 30, wherein the data characteristic includes a format of fields of records within the transferred data.

32. The method of any of claims 23 to 29, wherein the descriptor describes a computational characteristic of at least one of the third component of the first dataflow graph or the first component of the sub-graph.

33. The method of claim 32, wherein the computational characteristic includes a degree of parallelism of execution of a computation represented by the third component of the first dataflow graph or the first component of the sub-graph.

34. The method of any of claims 23 to 33, wherein the first user interface is generated by a first computing system.

35. The method of claim 34, wherein the second user interface is generated by a second computing system different from the first computing system.

36. Software stored in a non-transitory form on a computer-readable medium, for specifying a dataflow graph, the software including instructions for causing a computing system to:

render, in a first user interface, a representation of a first dataflow graph, the rendering including:

rendering a plurality of components of the first dataflow graph, at least one component that represents a computation associated with at least one of data flowing into an input port or data flowing out of an output port, and

rendering a link between an output port of a first component of the first dataflow graph and an input port of a second component of the first dataflow graph, based on user input indicating a flow of data from the output port of the first component to the input port of the second component; and

render, in a second user interface, a representation of a sub-graph of the first dataflow graph, the rendering including:

rendering at least one sub-graph interface, the sub-graph interface including one or more flow junctions, where a first flow junction of the sub-graph interface represents a connection between: (1) a flow of data outside the sub-graph interface to or from a port of a third component of the first dataflow graph, and (2) a flow of data inside the sub-graph interface to or from a port of a first component of the sub-graph, and

rendering a link between the first flow junction and a second flow junction of the sub-graph interface based on user input indicating a relationship between: (1) a first descriptor of a data or computational characteristic associated with the first flow junction of the sub-graph interface, and (2) a second descriptor of a data or computational characteristic associated with the second flow junction of the sub-graph interface.

37. A computing system for specifying a dataflow graph, the computing system including:

a first computing device configured to render, in a first user interface, a representation of a first dataflow graph, the rendering including:

rendering a plurality of components of the first dataflow graph, at least one component that represents a computation associated with at least one of data flowing into an input port or data flowing out of an output port, and

rendering a link between an output port of a first component of the first dataflow graph and an input port of a second component of the first dataflow graph, based on user input indicating a flow of data from the output port of the first component to the input port of the second component; and

a second computing device configured to render, in a second user interface, a representation of a sub-graph of the first dataflow graph, the rendering including:

rendering at least one sub-graph interface, the sub-graph interface including one or more flow junctions, where a first flow junction of the sub-graph interface represents a connection between: (1) a flow of data outside the sub-graph interface to or from a port of a third component of the first dataflow graph, and (2) a flow of data inside the sub-graph interface to or from a port of a first component of the sub-graph, and

rendering a link between the first flow junction and a second flow junction of the sub-graph interface based on user input indicating a relationship between: (1) a first descriptor of a data or computational characteristic associated with the first flow junction of the sub-graph interface, and (2) a second descriptor of a data or computational characteristic associated with the second flow junction of the sub-graph interface.

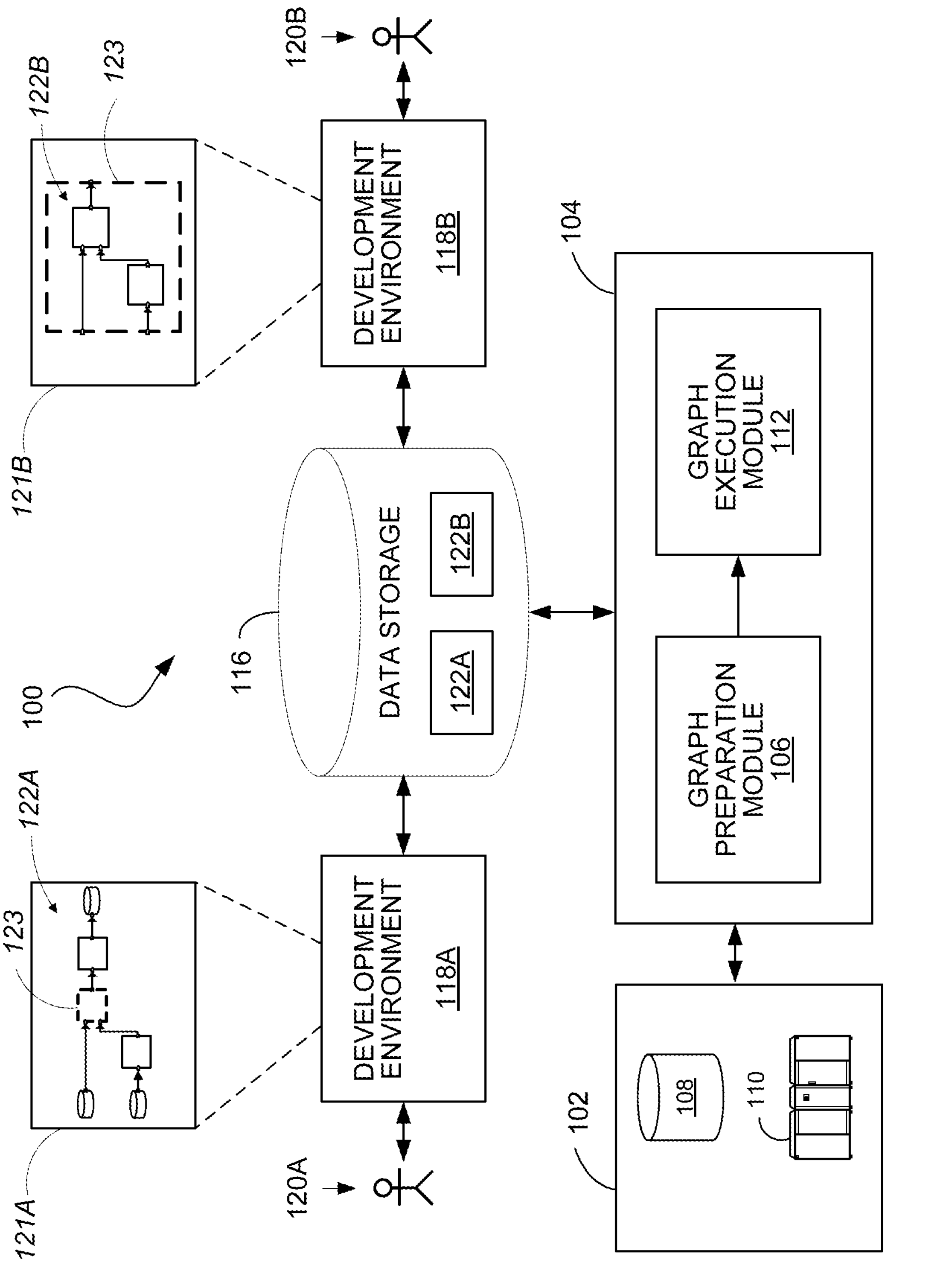

100
121A
122A
123
120A
DEVELOPMENT ENVIRONMENT
118A
116
DATA STORAGE
122A
122B
121B
122B
123
120B
DEVELOPMENT ENVIRONMENT
118B
104
GRAPH PREPARATION MODULE
106
GRAPH EXECUTION MODULE
112
102
108
110

