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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 March 2007 (29.03.2007)

PCT

(10) International Publication Number
WO 2007/035491 A1

(51) International Patent Classification:
G06F 21/24 (2006.01) *G06F 17/50* (2006.01)
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/US2006/036047

(22) International Filing Date:
15 September 2006 (15.09.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/227,462 15 September 2005 (15.09.2005) US

(71) Applicant *(for all designated States except US)*: **THE MATHWORKS, INC.** [US/US]; 3 Apple Hill Drive, Natick, MA 01760 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **WENDLING, Bill** [US/US]; 1572 MASSACHUSETTS AVENUE, Apt. 4, Cambridge, MA 02138 (US).

(74) Agents: **CANNING, Kevin, J.** et al.; LAHIVE & COCKFIELD, LLP, 28 State Street, Boston, MA 02109 (US).

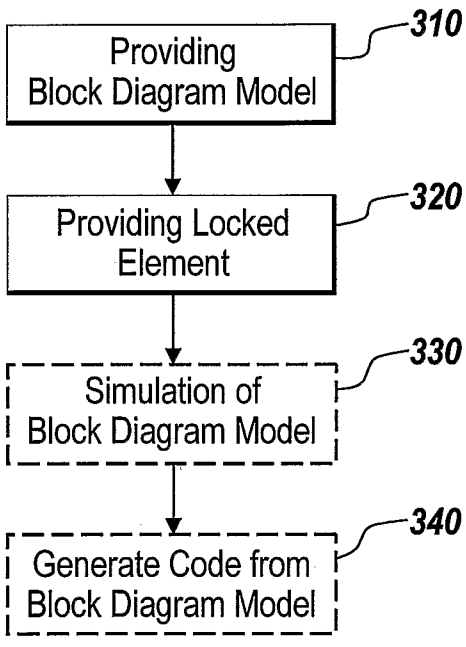
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, 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, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) **Title:** A LOCKED ELEMENT FOR USE IN A GRAPHICAL MODELING ENVIRONMENT



(57) **Abstract:** A locked element and methodology are provided for use in a block diagram model of a graphical modeling environment. The locked element displays within the block diagram model but access to the functionality of the locked element requires authorization. Without authorization the locked element will not function. Authorization may also be required to access the implementation details of the locked element. Without authorization the implementation details of the locked element cannot be viewed or modified. In certain implementations, any code generated from a locked element without authorization is obfuscated.

WO 2007/035491 A1

A LOCKED ELEMENT FOR USE IN A GRAPHICAL MODELING ENVIRONMENT

Related Application

This application claims the benefit of U.S. Patent Application Serial No. 11/227462, filed September 15, 2005, the contents of which are hereby incorporated by reference.

Field of the Invention

The present invention relates to a graphical modeling environment for modeling a static or dynamic system. More particularly, the present invention relates to providing a locked element that requires authorization to access the functionality of the locked element.

Background of the Invention

Many organizations are embracing the paradigm of Model Based Development in their production processes. "Model Based Development" refers to the practice of specifying, analyzing, and implementing systems using a common "model" consisting of a set of block diagrams and associated objects. System implementation typically consists of automatically generating code for portions of the model, particularly portions corresponding to the system's control algorithm or other functions implemented in software.

Graphical modeling environments are programs that enable a user to construct and analyze a model of a process or system. Examples of graphical modeling formalisms include time-based block diagrams, such as Simulink® from The MathWorks Inc., discrete event diagrams and reactive state machine diagrams, such as those found within Stateflow® also available from The MathWorks, Inc., data-flow diagrams, such as LabVIEW, available from National Instruments Corporation, and software diagrams and other graphical programming environments, such as Unified Modeling Language (UML) diagrams.

Some graphical modeling environments also enable simulation and analysis of models. Simulating a dynamic system in a graphical modeling environment is typically a two-step process. First, a user creates a graphical model, such as a block diagram, of the

system to be simulated. A graphical model may be created using a graphical user interface, such as a graphical model editor. The graphical model depicts relationships between the systems inputs, states, parameters and outputs. After creation of the graphical model, the behavior of the dynamic system is simulated using the information entered into the graphical model and information available in its environment such as files and variables in a general purpose workspace. In this step, the graphical model is used to compute and trace the temporal evolution of the outputs of the dynamic systems ("execute the graphical model"). Furthermore, it may automatically produce either deployable software systems or descriptions of hardware systems that mimic the behavior of either the entire model or portions of the model (code generation).

Block diagrams are graphical entities having an "executable meaning" that are created within graphical modeling environments for modeling static and dynamic systems, and generally comprise one or more graphical objects. For example, a block diagram model of a dynamic system is represented schematically as a collection of graphical objects, such as nodes, that are interconnected by edges, generally depicted as lines, which represent relations between the graphical objects that are connected to each of the edges.

In one subset of block diagramming paradigms, the nodes are referred to as "blocks" and drawn using some form of geometric object (e.g., circle, rectangle, etc.). The line segments are often referred to as "signals". Signals correspond to the time-varying quantities represented by each line connection and are assumed to have values at each time instant when connected to an enabled node. Each node may represent an elemental dynamic system, and the relationships between signals and state variables are defined by sets of equations represented by the nodes. Inherent in the definition of the relationship between the signals and the state variables is the notion of parameters, which are the coefficients of the equations. These equations define a relationship between the input signals, output signals, state, and time, so that each line represents the input and/or output of an associated elemental dynamic system. A line emanating at one node and terminating at another signifies that in terms of computational causality, the output of the first node is an input to the second node. Each distinct input or output on a node is referred to as a port. The source node of a signal writes to the signal at a given time instant when its system equations are solved. The destination node of this signal reads from the signal when their system equations are being solved. Those skilled in the art will recognize that the term "nodes" does not refer exclusively to

elemental dynamic systems but may also include other modeling elements that aid in readability and modularity of block diagrams.

It is worth noting that block diagrams are not exclusively used for representing time-based dynamic systems but also for other models of computation. For example, in Stateflow®, flow charts are block diagrams used to capture behavior of reactive systems and the flow of discrete state changes. Data flow models are block diagrams that describe a graphical programming paradigm where the availability of data is used to initiate the execution of blocks, where a block represents an operation and a line represents execution dependency describing the direction of data flowing between blocks.

In some instances, a user may wish to prevent others from using an element or block without authorization. As such a user may wish to lock access to the functionality of an element such that the locked element may be displayed in the graphical modeling environment but authorization is required in order to execute or simulate using the locked element.

In some instances a user may wish to limit the amount of functionality. In such instances authorization may be required to access the implementation details of a locked element. For example, a user may wish to share an element or a block diagram model with a third party. While the third party may need the element to function as part of a block diagram model, the user may not wish the third party to have access to the implementation details of the element. For example, the user may wish to provide a "black box" block element that can function as part of a block diagram model but does not provide the third party access to underlying functionality or implementation details of the element.

Summary of the Invention

One embodiment of the present invention provides a locked element that displays within a block diagram model but requires authorization to access the functionality of the element.

In accordance with a first aspect, in a graphical modeling environment, a method is provided. The method comprises providing a block diagram model; and providing a locked

element as part of the block diagram model; wherein the locked element displays within the block diagram model but access to the functionality of the locked element requires authorization. In some embodiments, authorization may be required to access the implementation details of the element.

In accordance with another aspect, in a graphical modeling environment, a method is provided for creating a locked element. The method comprises selecting an element to be locked; and locking the element to prevent access to the functionality of the element.

In accordance with another aspect, in a graphical modeling environment, a method is provided for creating a locked block diagram model. The method comprises selecting a block diagram model to be locked; and locking the block diagram model to prevent access to the functionality diagram model.

In accordance with another aspect, a medium is provided for use with an computational device holding instructions executable by the computational device for performing a method. The method comprises providing a block diagram model; and providing a locked element as part of the block diagram model; wherein the locked element displays within the block diagram model but access to the functionality of the locked element requires authorization.

In accordance with another aspect, a system is provided for generating and displaying a graphical modeling application. The system comprises user-operable input means for inputting data to the graphical modeling application; a display device for displaying a graphical model; and a computational device including memory for storing computer program instructions and data, and a processor for executing the stored computer program instructions, the computer program instructions including instructions for providing a block diagram model, and providing a locked element as part of the block diagram model; wherein the locked element displays within the block diagram model but access to the functionality of the locked element requires authorization.

In accordance with another aspect, a system is provided for generating and displaying a graphical modeling application. The system comprises a distribution server for providing to

a client device, locked element representing locked underlying functionality; and a client device in communication with the distribution server.

In accordance with another aspect, in a network having a server, executing a graphical modeling environment, and a client device in communication with the server, a method comprises the steps of providing, at the server, a block diagram model of a dynamic system having a locked element; receiving, at the server from the client device, a request to for access to the functionality of the locked element; and receiving, at the client device from the server, a request for authorization.

Brief Description of the Figures

Figure 1A illustrates an environment suitable for practicing an illustrative embodiment of the present invention.

Figure 1B is a flow chart illustrating the steps involved in simulating and generating code from a dynamic system using the environment shown in Figure 1A.

Figure 2A is an example of a block diagram of a dynamic system in a graphical modeling environment.

Figure 2B is an example of a Stateflow® state chart in a graphical modeling environment.

Figure 2C is an example of MATLAB® embedded block for use in a graphical modeling environment.

Figure 3 is a flow chart illustrating the steps performed in an exemplary embodiment of the method of the present invention.

Figure 4 is an example of a block diagram having a locked block.

Figure 5 is an example of an output generated when an Application Programming Interface (API) is used on a locked block without authorization.

