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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 July 2017 (20.07.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/123367 A1

(51) International Patent Classification:
G05B 19/05 (2006.01) *G06F 21/12* (2013.01)

(21) International Application Number:
PCT/US2016/066295

(22) International Filing Date:
13 December 2016 (13.12.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/277,014 11 January 2016 (11.01.2016) US

(71) Applicant: **SIEMENS AKTIENGESELLSCHAFT** [DE/DE]; Wittelsbacherplatz 2, 80333 München (DE).

(71) Applicant *(for BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW only)*: **SIEMENS CORPORATION** [US/US]; 170 Wood Avenue South, Iselin, New Jersey 08830 (US).

(72) Inventor: **MARTINEZ CANEDO, Arquimedes**; 9 Wethersfield Road, Plainsboro, New Jersey 08536 (US).

(74) Agent: **RASHIDI-YAZD, Seyed Kaveh E.**; Siemens Corporation- Intellectual Property Dept., 3501 Quadrangle Blvd. Ste. 230, Orlando, Florida 32817 (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, DJ, 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, KH, KN, KP, KR, KW, 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,

[Continued on next page]

(54) Title: PROGRAM RANDOMIZATION FOR CYBER-ATTACK RESILIENT CONTROL IN PROGRAMMABLE LOGIC CONTROLLERS

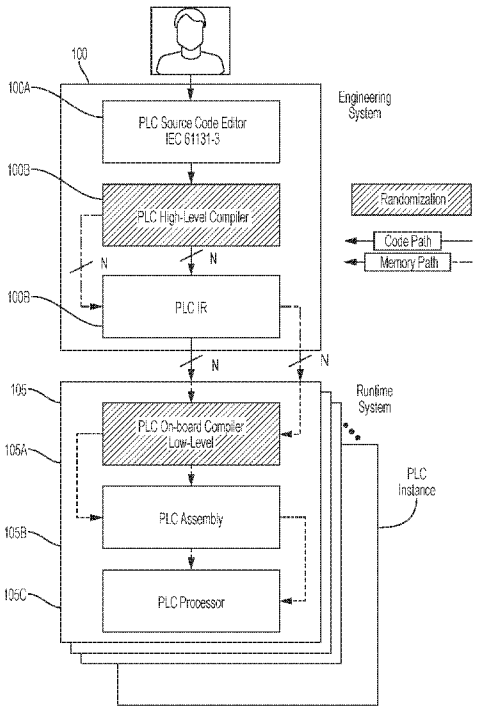

Fig. 1

(57) Abstract: A method for programmable logic controller (PLC) program randomization, the method comprising an engineering system computer receiving source code corresponding to a PLC program and compiling the source code into a plurality of functionally equivalent intermediate representations of the PLC program. Program structure of the PLC program is randomized during compilation such that each intermediate representation is unique among the plurality of intermediate representations. The engineering system computer transmits the plurality of intermediate representations to one or more PLCs.

WO 2017/123367 A1

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))*

PROGRAM RANDOMIZATION FOR CYBER-ATTACK RESILIENT CONTROL IN PROGRAMMABLE LOGIC CONTROLLERS

CROSS-REFERENCE TO RELATED APPLICATIONS

[1] This application claims the benefit of U.S. Provisional Application Serial No. 62/277,014 filed January 11, 2016, which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

[2] The present invention relates generally to randomizing programs for cyber-attack resilient control in programmable logic controllers. The disclosed technology may be applied to, for example, various automated production environments where industrial controllers such as programmable logic controllers (PLCs) and distributed control systems (DCS) are used.

BACKGROUND

[3] Conventional resilient programmable logic controller (PLC) architectures are designed to withstand random and isolated faults caused, for example, by component or power failure. The so called "high availability" PLCs offer hot standby features where in the case of a crash, another PLC can take over the control of the system immediately, within the same cycle. Over decades, these features were sufficient to cope with hardware failure and standard software faults.

[4] Unfortunately, conventional resilient architectures do not offer enough redundancy to cover faults originated by cyber-attacks because all the PLC instances are executing identical binaries in identical memory organization structures. Thus, a fault caused by a cyber-attack is likely to affect all the PLC instances and cause catastrophic consequences.