FIG. 1A

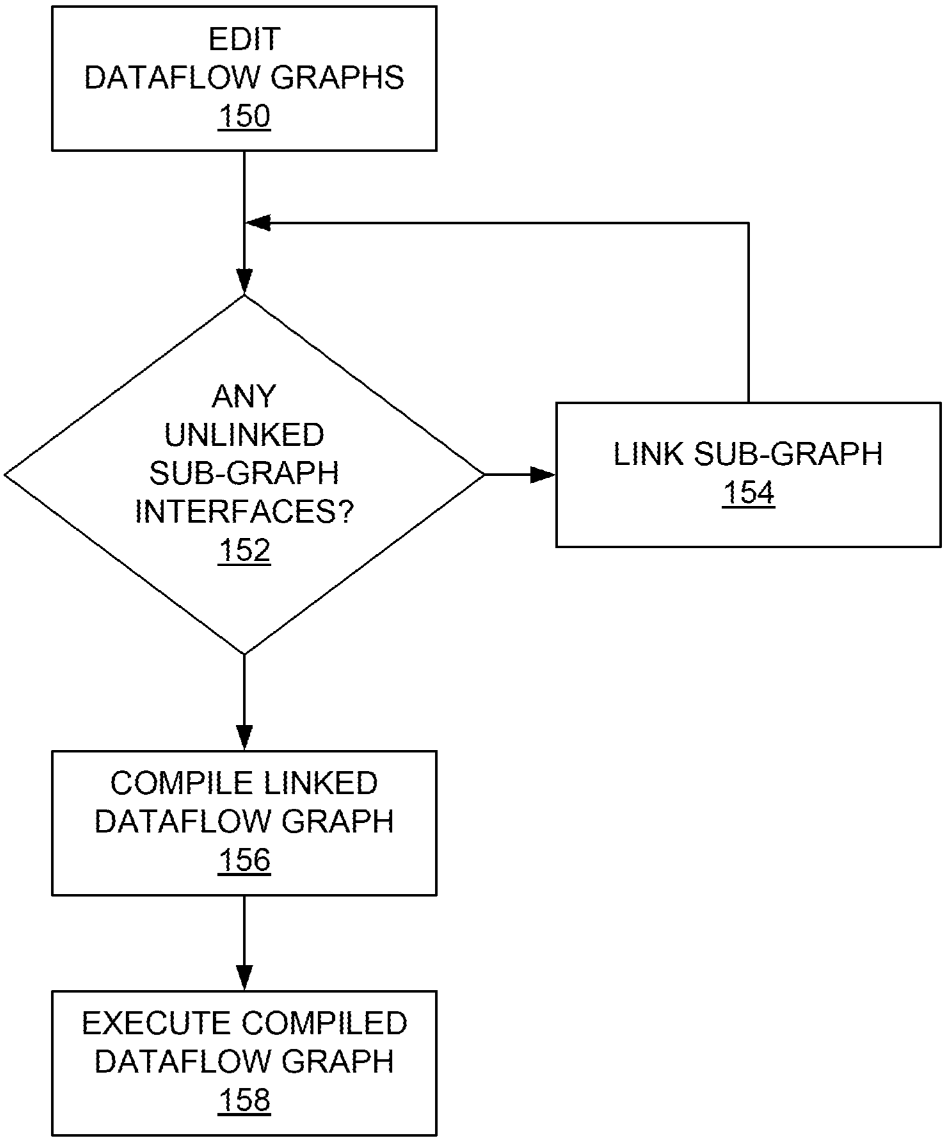

EDIT DATAFLOW GRAPHS 150
ANY UNLINKED SUB-GRAPH INTERFACES? 152
LINK SUB-GRAPH 154
COMPILE LINKED DATAFLOW GRAPH 156
EXECUTE COMPILED DATAFLOW GRAPH 158