Figure 6 is an example of obfuscated code generated from a locked block.

Figure 7 is an example of a graphical interface requesting authorization for access to the locked block.

Figure 8 is an example of an underlying subsystem of an unlocked block.

Figure 9 is an example of an output generated when an Application Programming Interface (API) is used on an unlocked block

Figure 10 is an example of a Graphical User Interface (GUI) that can be used to modify the parameters of an unlocked block.

Figure 11 is an example of non-obfuscated code generated from an unlocked block.

Figure 12 is an exemplary flow diagram of a method of creating a locked block.

Figure 13A is an example of a graphical interface used to create a locked block.

Figure 13B is an example of graphical interface used for specifying the authorization for a locked block.

Figure 14 illustrates an exemplary client-server environment suitable for practicing an illustrative embodiment of the present invention

Detailed Description of an Illustrative Embodiment

An illustrative embodiment of the present invention relates to a locked element that displays within a block diagram model but requires authorization to access the functionality of the element. The present invention will be described relative to illustrative embodiments. Those skilled in the art will appreciate that the present invention may be implemented in a

number of different applications and embodiments and is not specifically limited in its application to the particular embodiments depicted herein.

Figure 1A depicts an environment suitable for practicing an illustrative embodiment of the present invention. An electronic device 2 includes memory 4, on which software according to one embodiment of the present invention is stored, a processor (CPU) 7 for executing software stored in the memory, and other programs for controlling system hardware. Typically, the interaction of a human user 10 with the electronic device 2 occurs through an input/output (I/O) device 8, such as a user interface. The I/O device 8 may include a display device 8a (such as a monitor) and an input device (such as a mouse 8b and a keyboard 8c and other suitable conventional I/O peripherals).

For example, the memory 4 holds a diagramming application 6 capable of creating and simulating computational versions of system diagrams, such as time-based block diagrams, state diagrams, signal diagrams, flow chart diagrams, sequence diagrams, UML diagrams, dataflow diagrams, circuit diagrams, ladder logic diagrams, kinematic element diagrams, or other models, which may be displayed to a user 10 via the display device 8a. In the illustrative embodiment, the diagramming application 6 comprises a block diagram environment, such as Simulink® or another suitable graphical modeling environment. As used herein, the terms "block diagram environment" and "graphical modeling environment" refer to a graphical application where a system is graphically modeled. Examples of suitable diagramming applications include, but are not limited to, MATLAB® with Simulink®, from The MathWorks, Inc., LabVIEW, DasyLab and DiaDem from National Instruments Corporation, VEE from Agilent, SoftWIRE from Measurement Computing, VisSim from Visual Solutions, SystemVIEW from Elanix, WiT from Coreco, Vision Program Manager from PPT Vision, Khoros from Khoral Research, Halcon from MVTec Software, and numerous others. The memory 4 may comprise any suitable installation medium, e.g., a CD-ROM, DVD, floppy disks, or tape device; a computer system memory or random access memory such as DRAM, SRAM, EDO RAM, Rambus RAM, etc.; or a non-volatile memory such as a magnetic media, e.g., a hard drive, or optical storage. The memory may comprise other types of memory as well, or combinations thereof.

In an alternative embodiment, the electronic device 2 is also interfaced with a network, such as the Internet. Those skilled in the art will recognize that the diagrams used

by the diagramming application 6 may be stored either locally on the electronic device 2 or at a remote location 9 interfaced with the electronic device over a network. Similarly, the diagramming application 6 may be stored on a networked server or a remote peer.

The diagramming application 6 of an illustrative embodiment of the invention includes a number of generic components. Although the discussion contained herein focuses on Simulink®, from The MathWorks, Inc. of, Natick MA, those skilled in the art will recognize that the invention is applicable to other software applications. The generic components of the illustrative diagramming program 6 include a block diagram editor 6a for graphically specifying models of dynamic systems. The block diagram editor 6a allows users to perform such actions as construct, edit, display, annotate, save, and print out a graphical model, such as a block diagram, that visually and pictorially represents a dynamic system. The illustrative diagramming application 6 also includes graphical entities 6b, such as signal lines and buses that represent how data is communicated between functional and non-functional units, and blocks 6c. As noted above, blocks are the fundamental mathematical elements of a classic block diagram model. A block diagram execution engine 6d, also implemented in the application, is used to process a graphical model to produce simulation results or to convert the graphical model to executable code. For a block diagram graphical model, the execution engine 6d translates a block diagram to executable entities following the layout of the block diagram as provided by the user. The executable entities are compiled and executed on a computational device, such as a computer, to implement the functionality specified by the model. Typically, the code generation preserves a model hierarchy in a call graph of the generated code. For instance, each subsystem of a model in a block diagram environment can map to a user specified function in the generated code. Real-Time Workshop from The MathWorks, Inc. of Natick, Massachusetts is an example of a suitable execution engine 6d for generating code.

In the illustrative embodiment, the diagramming program 6 is implemented as a companion program to a technical computing program, such as MATLAB®, also available from The MathWorks, Inc.

Figure 1B is a flow chart diagramming the steps involved in simulating and generating code from a dynamic system according to an illustrative embodiment of the invention. In step 12, a user creates a block diagram model representing a dynamic system.

Once a block diagram model, or other graphical model, has been constructed using the editor 6a in step 12, the execution engine 6d simulates the model by solving equations defined by the model to trace the system outputs as a function of time, in steps 14-18. The solution of the model, which may be referred to as model execution, is carried out over a user-specified time span for a set of user-specified inputs. After creating the block diagram model in step 12, the execution engine 6d compiles the block diagram in step 14. Then, in step 16, the execution engine links the block diagram in to produce an "in-memory executable" version of the model. Then, the execution engine uses the "in-memory executable" version of the model to generate code and/or simulate a block diagram model by executing the model in step 18 or 20.

The block diagram editor 6a is the user interface component that allows a user to create and modify a block diagram model representing a dynamic system, in step 12. The blocks in the electronic block diagram may model the behavior of specialized mechanical, circuit or software components, such as motors, servo-valves, power plants, blocks, tires, modems, receivers, and other dynamic components. The block diagram editor 6a also allows a user to create and store data relating to graphical entities 6b. In Simulink®, a textual interface with a set of commands allows interaction with the graphical editor. Using this textual interface, users may write special scripts that perform automatic editing operations on the block diagram. A user generally interacts with a set of windows that act as canvases for the model. There is generally more than one window for a model because models may be partitioned into multiple hierarchical levels through the use of subsystems.

A suite of user interface tools within the block diagram editor 6a allows users to draft a block diagram model on the corresponding windows. For example, in Simulink® the user interface tools include a block palette, a wiring line connection tool, an annotation tool, a formatting tool, an attribute editing tool, a save/load tool and a publishing tool. The block palette is a library of all the pre-defined blocks available to the user for building the block diagram. Individual users may be able to customize this palette to: (a) reorganize blocks in some custom format, (b) delete blocks they do not use, and (c) add custom blocks they have designed. The palette allows blocks to be dragged through some human-machine interface (such as a mouse or keyboard) from the palette onto the window (i.e., model canvas). The graphical version of the block that is rendered on the canvas is called the icon for the block.

There may be different embodiments for the block palette including a tree-based browser view of all of the blocks.

A block diagram model of a dynamic system, created during step 12, is generally represented schematically as a collection of interconnected graphical objects, such as blocks, ports and lines, which represent signals. Figure 2A illustrates an example of a block diagram 200 created using the diagramming application 6. Each block in the block diagram 200 represents an elemental dynamic system. Each signal, denoted by lines connecting the blocks, represents the input and/or output of an elemental dynamic system. The illustrative block diagram 200 includes a subsystem block 210, a source block 220 and a destination block 230. A line emanating at one block and terminating at another signifies that the output of the first block is an input to the second block. Ports, such as input port 212 and output port 214 of the subsystem block 210, refer to a distinct inputs or outputs on a block. Signals correspond to the time-varying quantities represented by each line connection and are assumed to have values at each time instant when their connected blocks are enabled. The source block 220 for a signal 221 writes to the signal at a given time instant when its system equations are solved. As shown, the signal 221 from the source block passes to the subsystem 210. The signal 211 output from the subsystem 210 passes to the destination block 230. The destination block 230 for a signal 211 reads from the signal 211 when the system equation is being solved. As shown, the signal 211 represents the output of the subsystem 210. One skilled in the art will recognize that the block diagram 200 is merely illustrative of a typical application and is not intended to limit the present invention in any way.

Figure 2B illustrates an example of a Stateflow® diagram 240 containing Stateflow® blocks 250 and 260 created using the diagramming application 6. Each block in the block diagram 240 represents a state. The lines between the blocks represent state transitions. Ports, such as input port 252 and output port 254 of the Stateflow® block 250, refer to distinct inputs or outputs on a block. One skilled in the art will recognize that the block diagram 240 is merely illustrative of a typical application and is not intended to limit the present invention in any way.

Figure 2C illustrates an example of an Embedded MATLAB block 270 created using the diagramming application 6. This block uses MATLAB code to describe its functionality. It may be used as any other type of block as part of a block diagram model.

Once a block diagram model, or other graphical model, has been constructed using the editor 6a in step 12, the execution engine 6d simulates the model by solving equations defined by the model to trace the system outputs as a function of time, in steps 14-18. The solution of the model, which may be referred to as model execution, is carried out over a user-specified time span for a set of user-specified inputs.