SUMMARY

[5] Embodiments of the present invention address and overcome one or more of the above shortcomings and drawbacks, by providing methods, systems, and apparatuses related to randomizing programs for cyber-attack resilient control in PLCs. More specifically, binaries of PLC software are randomized in each of the instances of a redundant PLC architecture. Because the PLC instances are executing different binaries, the overall system has more resilience than conventional systems with respect to cyberattacks. Moreover, in some embodiments, each the

memory organization structures of each PLC are also varied to provide additional protection from cyberattacks.

[6] According to some embodiments, a method for PLC program randomization includes an engineering system computer receiving source code corresponding to a PLC program and compiling the source code into a plurality of functionally equivalent intermediate representations of the PLC program. The program structure of the PLC program is randomized during compilation such that each intermediate representation is unique among the plurality of intermediate representations. In some embodiments, randomization is performed prior to runtime of the program, while in other embodiments it may be performed at runtime, for example, using just-in-time compilation techniques generally known in the art. The engineering system computer transmits the plurality of intermediate representations to one or more PLCs.

[7] In some embodiments of the aforementioned method for PLC program randomization, the PLC receives a first intermediate representation of the PLC program and compiles it into PLC assembly code. Program structure of the first intermediate representation is randomized during compilation. This PLC assembly code is then executed by the PLC. Following failover of the PLC, the first intermediate representation may be re-compiled into new PLC assembly code which is then executed by the PLC. The program structure of the first intermediate representation can be randomized again during re-compilation.

[8] Various techniques may be applied to randomize program structure in the aforementioned method for PLC program randomization. For example, in some embodiments, the program structure of the PLC program is randomized during compilation by randomizing a memory layout of a plurality of data blocks used by the PLC program. For example, the memory layout of plurality of data blocks is randomized by assigning a unique memory address to each data block. The data blocks may be sorted by type during compilation to optimize memory access in the memory layout. In other embodiments, the program structure of the PLC program is randomized during compilation by randomizing usage of function blocks used by the PLC program. For example, the ordering of parameters used in calling each function block may be randomized or ordering cyclic variables used by each function block may be randomized. Additionally, usage of the plurality of function blocks may be randomized by constructing a

control flow graph where conditional statements are transformed into equivalent control flow constructs or constructing a data flow graph where conditional statements are transformed into equivalent data flow expressions. Then, the order of the control flow constructs or data flow expressions may randomized. In other embodiments, function blocks usage is randomized inserting one or more non-functional function blocks into the plurality of function blocks. In one embodiment, each function block is identified by a triple comprising a numerical identifier and the program structure of the PLC program is randomized by assigning a random value to each numerical identifier and sorting the plurality of function blocks by numerical identifier. In some embodiments, the program structure of the PLC program is randomized during compilation by reordering one or more organization blocks in the PLC program.

[9] According to other embodiments of the present invention, a method for PLC program randomization includes a PLC receiving an intermediate representation of a PLC program and compiling the intermediate representation into PLC assembly code. The program structure of the intermediate representation is randomized during compilation with respect to usage of one or more program blocks. The PLC can then execute the PLC assembly code. Following failover of the PLC, the PLC may re-compile the intermediate representation into a new PLC assembly code, again randomizing program structure of the intermediate representation during re-compilation. The new PLC assembly code may then be executed by the PLC.

[10] In some embodiments of the aforementioned method for PLC program randomization, program structure of the intermediate representation is randomized during compilation by randomizing address values for one or more data blocks used by the PLC program. Other techniques may alternatively (or additionally) be used for randomizing the intermediate representation during compilation including (a) randomizing control flow of function blocks and functions used by the PLC program; (b) randomizing data flow of function blocks and functions used by the PLC program; and/or (c) reordering one or more organization blocks used by the PLC program in the intermediate representation.

[11] According to other embodiments, a system for programmable logic controller (PLC) program randomization includes a computer readable storage medium storing source code

corresponding to a PLC program; and engineering system and a plurality of programmable logic controllers. The engineering system is configured to compile the source code into a plurality of functionally equivalent intermediate representations of the PLC program. The program structure of the PLC program is randomized during compilation such that each intermediate representation is unique among the plurality of intermediate representations Each programmable logic controller is configured to receive an intermediate representation of the PLC program from the engineering system and compile the intermediate representation into PLC assembly code, wherein program structure of the intermediate representation is randomized during compilation with respect to usage of one or more program blocks.