FIG. 1B

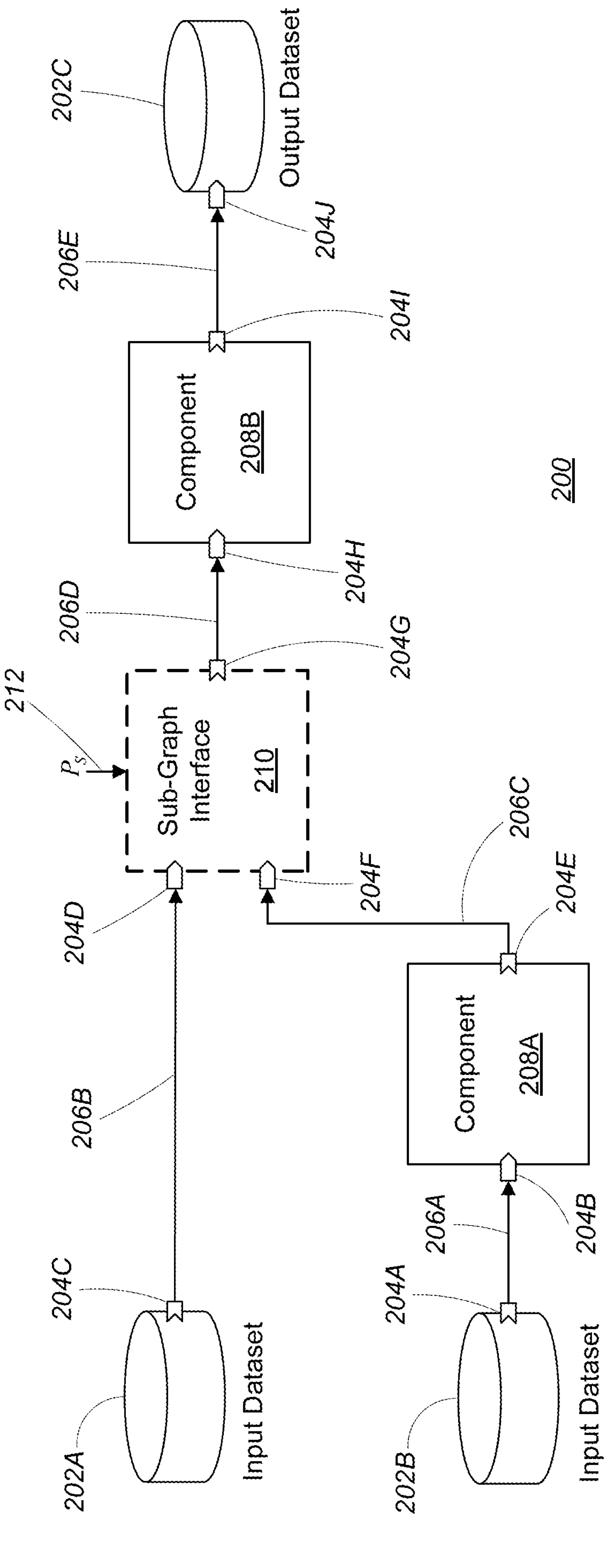

202A
Input Dataset
204C
206B
204D
202B
Input Dataset
204A
206A
204B
Component
208A
204E
206C
204F
212
P_S
Sub-Graph Interface
210
204G
206D
204H
Component
208B
204I
206E
204J
202C
Output Dataset
200

FIG. 2A

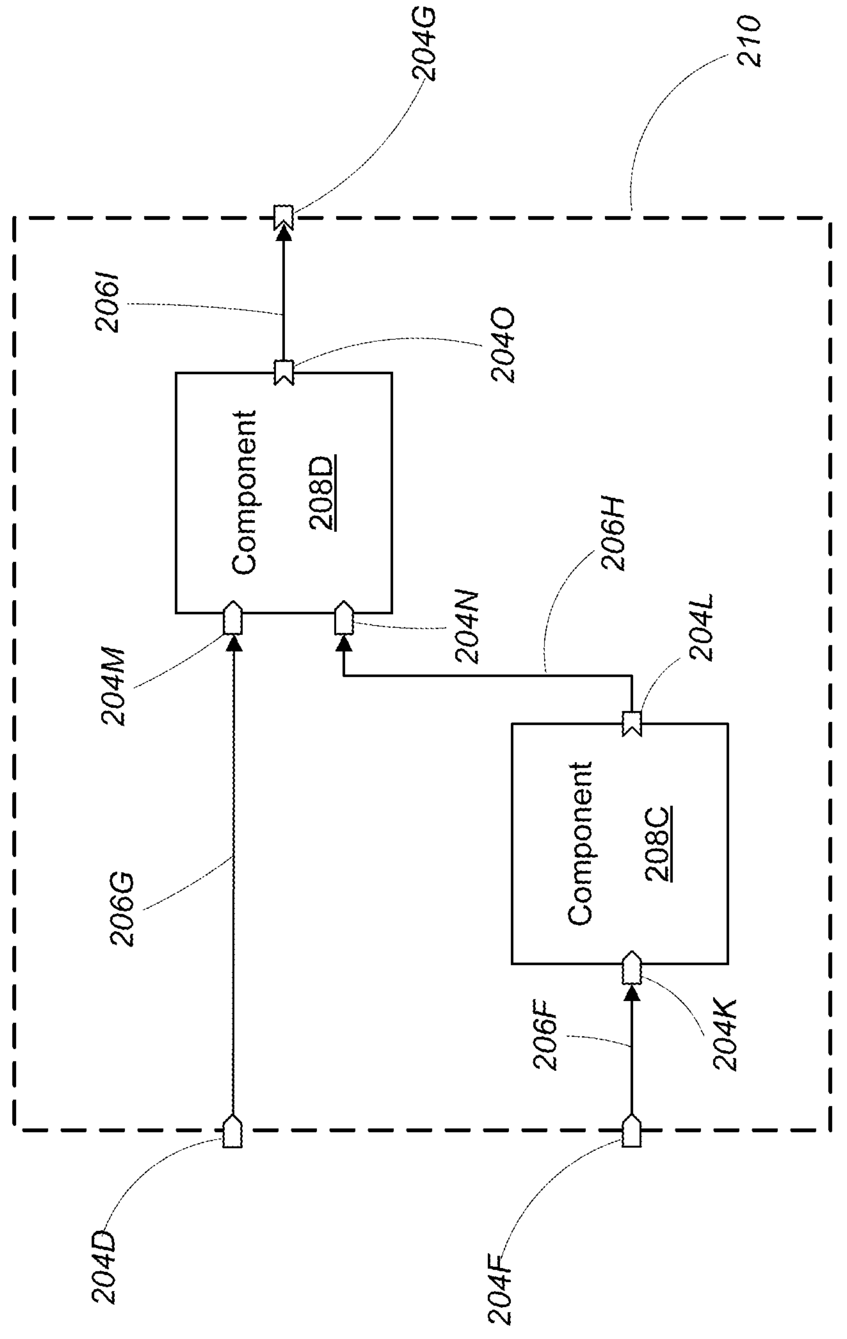

204G
210
206I
204O
Component
208D
206H
204N
204L
204M
Component
208C
206G
206F
204K
201
204D
204F