The compile stage in step 14 marks the start of model execution and involves preparing data structures and evaluating parameters, configuring and propagating block characteristics, determining block connectivity, and performing block reduction and block insertion. The compile stage involves checking the integrity and validity of the block interconnections in the block diagram. In this stage, the engine 6d also sorts the blocks in the block diagram into hierarchical lists that are used when creating the block method execution lists. The preparation of data structures and the evaluation of parameters create and initialize basic data-structures needed in the compile stage. For each of the blocks, a method forces the block to evaluate all of its parameters. This method is called for all blocks in the block diagram. If there are any unresolved parameters, execution errors are thrown at this point.

The compilation step also determines actual block connectivity. Virtual blocks play no semantic role in the execution of a block diagram. During compilation, the virtual blocks and signals, such as virtual bus signals, in the block diagram are optimized away (removed) and the remaining non-virtual blocks are reconnected to each other appropriately. This compiled version of the block diagram with actual block connections is used from this point forward in the execution process.

In the link stage, in step 16, the execution engine 6d uses the result of the compilation stage to allocate memory needed for the execution of the various components of the block diagram. The linking stage also produces block method execution lists, which are used by the simulation, linearization, or trimming, of the block diagram. Included within the link stage is the initialization of the model, which consists of evaluating "setup" methods (e.g. block start, initialize, enable, and constant output methods). The block method execution

lists are generated because the simulation and/or linearization of a model must execute block methods by type (not by block) when they have a sample hit.

The compiled and linked version of the block diagram may be directly utilized to execute the model over the desired time-span, in step 18. In step 20, the execution engine may choose to translate the block diagram model (or portions of it) into either software modules or hardware descriptions (broadly termed "code"). The code may be instructions in a high-level software language such as C, C++, Ada, etc., hardware descriptions of the block diagram portions in a language such as HDL, or custom code formats suitable for interpretation in some third-party software. Alternatively, the code may be instructions suitable for a hardware platform such as a microprocessor, microcontroller, or digital signal processor, etc., a platform independent assembly that can be re-targeted to other environments, or just-in-time code (instructions) that corresponds to sections of the block diagram for accelerated performance.

Figure 3 depicts a flowchart 300 of one exemplary embodiment of a method of the present invention. Here the method involves providing a block diagram model 310 and providing a locked element as part of the block diagram model 320. The locked element displays within the block diagram model but access to the functionality of the locked element requires authorization. In some embodiments, simulation of the block diagram may then be performed once access has been granted 330.

In certain embodiments code may be generated from the block diagram model 340. In some such embodiments the code generated from the locked block without authorization will be obfuscated. Code obfuscation in a graphical modeling environment is set forth in U.S. Patent Application No. 11/038,608 entitled "Obfuscation of Automatically Generated Code", filed on 01/18/2005, the specification of which is hereby incorporated by reference. With authorization the code may be generated normally.

An example of a block diagram model 400 as provided in the method of Figure 3 (step 310) can be seen in Figure 4. This exemplary block diagram model 400 includes an input node 410, an output node 420, and the locked element, here a locked block 430 provided in the method of Figure 3 (step 320). The input node 410 provides the locked block 430 with an input signal 415. Until authorization is provided the lock block 430 will not

operate in the model. Once access is granted, the block 430 may then perform some process on the received input and provides an output signal 425 to the output port 420.

In another embodiment, authorization may be required to access the implementation details of the locked block 430. Thus a block may be unlocked so as to function within the diagram model 400 system but access to the implementation details of the block are still locked.

When used in a block diagram model, the locked block may function as a "black box." Like any other element, the locked block represents some underlying functionality. Based on this underlying functionality, the locked block can receive an input signal 415 and provide an output signal 425. The locked block differs from other element in that access to its functionality requires authorization.

Without authorization, the functionality of a locked element such as locked block 430 may not be accessed. Likewise, application program interfaces (API's) such the "get_param" API of MATLAB will not function without authorization. Figure 5 depicts an example of an interface window 500 showing what happens when the "get_param" API is on the locked block 430 of Figure 4. Furthermore, any code generated from the locked block 430 without authorization may be obfuscated to prevent attempts to determine the implementation details of the block from the code. An example of such obfuscated code 600 generated from the locked block 430 can be seen if Figure 6.

In another embodiment, authorization may be required to access the implementation details of a locked element or block. For example, the basic functionality of the element may be unlocked so that the element operates in a block diagram model, but to view or edit the implementation details that provide the functionality, authorization is required. Using such an implementation, a locked element may be provided to a third party user without having to provide the implementation details of the locked element which may be proprietary. The third party user may use the locked element in a block diagram model, perform simulations, generate code, and even execute the generated code, but cannot access the implementation details of the locked element without authorization.

One method of determining if an element is locked is to check how the data for the element is stored on a physical medium. If the data for the element is stored on a restricted

access device that requires a user to login before accessing it, then the element could be designated as locked. Another method, for when the data for the element is on a non-restricted device, is to provide a series of bytes at the beginning of the data file for the element indicating that the element is an encoded element that a graphical environment (such as Simulink) has access to. A password or phrase internal to the graphical environment could then be required to access the functionality of the element.

In one exemplary embodiment, authorization for the locked element is provided by a password or authorization code. Thus in the example wherein a third party user is working with the block diagram model of Figure 4, when the third party user attempts to access the functionality of the locked block 430, the third party is prompted to provide a password to gain access to the functionality. An example of window 700 providing such a prompt can be seen in Figure 7. Here the window 700 prompts the third party user for a password 710 and provides a field for the user to enter the password 720. In this example the third party user may then choose to submit a password by clicking the "OK" button 730 or cancel the authorization using the "Cancel" button 740. If the third party does not provide the correct password or chooses to cancel the authorization, the user may still continue to view the locked block 430 without access to its functionality. Although, in this embodiment access to the functionality of a locked element is obtained using a graphical user interface, it should be understood that access may be obtained in other ways. For example, a textual interface or Application Program interface (API) may be used to gain access to the functionality of a locked element.

It should also be understood that authorization is not limited to the use of passwords. Other authorization methods may also be used. For example, in some embodiments, a hardware dongle or key physically attached to the third party user's system may be required. Such a dongle or key could be attached to the serial, parallel, USB, IEEE 1394 (Firewire, iLink) port, or the like of the system and be checked for before access to the functionality is provided. In other embodiments, biometrics signatures such as fingerprints may be required for authorization. Other possible authorization techniques will be apparent to one skilled in the art given the benefit of this disclosure.

Returning now to the example wherein the third party user has been prompted for a password, if the correct password is provided, the third party user is then authorized to access

the functionality of the now unlocked block. In this example, the previously locked block 430 of Figure 4 represents a subsystem shown in Figure 8 which is now accessible to the third party user.

The subsystem 800 of Figure 8 is a block diagram model that determines the average for an array. An array is received at the input 810 and is passed to a block 820 that adds the elements of the array and a block 830 that determines the number of elements in the array. The outputs of these blocks are then passed to a block 840 that divides the sum of the elements by the number of elements of the array. The resulting average is then passed to the output 850.

Although in the present example the locked element represented a block diagram model subsystem, it should be understood that the implementation details of a locked element are not limited to a block diagram model subsystem or referenced model. In some embodiments, the locked element may represent a Stateflow model such as seen in Figure 2B. In other embodiments the locked element may represent an Embedded MATLAB® system wherein the functionality of the block is described using MATLAB® code as shown in Figure 2C. These and any number of other implementations of element may be locked under the present invention. In certain embodiments it may be advantageous to combine such different implementations together providing a nesting of systems. For example a locked element may represent a block diagram model subsystem comprising locked element representing both additional block diagram models and embedded systems wherein the functionality is described in code. In such a system each locked element may require is own separate authorization to gain access to its implementation details or access could be obtained globally.

Once an element is unlocked its functionality becomes accessible. The implementation details may also become accessible. For example, if the implementation details become available, an API, such as "get_param" may now operate on the element, such as block 430 of Figure 4, as shown in the example window 900 of Figure 9. In some embodiments access is limited to the ability to view the implementation details. In other embodiments access includes the ability to edit or otherwise modify the implementation details of the element. In such embodiments a graphical user interface (GUI) or programmatic interface may be used to interact with the implementation details of the

element. For example, if the element represents a block diagram model subsystem, blocks or other elements may be added or removed from the system. If, for example, the functionality of the element is defined by code, the code may be edited to modify the functionality of the element. Likewise, any number of parameters for an element, including but not limited to, input size, output size, input type, output type, delay, gain, and ranges, can be modified.

An example of a graphical user interface 1000 for editing the parameters of an element such as a block can be seen in Figure 10. In this example the interface 1000 provides instructions for adding or subtracting inputs 1010 as well as the ability to specify data signal types 1020, change the shape of the block icon 1030, and set sample time for the block 1040.

In certain embodiments, wherein code generated from a locked element is obfuscated, unlocking the element allows the code generated from the element to be non-obfuscated. An example of non-obfuscated code 1100 can be seen in Figure 11. In this example the non-obfuscated code 1100 is the code generated from the now unlocked block 430 of Figure 4.

In certain embodiments, it may be beneficial to have different levels of authorization for a locked element. For example, the locked element may have a first authorization for enabling the execution or simulation of the element, a second authorization for viewing the implementation details, a third authorization for editing or modifying the implementation details of the element, and a forth authorization for generating non-obfuscated code. Likewise, different types of authorizations can be implemented for different levels. In embodiments having multiple or nested locked element, different levels and types of authorization can be used for each element. In other embodiments, a single authorization may give access to the implementation details of a number of locked elements. Other variations, combinations, and implementations will be apparent to one skilled in the art given the benefit of this disclosure.