[12] Additional features and advantages of the invention will be made apparent from the following detailed description of illustrative embodiments that proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[13] The foregoing and other aspects of the present invention are best understood from the following detailed description when read in connection with the accompanying drawings. For the purpose of illustrating the invention, there are shown in the drawings embodiments that are presently preferred, it being understood, however, that the invention is not limited to the specific instrumentalities disclosed. Included in the drawings are the following Figures:

[14] FIG. 1 provides an example randomization done at an engineering system and the runtime system, according to some embodiments;

[15] FIG. 2 provides an illustration of a Quad-Redundant Control Architecture, as it may be implemented in some embodiments;

[16] FIG. 3 provides an illustration of PLC program randomization implemented through an API, according to some embodiments;

[17] FIG. 4 illustrates an example randomization of legacy PLC projects, as it may be implemented in some embodiments;

[18] FIG. 5 shows a data block randomization changes the address and offset of data block (DB) variables, which may be utilized in some embodiments;

[19] FIG. 6 shows how DB block optimization eliminates memory padding and re-arranges the memory layout, according to some embodiments;

[20] FIG. 7 provides an illustration of how function block parameter reordering changes the binary representation of PLC programs, according to some embodiments;

[21] FIG. 8 provides an illustration of cyclic variable reordering introduces binary changes to the PLC binary programs, according to some embodiments;

[22] FIG. 9 illustrates a cyclic variable reordering introduces binary changes to the PLC binary programs, according to some embodiments; and

[23] FIG. 10 provides an illustration of organization block randomization introduces variability in the timing and priority of OBs, according to some embodiments.

DETAILED DESCRIPTION

[24] Systems, methods, and apparatuses are described herein which relate generally to randomizing programs for cyber-attack resilient control in PLCs. More specifically, the techniques described herein allow for the execution of randomized binaries of PLC software in each of the instances of a redundant PLC architecture. This architecture is used to improve the system resilience (i.e., a factory and its subsystems) against failure modes originated by cyber-attacks. The architecture, referred to herein as REsilient CONtrol , or "RECON", relies on arrays of legacy PLCs as building blocks for creating a resilient Industrial Control Systems (ICS). Redundant PLCs are provided the same sensor inputs from the physical system and all PLCs execute a functionally identical program; however, each PLC contains a different randomized binary. In case of a successful exploit against the primary PLC, a fault is detected by the redundant system and a backup PLC takes control. Using different randomized binaries reduces the chances of the same exploit affecting the backup PLCs. Moreover, after a fault is detected, a new randomized binary can be re-compiled on-the-fly to the affected PLC instance to increase the availability of the system. As described in further detail below, in some

embodiments of RECON, randomization occurs in two stages: (1) at the high-level compile time where PLC source code is transformed into the PLC intermediate representation; and (2) at runtime where the PLC intermediate representation is transformed into assembly instructions for execution in the PLC processor. The different randomization opportunities that each code representation offers may be leveraged to create a large combination of code and memory paths that enable a scalable RECON architecture.

[25] FIG. 1 provides a high-level overview of randomization performed at an Engineering System 100 (e.g., Siemens TIA Portal Engineering System) and a Runtime System 105 (e.g., Siemens PLC Operating System), according to some embodiments. PLCs are hard real-time control systems designed to interact with and control physical processes through sensors and actuators in harsh environments and for very long periods of time. PLCs are programmed in an Engineering System 100 using domain-specific, high-level PLC programming languages (e.g., IEC 61131-3) designed for physical system experts (i.e., electrical, mechanical engineers).

[26] As shown in FIG. 1, the PLC source code is developed in a Source Code Editor 100A in the Engineering System 100. The source code compiled by a High-Level Compiler 100B into a PLC Intermediate Representation (IR) 100C. The PLC IR 100C allows the Engineering System 100 to target multiple PLC versions with different hardware and firmware. In some embodiments, multiple PLC languages are used to develop the program. An Integrated Development Environment (e.g., part of the Source Code Editor 100A) compiles the various source code languages into the PLC IR 100C that is retargetable to various PLC architectures (e.g., x86, MIPS, ARM, etc.). Abstract Syntax Tree (AST) randomization of the code and memory layout may additionally be performed.