FIG. 2B

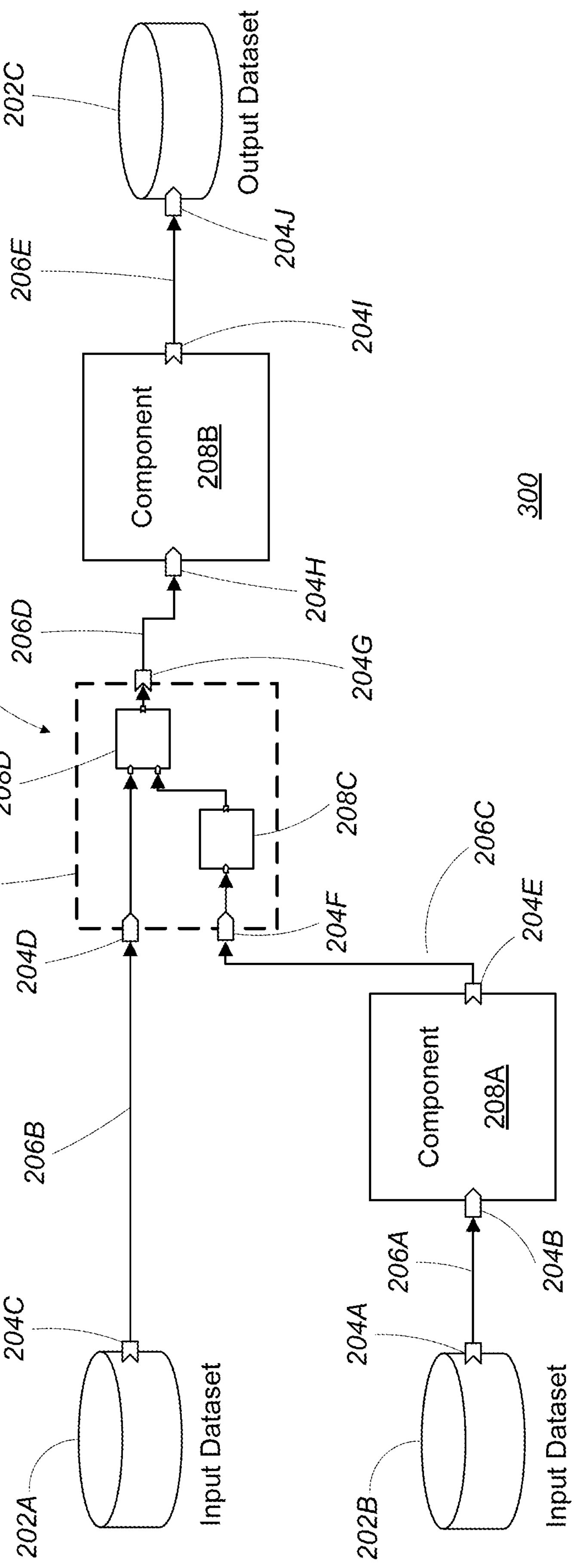

202C
Output Dataset
204J
206E
204I
Component
208B
204H
201
206D
204G
208D
210
208C
206C
204F
204E
204D
206B
Component
208A
204B
206A
204A
204C
202A
Input Dataset
202B
Input Dataset
300