Figure 12 depicts a flowchart 1200 of one exemplary embodiment of a method of the present invention for creating a locked element. Here the method involves selecting an element such as a block or block diagram model to be locked 1210 and locking the block or block diagram model 1220. In embodiments wherein a block diagram model is being locked, the method may further include creating a block representing the locked block diagram model 1230.

In some embodiments the selection and locking of an element such as a block or block diagram model may be accomplished using a graphical user interface (GUI). For example, a user may select the block or block diagram model using an interface such as a mouse and then use a menu to lock the block or block diagram model. An example of this can be seen in Figure 13A. Here a block diagram model 1300 is selected and then locked using a pull down menu 1310. In another embodiment, an element or block diagram model may be locked using an Application Program Interface (API). Other possible variations and implementations will be apparent to one skilled in the art given the benefit of this disclosure.

As part of the locking process the user may indicate what authorization is to be required to unlock the locked element. For example, after the user has chosen to lock a selected block or block diagram model, the user may be prompted to specify the desired authorization to unlock the block. In some embodiments this may be done using a GUI, for example the user may be provided with a window allowing the user to select the desired authorization. An example of such a window 1320 can be seen in Figure 13B. Here, the window 1320 prompts the user to specify a password that will unlock the locked block 1330 and provides a field for the user to enter the desired password 1340.

One possible locking technique is encryption. Here, to lock an element such as a block or block diagram the user specifies that the element is to be written to a persistent file, such as an MDL file in an encrypted format. Once written in this format, the implementation details of the element can only be read either by the environment that created the element or an environment that has the authorization to unlock the element. Suitable encryption methods include, but are not limited to, PGP, DES3, Blowfish, or the like.

It should be understood that the methodology for locking an element is not limited to encryption. Other methodologies may be employed depending on the level of protection required. In some embodiments different methodologies and levels of protection may be used together to lock different elements, block diagram models, and levels of authorization.

The examples to this point have focused primarily on the system where the graphical modeling environment was on a local computational device, in one embodiment an electronic device. The graphical modeling environment may of course also be implemented on a

network 1400, as illustrated in Figure 14, having a server 1410 and a client device 1420. Other devices, such as a storage device 1430, may also be connected to the network.

In one such embodiment a system for generating and displaying a graphical modeling application, comprises a distribution server for providing to a client device, a locked element requiring authorization to access the implementation details of the element; and a client device in communication with the distribution server. Here the distribution server provides a client device, such as an electronic device discussed above, with a locked element. The client may then use the locked element in a block diagram model but would require authorization to access the functionality of the locked element. Without authorization, the client cannot run a simulation of the block diagram model featuring the locked element.

In another embodiment, a user may interact with a graphical modeling interface on the server through the client device. In one example of such a system a server and client device are provided. The server is capable of executing a graphical modeling environment. The client device is in communication with the server over a network. A block diagram model of a dynamic system having a locked element is provided at the server. A request for access to the functionality of the locked element is received at the server from the client. In response to request for access from the server, the server requests authorization from the client. If the client provides the server with the proper authorization then the server grants the client access to the functionality of the locked element.

It will be understood by one skilled in the art that these network embodiments are exemplary and that the functionality may be divided up in any number of ways over a network.

The present invention has been described relative to illustrative embodiments. Since certain changes may be made in the above constructions without departing from the scope of the invention, it is intended that all matter contained in the above description or shown in the accompanying drawings be interpreted as illustrative and not in a limiting sense.

It is also to be understood that the following claims are to cover all generic and specific features of the invention described herein, and all statements of the scope of the invention which, as a matter of language, might be said to fall therebetween.

What is claimed is:

1. In a graphical modeling environment, a method comprising:

providing a block diagram model; and

providing a locked element as part of the block diagram model,

wherein the locked element displays within the block diagram model but access to the functionality of the locked element requires authorization.

2. The method of claim 1 further comprising requiring authorization to access the implementation details of the locked element.

3. The method of claim 1, further comprising generating code from the block diagram model.

4. The method of claim 3, wherein code generated from the locked element without the authorization is obfuscated.

5. The method of claim 1 wherein the locked element comprises a block.

6. The method of claim 1 wherein the locked element represents a subsystem.

7. The method of claim 6 wherein authorization grants access to the underlying subsystem that the locked element represents.

8. The method of claim 7 wherein authorization allows the underlying subsystem to be modified.

9. The method of claim 1 wherein the locked element represents another model.

10. The method of claim 9 wherein authorization grants access to the underlying model that the locked element represents.

11. The method of claim 10 wherein authorization allows the underlying model to be modified.

12. The method of claim 1 wherein, authorization allows the use of an application program interface (API) on the element.

13. The method of claim 1 wherein, authorization allows interaction with the element in programmatic interfaces.

14. The method of claim 1 wherein, authorization allows the implementation details of the element to be modified.

15. The method of claim 1 wherein the locked element is locked using an encryption method.

16. The method of claim 15 wherein the encryption method comprises PGP.

17. The method of claim 15 wherein the encryption method comprises DES3.

18. The method of claim 1, further comprising execution of the block diagram model having a locked element.

19. In a graphical modeling environment, a method for creating a locked element, the method comprising:

selecting an element to be locked; and

locking the element to prevent access to the functionality of the element.

20. The method of claim 19, wherein locking the element comprises selecting an authorization code to allow access to the implementation details of the element.

21. The method of claim 19, wherein locking the element comprises encrypting the element.

22. The method of claim 19, wherein locking the element is performed by an application program interface (API).

23. In a graphical modeling environment, a method of creating a locked block diagram model, the method comprising:

selecting a block diagram model to be locked; and

locking the block diagram model to prevent access to the functionality of the block diagram model.

24. The method of claim 23, further comprising creating a block representing the locked block diagram model.

25. The method of claim 23, wherein locking the block diagram model comprises selecting an authorization code to allow access to the functionality of the block diagram model.

26. The method of claim 23, wherein locking the block diagram model comprises encrypting the block diagram model.

27. A medium for use with a computational device holding instructions executable by the computational device for performing a method, comprising the steps of:

providing a block diagram model; and

providing a locked element as part of the block diagram model;

wherein the locked element functions within the block diagram model but access to the functionality of the locked element requires authorization.

28. The medium of claim 27 wherein the method further comprises requiring authorization to access the implementation details of the locked element.

29. The medium of claim 27, wherein the method further comprises generating code from the block diagram model.

30. The medium of claim 28, wherein code generated from the locked element without the authorization is obfuscated.

31. The medium of claim 27 wherein authorization allows the use of an application program interface (API) on the element.

32. A system for generating and displaying a graphical modeling application, comprising:

user-operable input means for inputting data to the graphical modeling application;

a display device for displaying a graphical model; and

a computational device including memory for storing computer program instructions and data, and a processor for executing the stored computer program instructions, the computer program instructions including instructions for

providing a block diagram model; and

providing a locked element as part of the block diagram model;

wherein the locked element displays within the block diagram model but access to the functionality of the locked element requires authorization.

33. The system of claim 32 wherein the instructions further comprise requiring authorization to access the implementation details of the locked element.

34. The system of claim 32, wherein the instructions further comprise generating code from the block diagram model.

35. The medium of claim 34, wherein code generated from the locked element without the authorization is obfuscated.

36. The system of claim 32 wherein authorization allows the use of an application program interface (API) on the element.

37. A system for generating and displaying a graphical modeling application, comprising:

a distribution server for providing to a client device, a locked element requiring authorization to access the functionality of the locked element; and

a client device in communication with the distribution server.

38. The system of claim 37 wherein the locked element further requires authorization to access the implementation details of the locked element.

39. In a network having a server, executing a graphical modeling environment, and a client device in communication with the server, a method comprising the steps of:

providing, at the server, a block diagram model of a dynamic system having a locked element;

receiving, at the server from the client device, a request for access to the functionality of the locked element; and

receiving, at the client device from the server, a request for authorization

40. The method of claim 39 further comprising the steps of:

receiving, at the server from the client, authorization; and

receiving, at the client from the server, access to the functionality of the locked element.

41. The system of claim 40 further comprising receiving, at the client from the server, access the implementation details of the locked element.

42. The system of claim 40 further comprising generating, at the server, code from the block diagram model.

43. The system of claim 40 wherein authorization allows the use of an application program interface (API) on the element.