[27] To deploy the PLC program to the field, the PLC IR 100C is "downloaded" by the Engineering System 100 into the Runtime System 105 of the PLC; the Runtime System 105 is analogous to an operating system in a general purpose computer. The Runtime System then compiles the PLC IR into PLC Assembly code using an On-Board Compiler 105A (included in firmware of the PLC). Finally, the PLC Assembly 105B is executed by the PLC Processor 105C in a cyclic manner. Notice that in the case of a traditional resilient PLC architecture, the same PLC IR is distributed to all PLC instances, and assuming that all instances have the same version

of the on-board compiler, all instances generate identical PLC Assembly for the common PLC IR.

[28] To improve the fault-tolerance against faults caused by deliberate and pervasive cyber-attacks, in some embodiments RECON utilizes code and memory path randomization in the high-level and in the on-board compilers. High-level compiler randomization takes advantage of a suite of rich compiler transformation and optimizations done at the Abstract Syntax Tree (AST) and Control Flow Graph (CFG) levels. The first randomization step introduces high variability at the PLC IR level.

[29] The second randomization step done in the on-board compiler has more restrictions imposed by the target assembly language and has less flexibility than the randomization done at a higher-level. However, the on-board compiler randomization provides a second line of defense to the system in case that an attacker compromises the first-stage of randomization. The on-board randomization mutates the PLC IR into a unique binary that cannot be accessed outside of the PLC. One way to further protect any outsiders from reading the on-board randomized binary is through PLC configuration. Thus, the on-board randomization hides the binary during the software development and maintenance cycle. Moreover, it should be noted that on-board randomization offers the ability to redeploy after failover using a newly randomized binary that is functionally equivalent to the prior version of the PLC software. In this way, a PLC that succumbs to cyberattack at the code level can re-launch with additional security measures which prevent the attack from being repeated.

[30] FIG. 2 shows a Quad-Redundant Control architecture 200, according to some embodiments. State-of-the-art PLCs provide more computation power than legacy PLCs with the use of multi-core processors. The Quad-Redundant Control PLC architecture 200 shown in FIG. 2 may be implemented, for example, using a plurality of dual-core PLCs (SIMATIC S7-1500). In addition of providing more logical PLCs, a benefit of this architecture is the virtualization of the PLCs in multi-core where each PLC has a disjointed memory and code spaces and prevents software failures in the master PLC to affect the slave PLC. Having virtualized dual-redundancy in the same core allows a faster recovery in case of a failure.

[31] In FIG. 2, the Engine represents the physical system under control. Two sensors (S1, S2) measure the temperature and rotations-per-minute of the Engine. Two redundant I/O boards (I/O 1, I/O 2) connect the sensor data to the PLCs (PLC 1, ..., PLC 4). In every Scan Cycle Time (e.g., 100ms), the PLC executes three main stages: Read, Process, and Write. It is important to note that the Read and Write are not done directly to sensors and actuators, but through the PLC's Process Image. The Process Image is a buffer that decouples the execution from the physical process and allows the PLC to recover in case of any interrupts or faults. That is, the process image is guaranteed to be consistent in every cycle boundaries (e.g., 100ms, 200ms, 300ms). The Process Image is checked for consistency by the DB Consistency Check. In the Process stage, RECON uses different randomized binaries compiled using the method described in FIG. 1. Since the code and memory paths are different for every randomized binary, the Synchronizer checks for functional equivalence of the code executed in the master and slave PLCs. Functional equivalence may be accomplished through the insertion of runtime checkpoints at equivalent locations at various granularities. For example, if the randomization is done at the basic block granularity (e.g., micro-instruction shuffling), checkpoints can be inserted at basic block boundaries.

[32] FIG. 3 is a system diagram which provides additional details of how PLC program randomization may be implemented, according to some embodiments. As is understood in the art, a PLC comprises an operating system and a user program. The operating system provides the intrinsic PLC functionality (e.g., handling of restart and errors, memory management, calling the user program, etc.). As shown in FIG. 3, the user program comprises all "blocks" that perform the automation task. The user program is programmed with program blocks including Function Blocks (FB) and Functions (FC). FBs and FCs reference local and global variables organized in Data Blocks (DB). In PLCs, the user program is typically executed cyclically (non-cyclic programs are also possible) in an Organization Block (OB). The main cycle OB always exists and it is available when a PLC program is initialized and executed in an infinite loop.