FIG. 2C

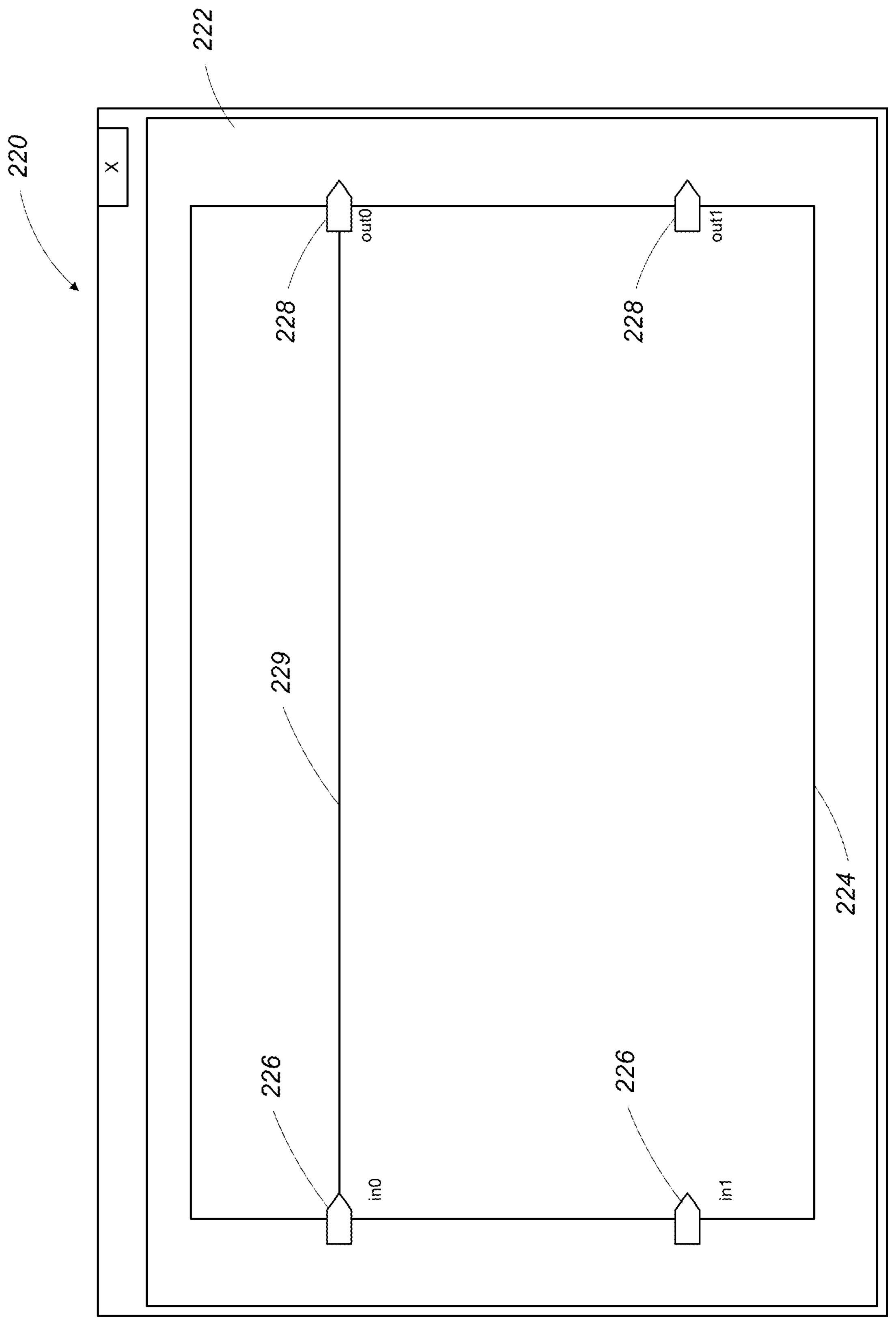


FIG. 2D

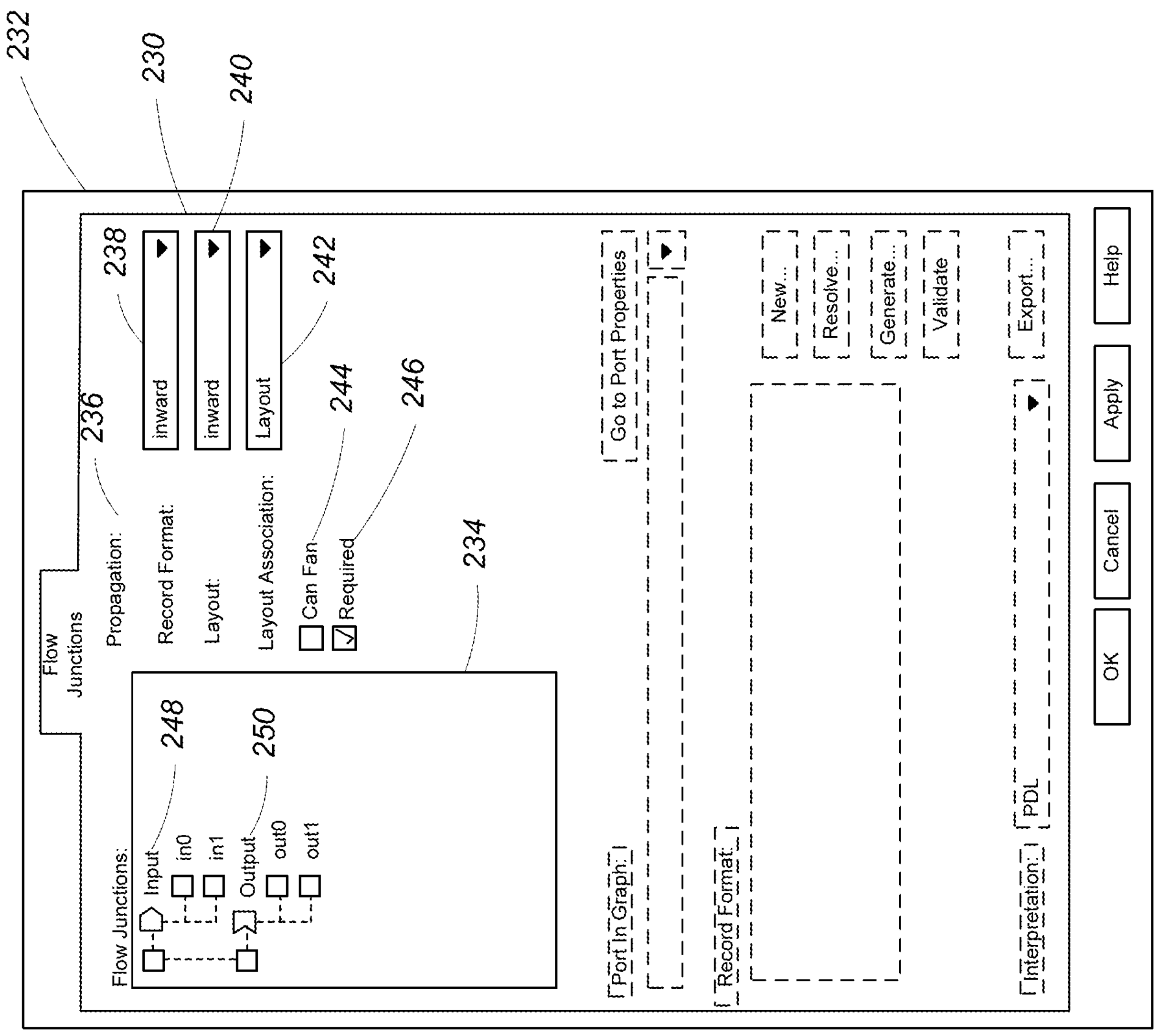

Propagation: inward
Record Format: inward
Layout: Layout
Layout Association:
☐ Can Fan
☑ Required

FIG. 2E

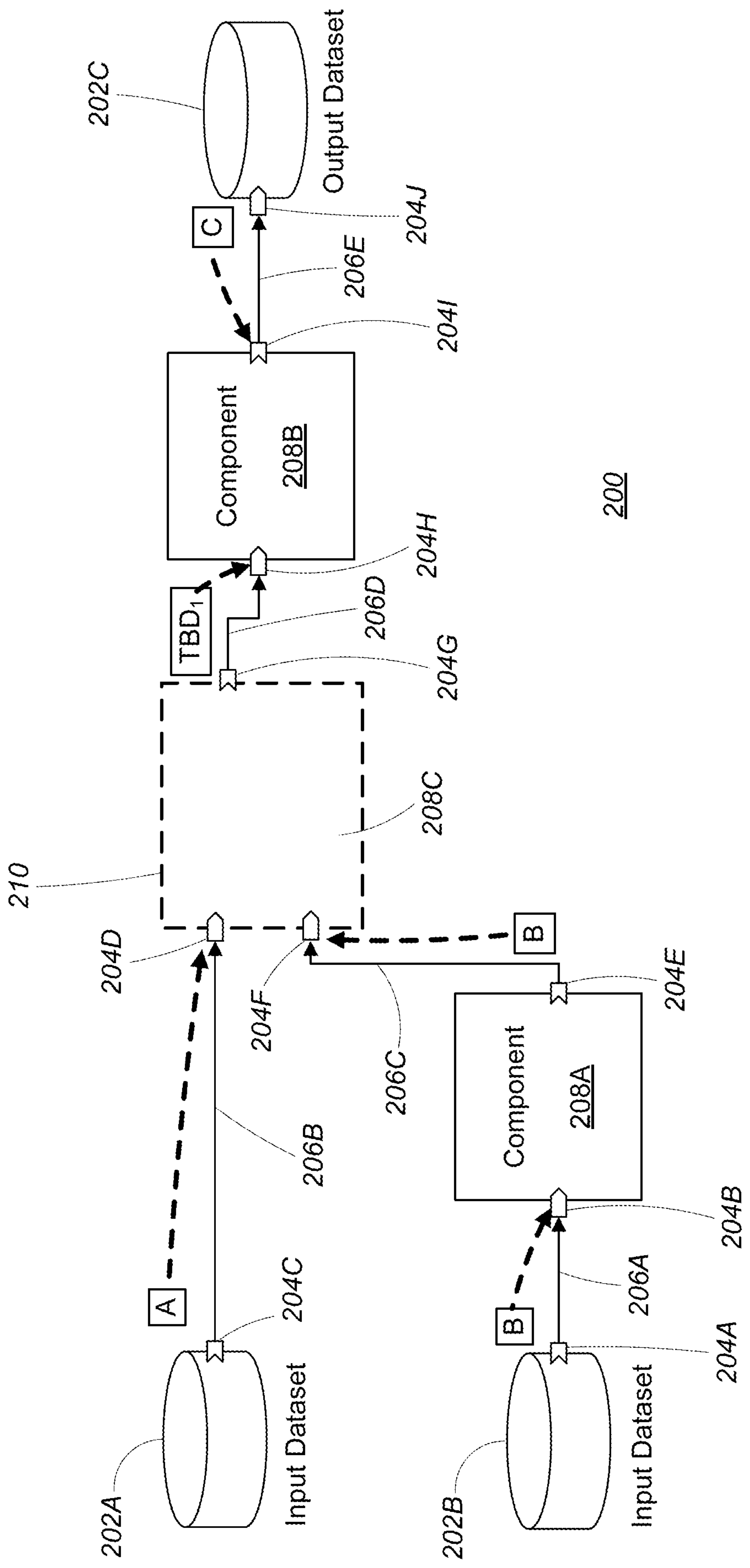

202A
Input Dataset
204C
A
206B
204D
210
208C
204F
206C
B
204E
Component
208A
204B
206A
B
204A
202B
Input Dataset
204G
206D
TBD$_1$
204H
Component
208B
204I
206E
C
204J
202C
Output Dataset
200