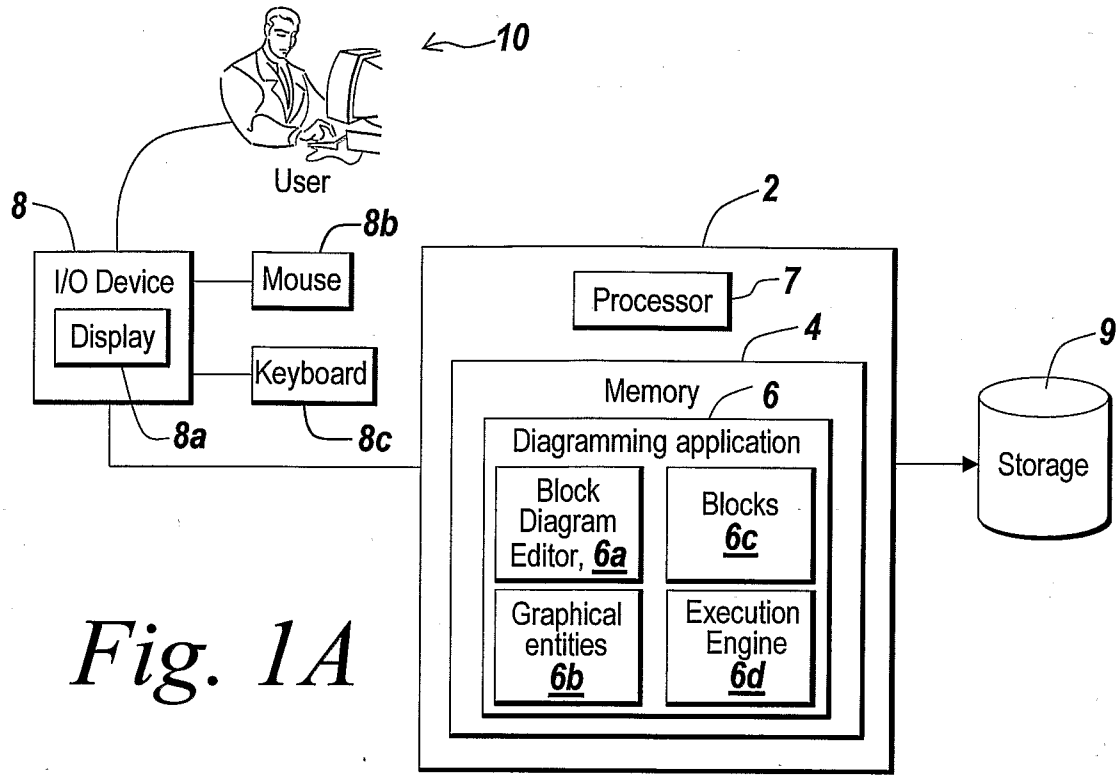

10
User
8
8b
2
I/O Device
Display
Mouse
Keyboard
8a
8c
Processor
7
4
Memory
6
Diagramming application
Block Diagram Editor, 6a
Blocks 6c
Graphical entities 6b
Execution Engine 6d
9
Storage

Fig. 1A

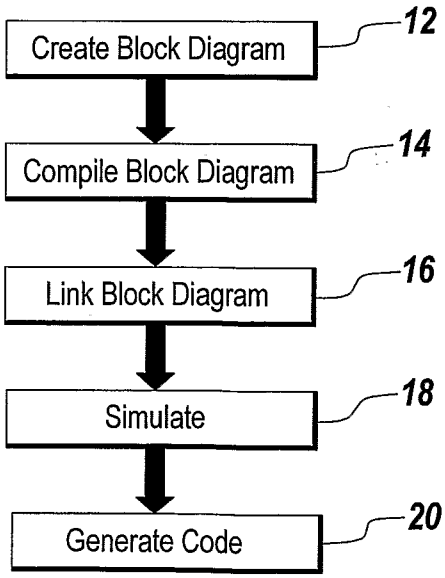

Create Block Diagram
12
Compile Block Diagram
14
Link Block Diagram
16
Simulate
18
Generate Code
20

Fig. 1B

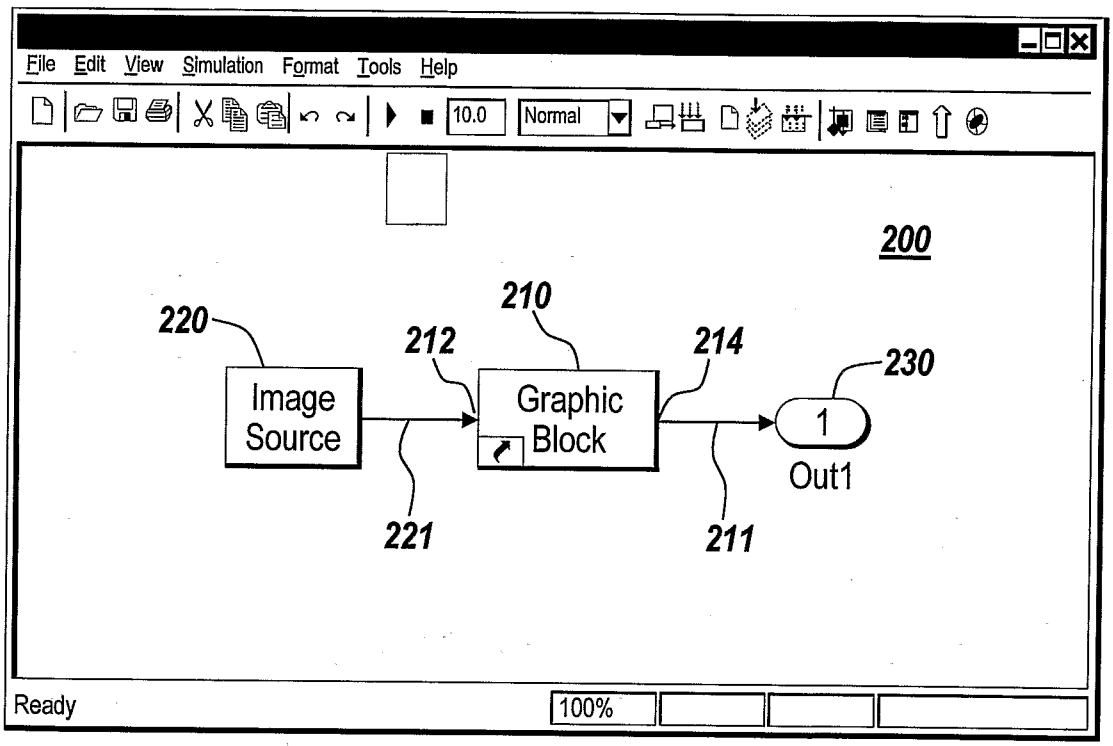

File Edit View Simulation Format Tools Help
10.0
Normal
200
220
210
212
214
230
Image Source
Graphic Block
1
Out1
221
211
Ready
100%