[33] The PLC program structure shown in FIG. 3 is fixed and provides a clear structure with different block types. This example illustrates the method to connect the RECON system to an Engineering System 305. RECON 315 in this example comprises software components which parse the PLC structure, apply transformations, and package a new PLC structure. More

specifically, using an Application Programming Interface (API) 310, RECON 315 accesses and randomizes the PLC program before it is downloaded. Modifying the block structure of a PLC program has a profound effect in the generated code because the mappings of blocks (OB, FB, FC, DB) to binary representation change. Therefore, modification to the PLC program structure achieves the binary diversification sought by RECON. Using the general architecture set forth in FIG. 3, randomization of all block types in the PLC program structure may be achieved

[34] RECON may be used to perform PLC program randomization in both state-of-the-art and in legacy systems. FIG. 4 shows two techniques how randomization of legacy PLC projects may be performed using RECON, according to some embodiments. As shown in FIG. 4, the PLC programs can be extracted from a Legacy PLC 405 into the Engineering System 410. Alternatively, instead of extracting the PLC program from the Legacy PLC 405, a Legacy Database 415 can provide the PLC project files. The condition for either mechanism is that the project files are unencrypted and unprotected, as shown in FIG. 4.

[35] Elaborating on the techniques described above, PLC program randomization is performed by three software components: a Data Block Randomization Component, a Function Block and Function Randomization Component, and an Organization Block Randomization Component. Each of these components is described in further detail below.

[36] DBs organize the memory in addressable blocks that are used by the program to access variables, sensors, and actuators. A DB is a collection of named variables referred to as Tags. Tags are reserved memory areas for values in the PLC. Tags contain a data type (e.g., Bool, Integer, etc.), a value (e.g., "15"), and a physical address in the DB.

[37] FIG. 5 illustrates how block randomization changes the address and offset of DB variables, according to some embodiments. As shown in FIG. 5, the "Motor_1" tag in PLC 1 has a Bool type and an address "%I0.0". After processing by the Data Block Randomization Component, the address of the same variable in PLC 2 can be changed to "%I2.0". Similarly, the offset in static variable "tag" in PLC 1 of Bool type can be changed from "0.0" to "0.1" by introducing the "new_tag" before "tag" in the symbol table in PLC2. These changes diversify the memory layout in a PLC in such a way that the same cyber-exploit is less likely to be effective in two PLCs with different memory layouts.

[38] An additional layer of randomization is to enable the DB block optimization where all tags are sorted by their data type. The sorting minimizes the data gaps (padding) between tags and optimizes the memory access for the target PLC processor as shown in FIG. 6. In this example, the Standard DB 605 has data gaps between variables that are eliminated in the Optimized DB 610. In combination, the two presented DB randomization techniques allow RECON to generate hundreds of equivalent memory layouts for PLC diversified programs.

[39] The Function Block and Function Randomization Component changes the structure of both the code and memory stack, and therefore it is important for RECON to create program variations. The difference between FB and FC lies in the cyclic data storage. FCs are blocks without data storage. This means that the values FCs are stateless and the block variables cannot be persisted over cycles. And FBs, on the other hand, are blocks with cyclic data storage. This means that FBs are stateful and block variable values are persisted over cycles. Both FB and FC are programmed using a variety of high-level languages including graphical and textual languages. The most popular PLC programming language is the Structured Control Language (SCL) which is based on PASCAL. SCL is also based on the Structured Text (ST) specification for PLC programming standardized by the IEC 61131-3. The FB and FC randomization techniques described below are based on SCL for illustration purposes; however, it should be understood that these techniques may be generalized and applied to any PLC programming language which provides similar functionality.

[40] Function blocks and functions have a declaration comprising input, output, in-out parameters, and the body of the function. This declaration is common to all languages, including SCL. FIG. 7 shows an exemplary function block declaration for a "TEST" program comprising an input parameter "FINALVAL", an in-out parameter "IQ1", and an output parameter "CONTROL". One technique to introduce randomization is to reorder the parameters when calling the function block as shown by the two examples on the right. Because the parameters are named, the order in which the parameters are listed does not change the meaning of the program. However, this creates a different memory layout for the two programs. In real PLC programs, the number of parameters tends to be very large (sometimes hundreds of parameters), and therefore this technique can be very effective in creating a large number of different but equivalent SCL programs.

[41] A similar technique is applicable to FB's cyclic variables that are persisted across cycles. FIG. 8 illustrates how cyclic variable reordering introduces binary changes to the PLC binary programs, according to some embodiments. As shown in FIG. 8, the order of the variables PID_CONTROLLER_1 and PID_CONTROLLER_2 can be rearranged such that cyclic variables occupy different memory locations in different PLC instances.