FIG. 3A

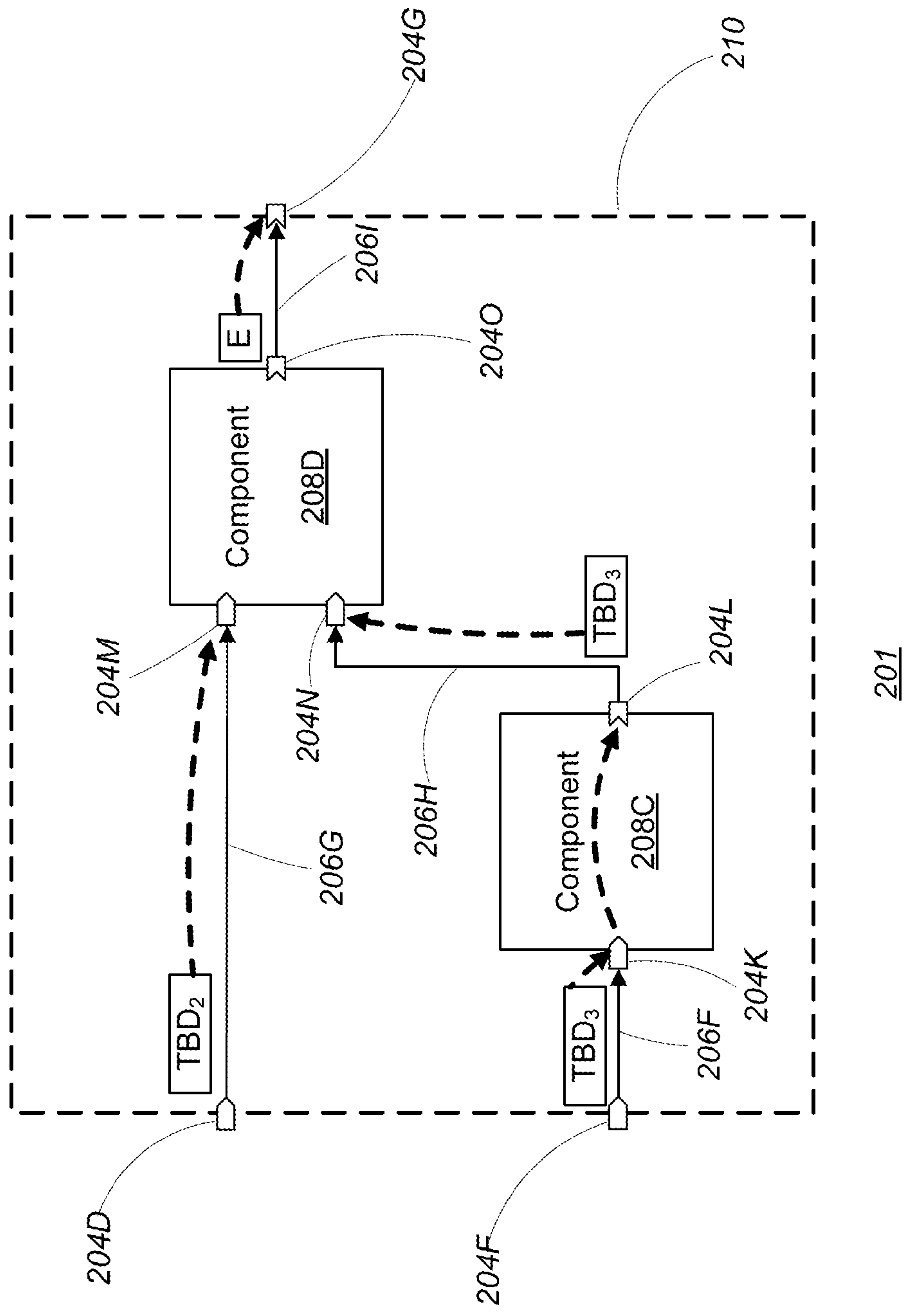
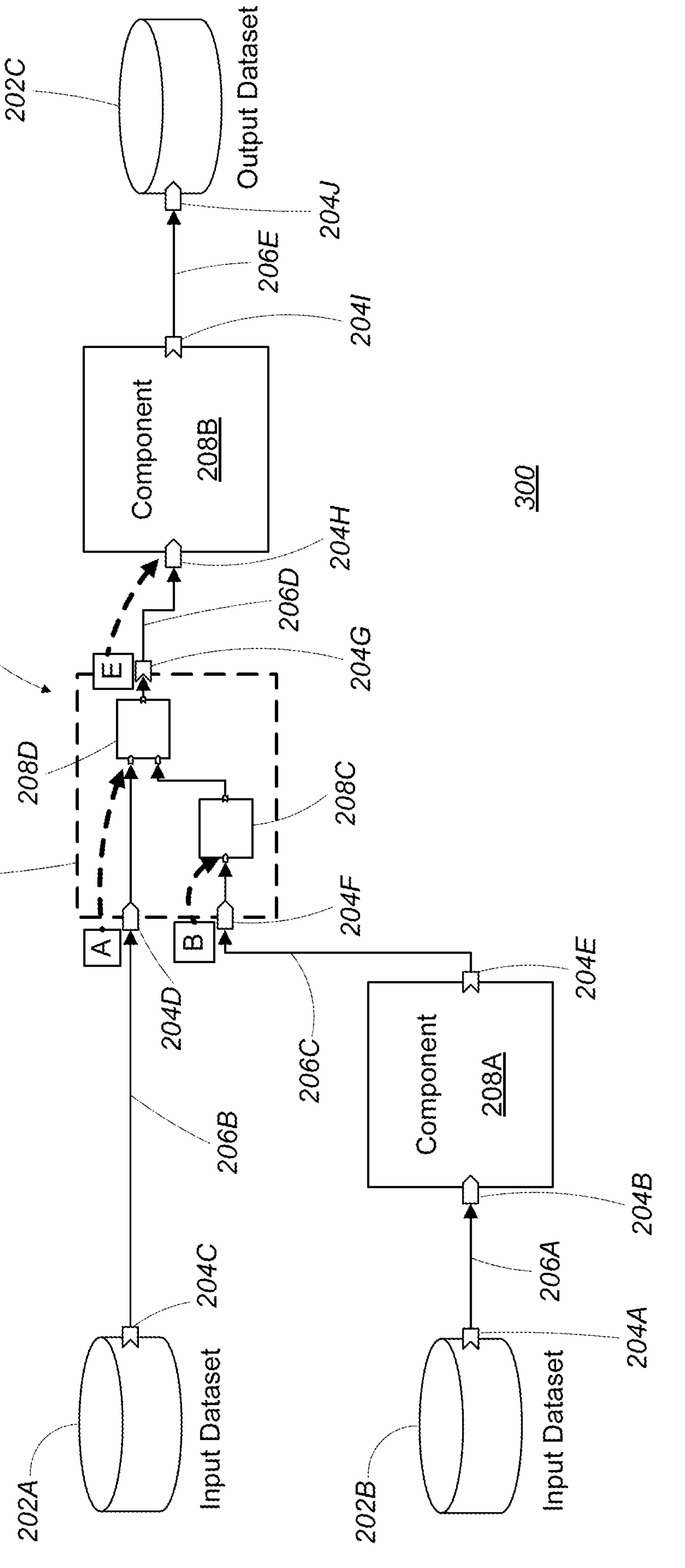