Fig. 2A

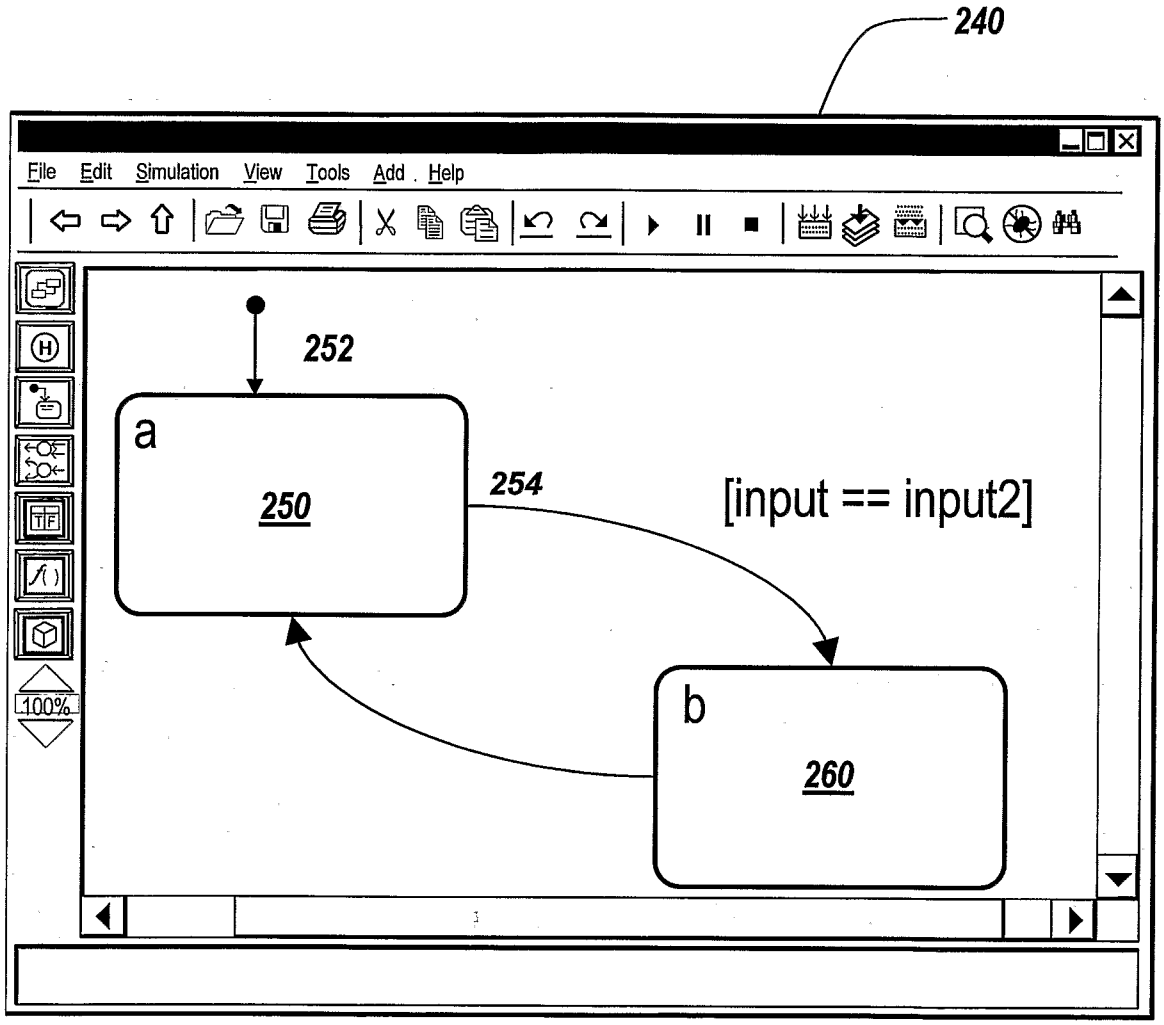

240
File Edit Simulation View Tools Add Help
252
a
250
254
[input == input2]
b
260
100%

Fig. 2B

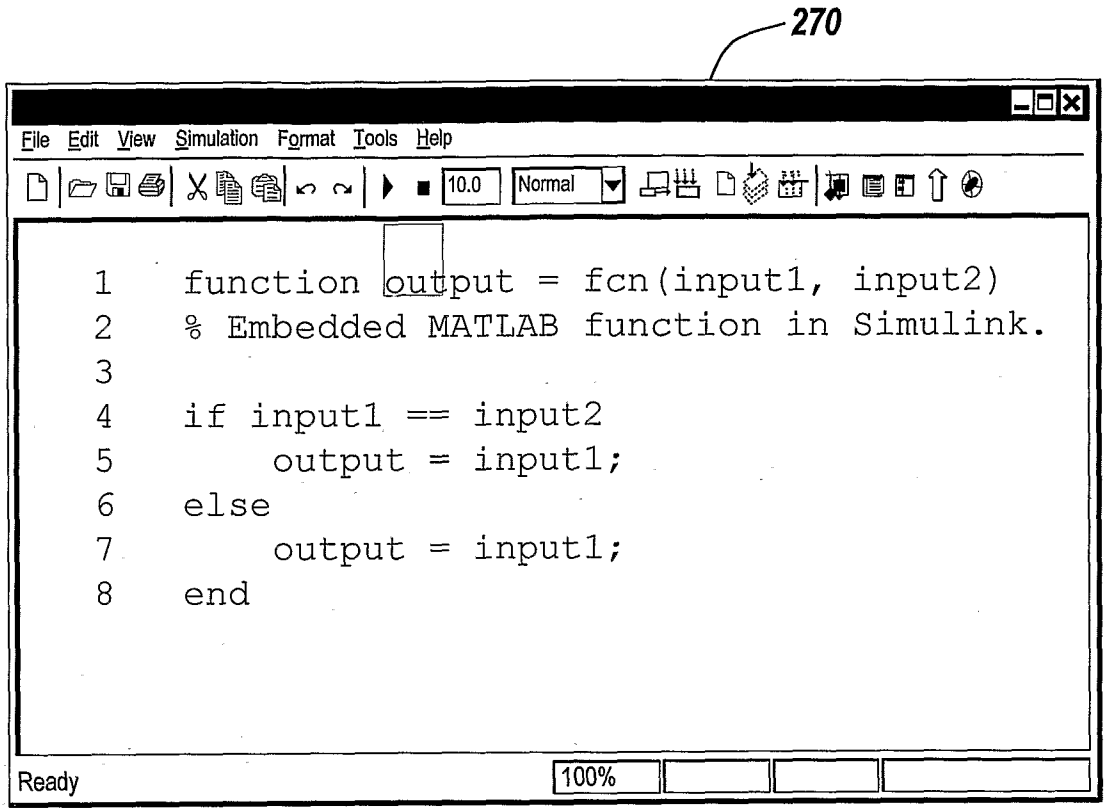

270
File Edit View Simulation Format Tools Help
10.0
Normal
1 function output = fcn(input1, input2)
2 % Embedded MATLAB function in Simulink.
3
4 if input1 == input2
5 output = input1;
6 else
7 output = input1;
8 end
Ready
100%

Fig. 2C

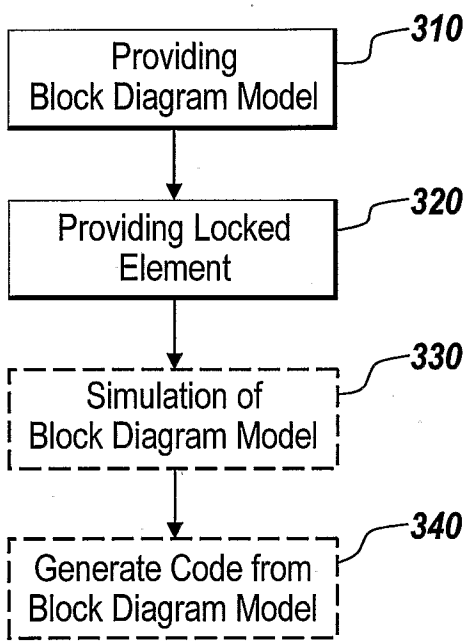

Providing Block Diagram Model
310
Providing Locked Element
320
Simulation of Block Diagram Model
330
Generate Code from Block Diagram Model
340

Fig. 3

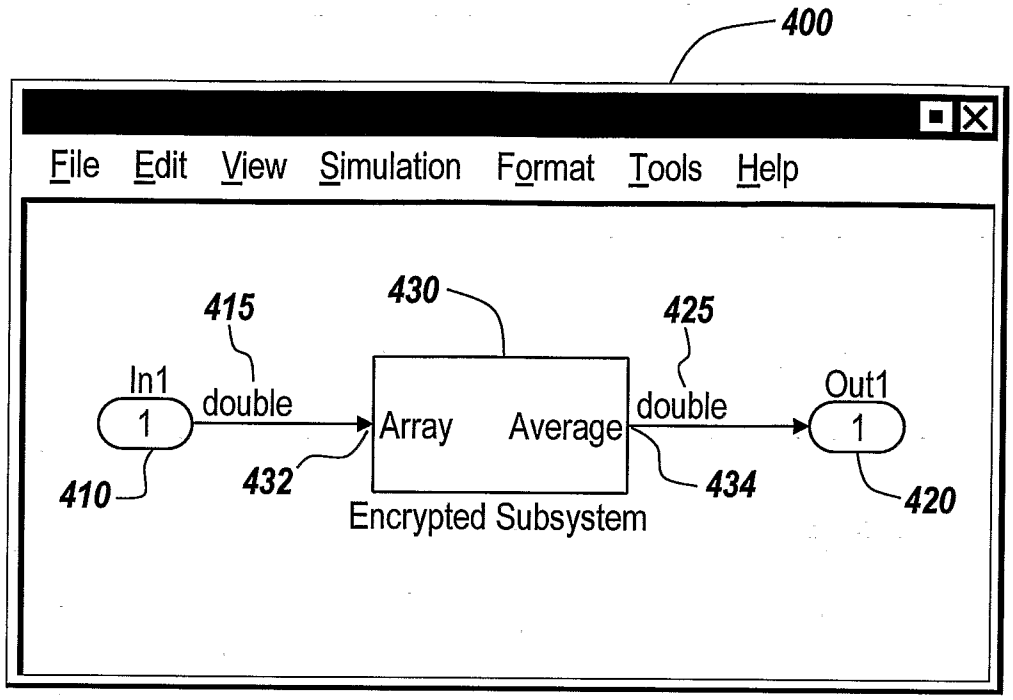


Fig. 4

500

```
>> get_param('EncryptedSubsystemExample/Encrypted Subsystem', 'Blocks')
??? Error using ==> get_param
Cannot access encrypted block diagram: EncryptedSubsystemExample/Encrypted Subsystem

>> |
```

Fig. 5

600

```
_Qg15QcdaPeOVO5NVG4kJGO _bSqObCxoIECPcW5PSpKMSD:
_bSqObCxoIECPcW5PSO = _6Uw166Qg15Ak_2Bkk_2Bk_.
_jwmAjgoHAhoXAisXMR14M_[_7W3H7_G5OZO7V_17gm9fg1];{
_cW5PcOZG5O3NVO5Nfjyuf2 _04JFOpF2Cok5Qg15VNKLHO;const
_Qg15QcdaPeOVO5NVG4kJGO *_2DsV2uW6TsGfhqPfnBnBn_=
&_6Uw166Qg15Ak_2Bk_2Bk_._jwmAjgoHAhoXAisXMR14M_[
_edaPe13IBlo_2Bk_6Rk06_];for(_04JFOpF2Cok5Qg15VNKLHO=
_7W3H7_G5OZO7V_17gm9fg1;_04JFOpF2Cok5Qg15VNKLHO<
_QhpLQWCqNYC5NVOSeeeee1;_04JFOpF2Cok5Qg15VNKLHO++) (
_bSqObCxoIECPcW5PSpKMSO+=_2DsV2uW6TsGfhqPfnBnBn_[
_04JFOpF2Cok5Qg15VNKLHO];}}_zyvjzaRmSbBH7W3HTudbT1.
_cX9ec5RkO6w3IB134KJG40=_bSqObCxoIECPcW5PSpKMSO/
_VhioOaAA99023498ALCh__._VNKLHx16Uw12Ewks7X7X71;
```