[42] The SCL code itself can be randomized in both the control flow and data flow levels. For example, for the control flow, the extensible markup language (XML) file containing the SCL statements may be parsed and the control flow graph may be reconstructed where conditional statements are transformed into equivalent control flow constructs.

[43] FIG. 9 shows how control flow and data flow transformations introduce binary diversification at the SCL code level, according to some embodiments. In FIG. 9, for example, the structure of the if-else statement is inverted with a "NOT" instruction such that the body of the "if" statement in PLC 1 becomes the body of the else statement in PLC 2, and vice-versa. For data flow, the XML file containing the SCL statement may be parsed, and the data flow graph may reconstructed where expressions are transformed into equivalent data flow expressions. For example, FIG. 7 shows that the expression "IQ1 := IQ1 * 2" in PLC 1 is transformed into "IQ1 := IQ1 + IQ1" in PLC 2. Additionally, additional expressions such as "N := 0" and "SUM := 0" may be inserted that do not contribute to the control program but generate binary diversity at the code level when compiled. This is similar to a NOOP (no-operation) randomization technique. It is important to note that NOOP operands do not exist in some PLCs but inserting operations that do not contribute functionally to the program is functionally equivalent.

[44] In some embodiments, an ordering randomization technique for FBs and FCs is applied that affects their arrangement in memory. Every FB and FC has a triple <name, type, number>, where name is an arbitrary name provided by the user in the PLC program, the type is an enumeration depending on the block type (e.g., FB, FC, OB, DB), and number is a unique identifier associated to the block. The key observation is that the locations of blocks in the PLC memory correspond to their number as they are first sorted and then downloaded to the PLC for execution. Thus, an ordering randomization technique may be applied to randomize the number

in the triples to change their order during execution. Additionally (or alternatively), dummy FCs and FBs may be inserted to take up memory space and arbitrarily move other FBs and FCs that provide useful work. This transformation is also possible through the open interfaces and can also be extended to DBs.

[45] OBs are the interface between the operating system and the user program. User programs can only be called by an OB and therefore are an important part of the PLC program structure. OBs are called by the operating system cyclically or when certain events occur. In most PLCs, there are several types of OBs (or similar concept in other vendors) that the user can chose for their programs depending on the desired functionality. For example, there are startup OBs, cyclic program execution OBs, interrupt (time-of-day, cyclic, time-delay, hardware) OBs, and error (asynchronous and synchronous) OBs. OBs are identified by a number, and different numbers have different priorities. For example, "OB 1" is the cyclic program OB with the highest priority, and OB 10 is a time-of-day interrupt OB with a lower priority. For example, Some PLCs provide multiple interrupts of the same type, for example, in Siemens PLCS, OB 32, OB 33, OB 34, and OB 35 are cyclic interrupt OBs that users have at their disposal to organize their program.

[46] The Organization Block Randomization Component reorders the OBs such that different PLC programs use different OBs (with the same priority and real-time guarantees). For example, FIG. 10 shows how a PLC program structure with two OBs (OB 1 and OB 32) can be transformed into a program structure with three OBs (OB 1, OB 32, and OB 35). In this example, the Controller code is split into two and the first half is assigned to OB 32, and the second half is assigned to OB 35. Notice that this creates a dependency between OB 35 and OB 32. However, synchronization between OBs is supported in PLCs. FIG. 10 also illustrates that the randomization that RECON applies to OB, FB, FC, and DB to create a large combinatorial space can be leveraged to create diversified PLC binaries.

[47] The processors described herein as used by control layer devices may include one or more central processing units (CPUs), graphical processing units (GPUs), or any other processor known in the art. More generally, a processor as used herein is a device for executing machine-readable instructions stored on a computer readable medium, for performing tasks and may

comprise any one or combination of, hardware and firmware. A processor may also comprise memory storing machine-readable instructions executable for performing tasks. A processor acts upon information by manipulating, analyzing, modifying, converting or transmitting information for use by an executable procedure or an information device, and/or by routing the information to an output device. A processor may use or comprise the capabilities of a computer, controller or microprocessor, for example, and be conditioned using executable instructions to perform special purpose functions not performed by a general purpose computer. A processor may be coupled (electrically and/or as comprising executable components) with any other processor enabling interaction and/or communication there-between. A user interface processor or generator is a known element comprising electronic circuitry or software or a combination of both for generating display images or portions thereof. A user interface comprises one or more display images enabling user interaction with a processor or other device.