204G
210
206I
E
204O
Component
208D
TBD$_3$
204L
204M
204N
206H
Component
208C
206G
TBD$_2$
TBD$_3$
206F
204K
204D
204F
201

FIG. 3B



202C
Output Dataset
204J
206E
204I
Component
208B
300
201
204H
206D
204G
E
208D
210
208C
A
B
204F
204D
206C
204E
Component
208A
206B
204B
206A
204C
204A
202A
Input Dataset
202B
Input Dataset

FIG. 3C

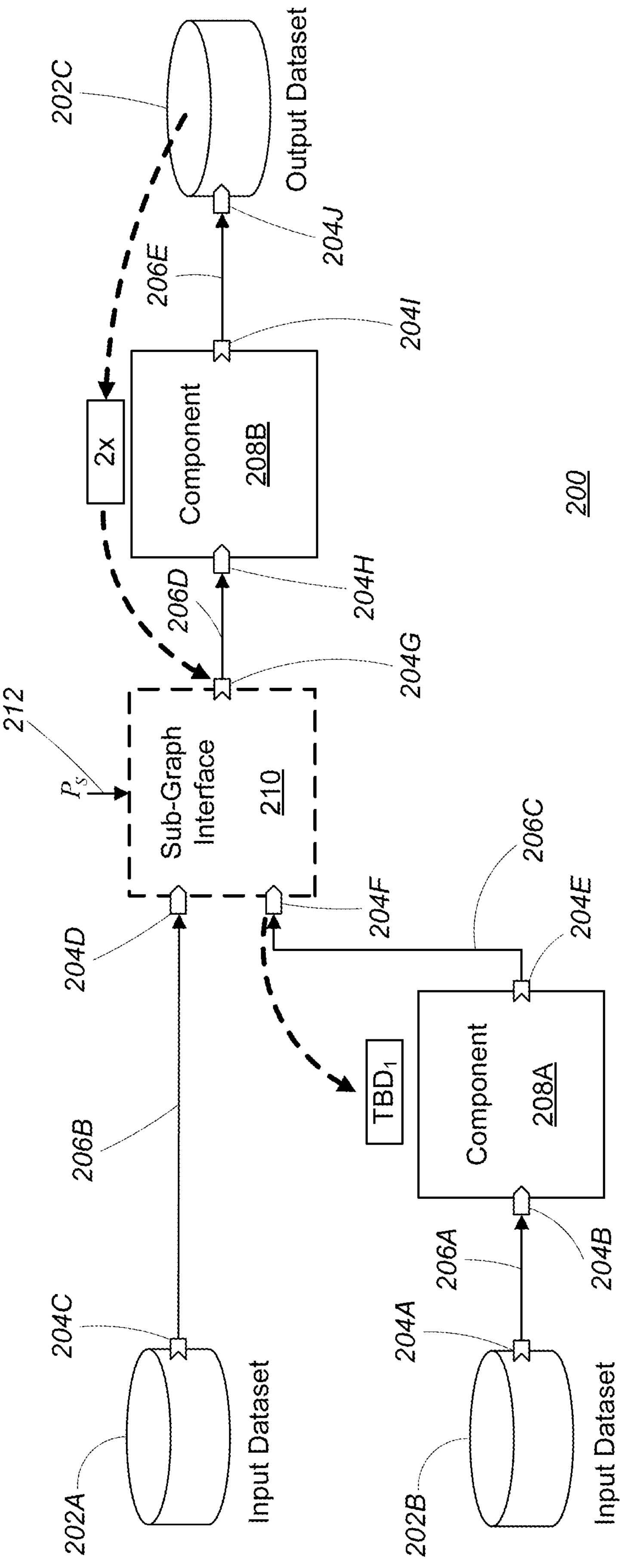
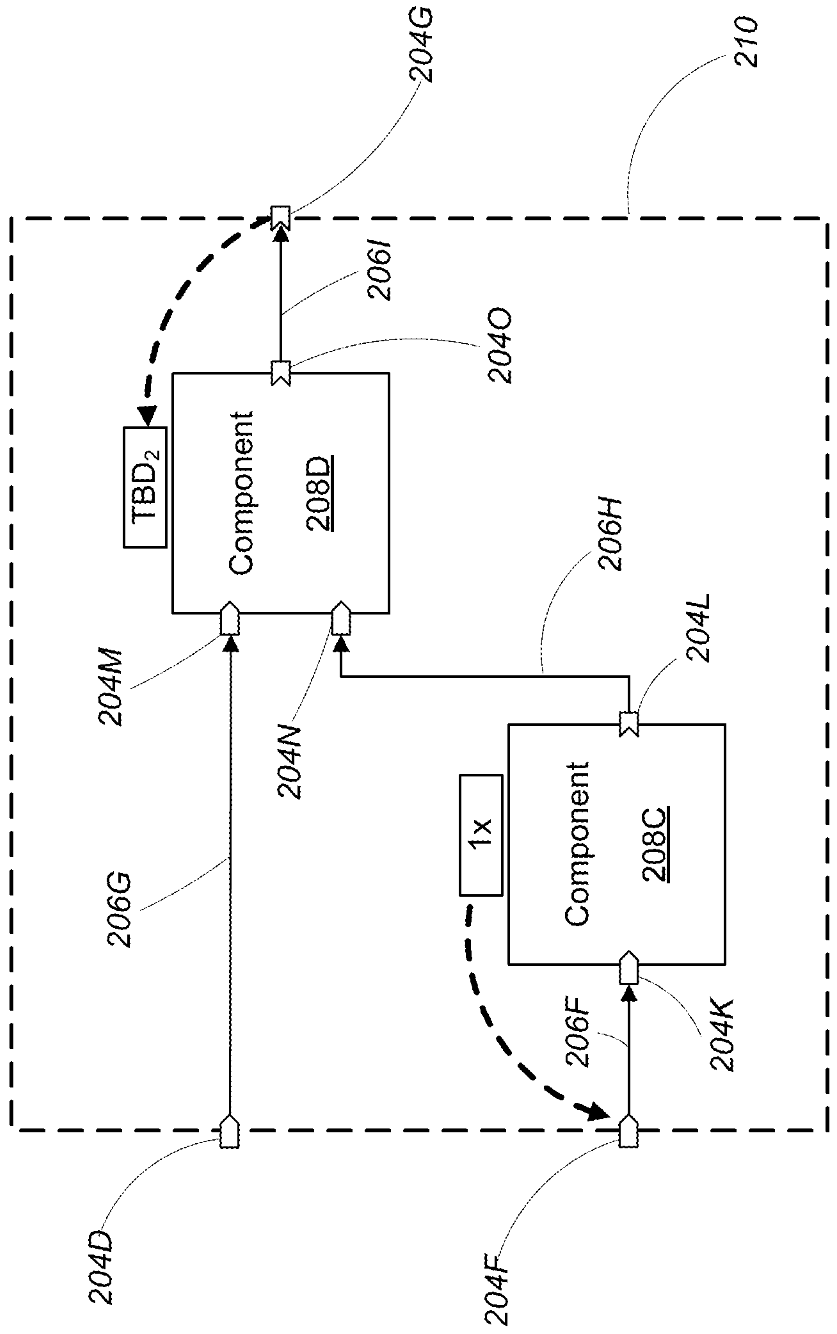