Fig. 6

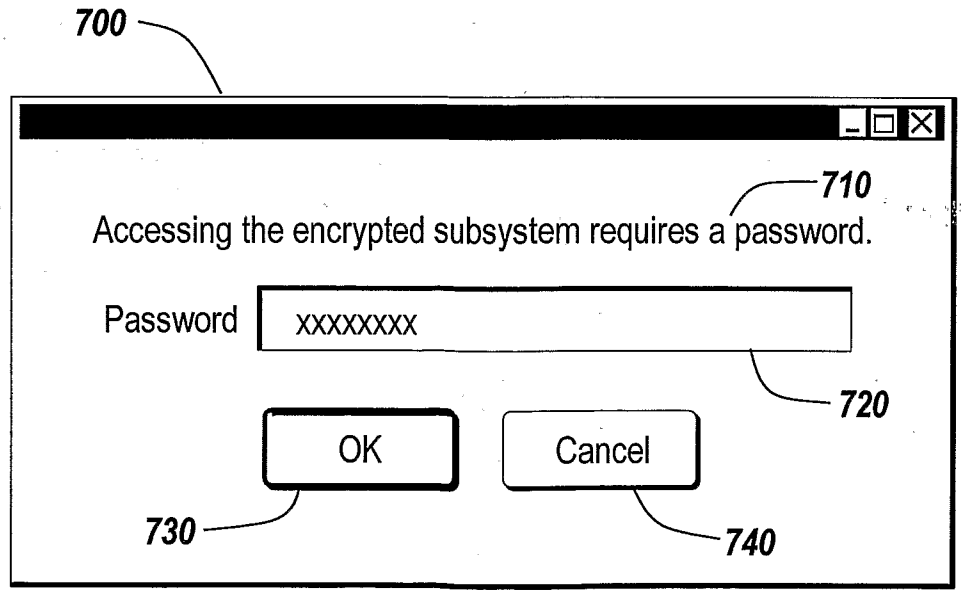


Fig. 7

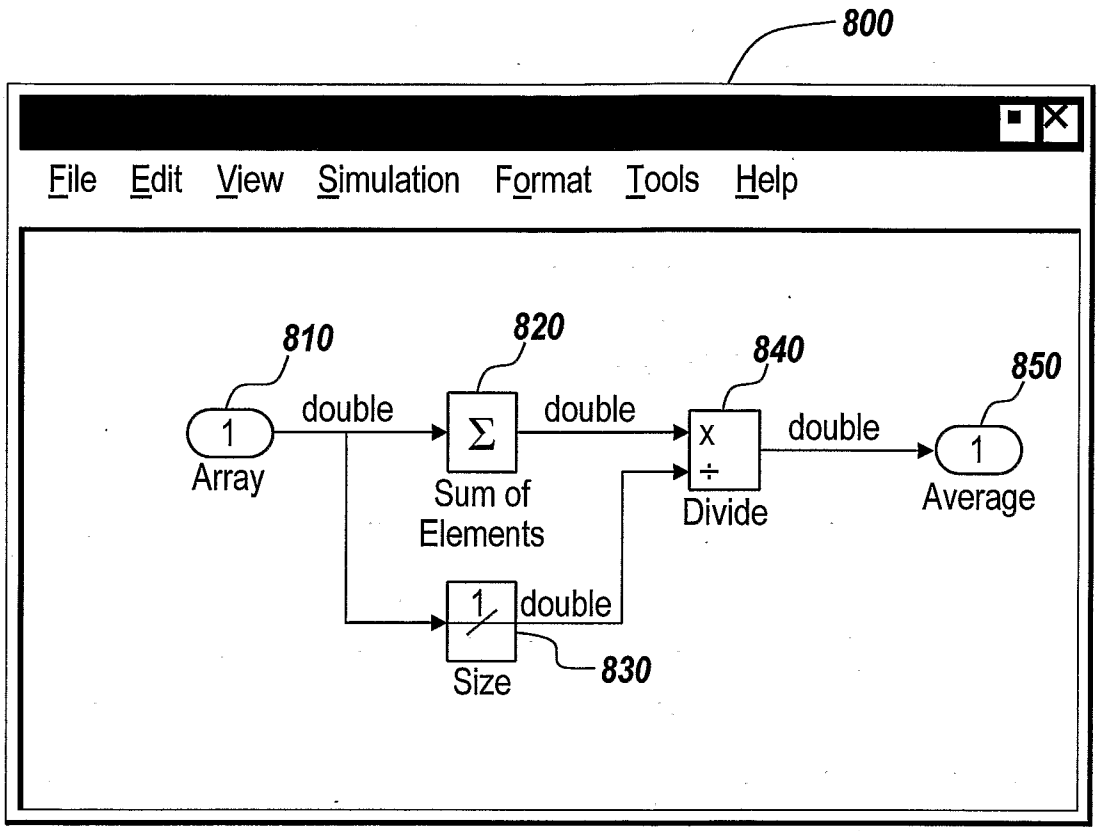

800
File Edit View Simulation Format Tools Help
810
820
840
850
1
Array
double
Σ
Sum of
Elements
double
x
÷
Divide
double
1
Average
1
double
Size
830

Fig. 8

900

```
>> get_param('EncryptedSubsystemExample/Encrypted Subsystem', 'Blocks')

ans =

    'Array'
    'Divide'
    'Size'
    'Sum of
Elements'
    'Average'

>>
```

Fig. 9

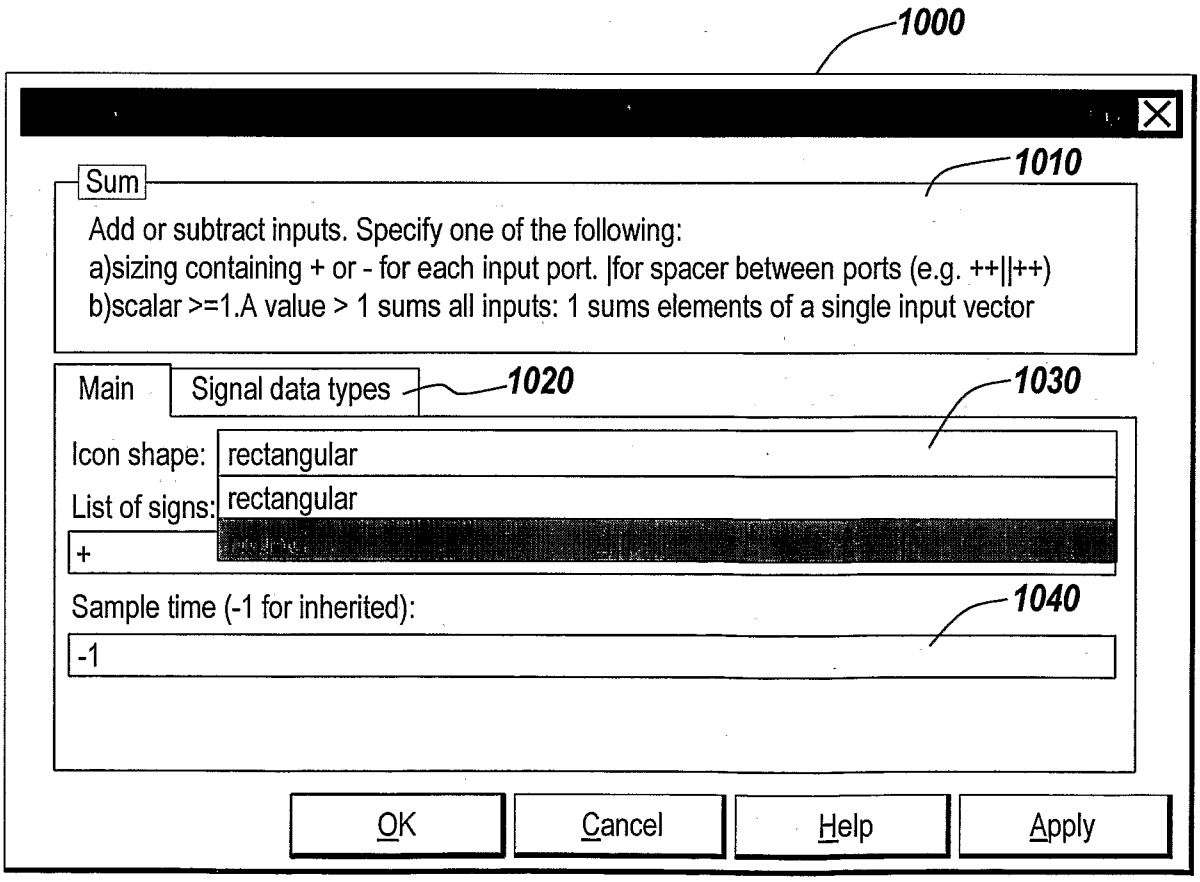


Fig. 10

1100

```
/* local block i/o variables*/
real_T rtb_SumofElements;

/* Sum: '<S1>/Sun of Elements' incorporates:
 *   Inport: '<Root>/In1'
 */
rtb_SumofElements = EncryptedSubsystemExample_U.In1[0];
{
  int_T i1;

  const real_T *u0 = &EncryptedSubsystemExample_U.In1[1]

  for (i1=0; i1 < 37926; i1++) {
    rtb_SumofElements += u0[i1];
  }
}

/* Outport: '<Root>/Out1' incorporates:
 *  Product: '<S1>/Divide'
 */
EncryptedSubsystemExample_Y.Out1 = rtb_SumofElements /
  EncryptedSubsystemExample_B.Size;
```

Fig. 11

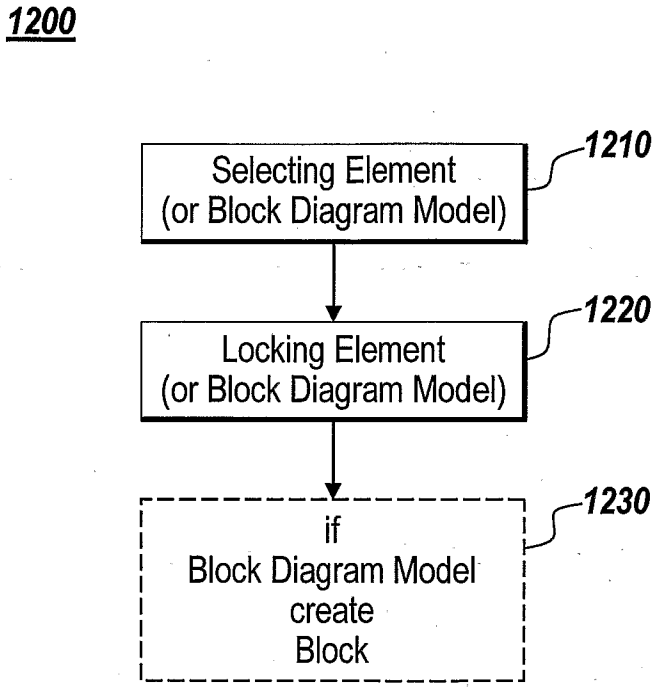

1200
Selecting Element (or Block Diagram Model)
1210
Locking Element (or Block Diagram Model)
1220
if Block Diagram Model create Block
1230