202C
Output Dataset
204J
206E
204I
2x
Component
208B
204H
206D
204G
200
212
P_S
Sub-Graph Interface
210
204D
204F
206C
204E
206B
TBD$_1$
Component
208A
204B
206A
204C
204A
202A
Input Dataset
202B
Input Dataset

FIG. 4A


204G
210
206I
204O
TBD_2
Component
208D
206H
204L
204M
204N
206G
1x
Component
208C
206F
204K
204D
204F

201

FIG. 4B

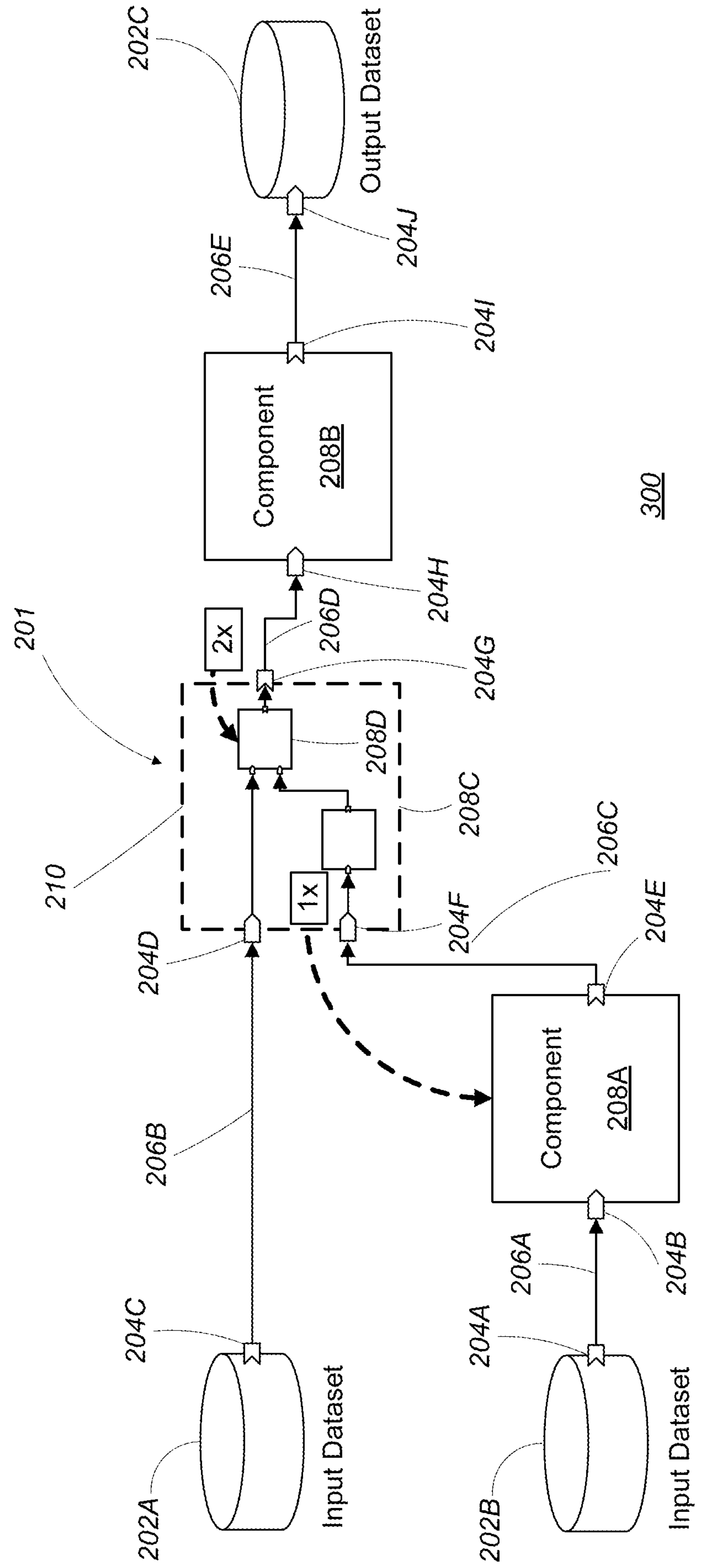

202A
Input Dataset
204C
202B
Input Dataset
204A
206A
204B
206B
Component
208A
204D
1x
204F
204E
206C
210
208C
201
208D
2x
204G
206D
204H
Component
208B
204I
206E
204J
Output Dataset
202C
300

FIG. 4C