Fig. 12

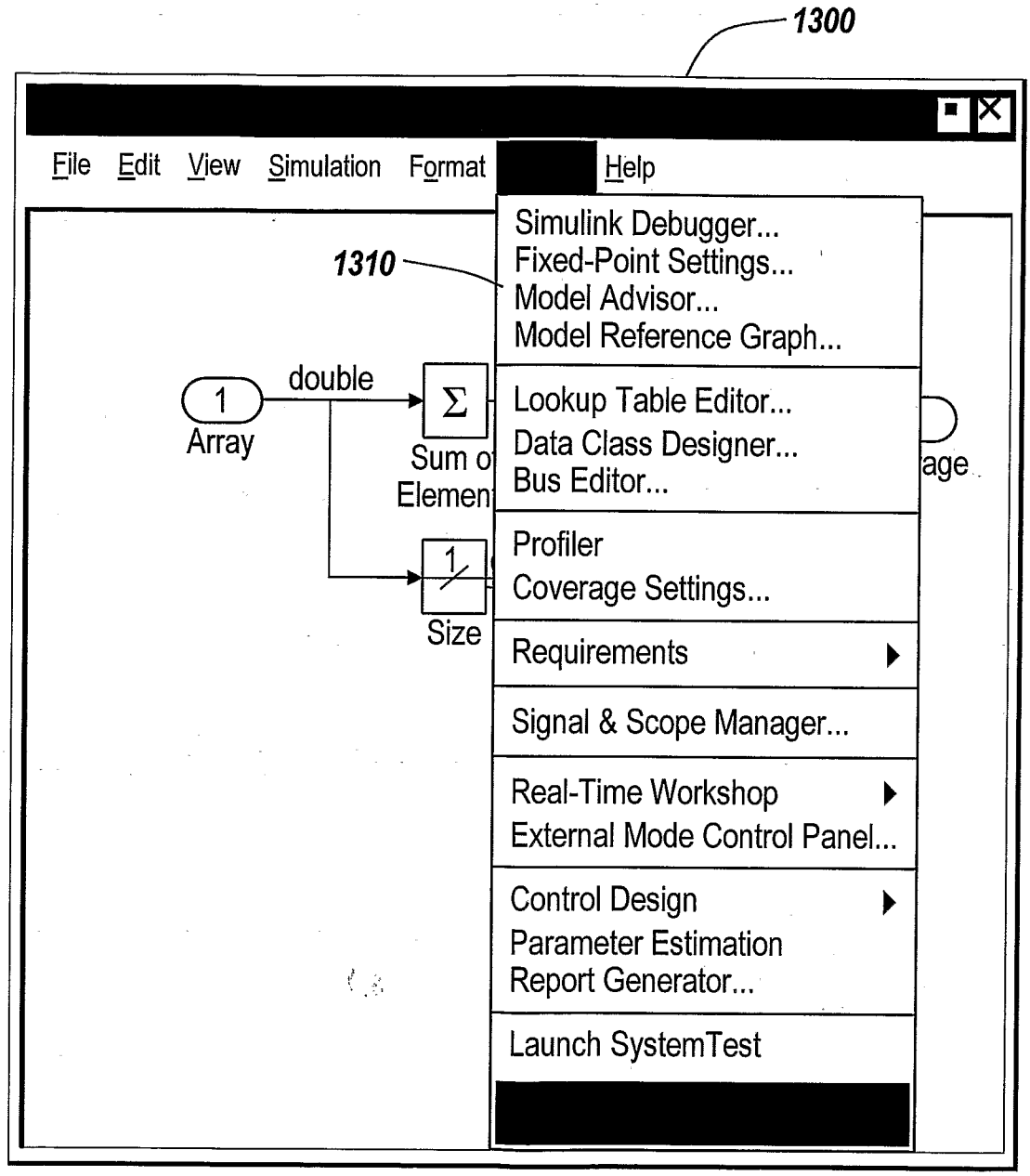

1300
File Edit View Simulation Format Help
1310
Simulink Debugger...
Fixed-Point Settings...
Model Advisor...
Model Reference Graph...
Lookup Table Editor...
Data Class Designer...
Bus Editor...
Profiler
Coverage Settings...
Requirements
Signal & Scope Manager...
Real-Time Workshop
External Mode Control Panel...
Control Design
Parameter Estimation
Report Generator...
Launch SystemTest
1
Array
double
Σ
Sum o
Elemen
1
Size
age

Fig. 13A

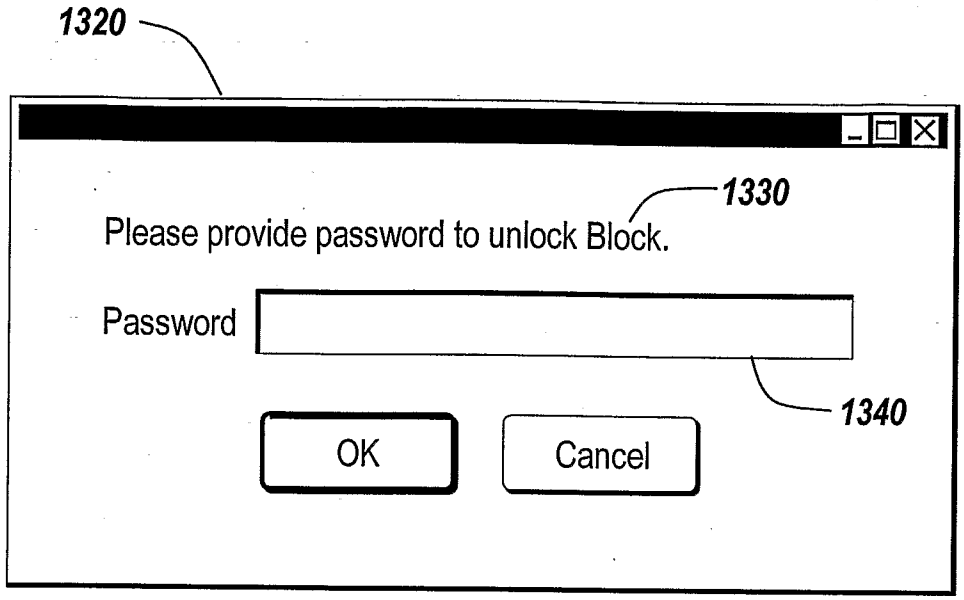


Fig. 13B

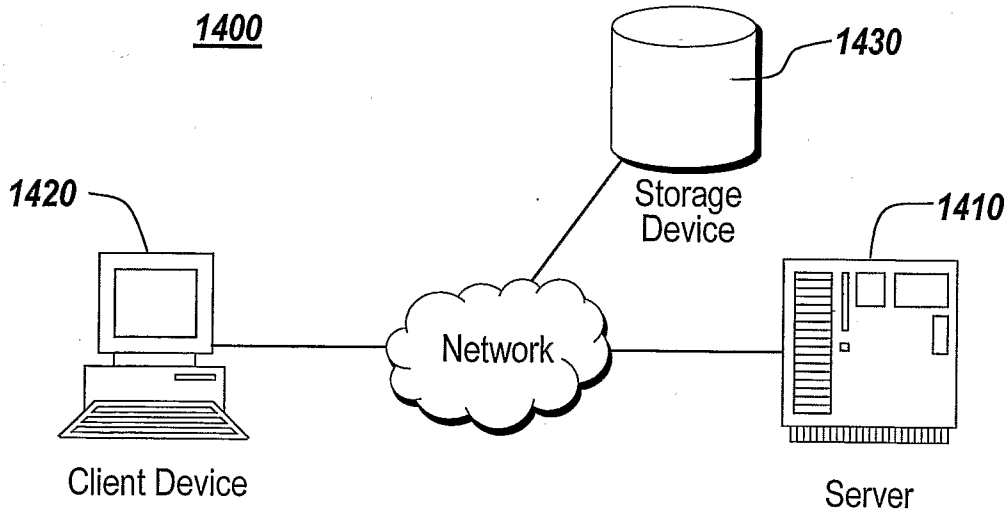


Fig. 14

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2006/036047

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F21/24 G06F9/44 G06F17/50

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2004/221256 A1 (MARTIN MAURICE [US] ET AL) 4 November 2004 (2004-11-04) paragraphs [0027] - [0066], [0291] - [0436]	1-43
A	BOULILA N: "Group support for distributed collaborative concurrent software modeling" AUTOMATED SOFTWARE ENGINEERING, 2004. PROCEEDINGS. 19TH INTERNATIONAL CONFERENCE ON LINZ, AUSTRIA SEPT. 20-24, 2004, PISCATAWAY, NJ, USA,IEEE, 20 September 2004 (2004-09-20), pages 422-425, XP010730483 ISBN: 0-7695-2131-2 the whole document	1-43

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
17 January 2007	31/01/2007

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Mäenpää, Jari

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2006/036047

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CERA C D ET AL: "Role-based viewing envelopes for information protection in collaborative modeling" COMPUTER AIDED DESIGN, ELSEVIER PUBLISHERS BV., BARKING, GB, vol. 36, no. 9, August 2004 (2004-08), pages 873-886, XP004511139 ISSN: 0010-4485 the whole document	1-43

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2006/036047

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2004221256 A1	04-11-2004	NONE	