[48] Various devices described herein including, without limitation, the control layer devices and related computing infrastructure, may include at least one computer readable medium or memory for holding instructions programmed according to embodiments of the invention and for containing data structures, tables, records, or other data described herein. The term "computer readable medium" as used herein refers to any medium that participates in providing instructions to one or more processors for execution. A computer readable medium may take many forms including, but not limited to, non-transitory, non-volatile media, volatile media, and transmission media. Non-limiting examples of non-volatile media include optical disks, solid state drives, magnetic disks, and magneto-optical disks. Non-limiting examples of volatile media include dynamic memory. Non-limiting examples of transmission media include coaxial cables, copper wire, and fiber optics, including the wires that make up a system bus. Transmission media may also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

[49] An executable application, as used herein, comprises code or machine readable instructions for conditioning the processor to implement predetermined functions, such as those of an operating system, a context data acquisition system or other information processing system, for example, in response to user command or input. An executable procedure is a segment of code or machine readable instruction, sub-routine, or other distinct section of code or portion of

an executable application for performing one or more particular processes. These processes may include receiving input data and/or parameters, performing operations on received input data and/or performing functions in response to received input parameters, and providing resulting output data and/or parameters.

[50] The functions and process steps herein may be performed automatically, wholly or partially in response to user command. An activity (including a step) performed automatically is performed in response to one or more executable instructions or device operation without user direct initiation of the activity.

[51] The system and processes of the figures are not exclusive. Other systems, processes and menus may be derived in accordance with the principles of the invention to accomplish the same objectives. Although this invention has been described with reference to particular embodiments, it is to be understood that the embodiments and variations shown and described herein are for illustration purposes only. Modifications to the current design may be implemented by those skilled in the art, without departing from the scope of the invention. As described herein, the various systems, subsystems, agents, managers and processes can be implemented using hardware components, software components, and/or combinations thereof. No claim element herein is to be construed under the provisions of 35 U.S.C. 112, sixth paragraph, unless the element is expressly recited using the phrase “means for.”

CLAIMS

1. A method for programmable logic controller (PLC) program randomization, the method comprising:

receiving, by an engineering system computer, source code corresponding to a PLC program;

compiling, by the engineering system computer, the source code into a plurality of functionally equivalent intermediate representations of the PLC program, wherein program structure of the PLC program is randomized during compilation such that each intermediate representation is unique among the plurality of intermediate representations; and

transmitting, by the engineering system computer, the plurality of intermediate representations to one or more PLCs.

2. The method of claim 1, further comprising:

receiving, by the PLC, a first intermediate representation of the PLC program;

compiling, by the PLC, the first intermediate representation into a PLC assembly code, wherein program structure of the first intermediate representation is randomized during compilation; and

executing, by the PLC, the PLC assembly code.

3. The method of claim 2, further comprising:

following failover of the PLC, re-compiling the first intermediate representation into a new PLC assembly code, wherein program structure of the first intermediate representation is randomized during re-compilation; and

executing, by the PLC, the new PLC assembly code.

4. The method of claim 1, wherein the program structure of the PLC program is randomized during compilation by randomizing a memory layout of a plurality of data blocks used by the PLC program.

5. The method of claim 4, wherein the memory layout of plurality of data blocks is randomized by assigning a unique memory address to each data block.

6. The method claim 5, further comprising:

sorting the plurality of data blocks by type during compilation to optimize memory access in the memory layout.

7. The method of claim 1, wherein the program structure of the PLC program is randomized during compilation by randomizing usage of a plurality of function blocks used by the PLC program.

8. The method of claim 7, wherein randomization of usage of the plurality of function blocks used by the PLC program comprises:

randomizing ordering of parameters used in calling each function block.

9. The method of claim 7, wherein randomization of usage of the plurality of function blocks used by the PLC program comprises:

randomizing ordering cyclic variables used by each function block.

10. The method of claim 7, wherein randomization of usage of the plurality of function blocks used by the PLC program comprises randomizing control flow of each function block by:

constructing a control flow graph where conditional statements are transformed into equivalent control flow constructs; and

randomizing ordering of the control flow constructs.

11. The method of claim 7, wherein randomization of usage of the plurality of function blocks used by the PLC program comprises randomizing data flow of each function block by:

constructing a data flow graph where conditional statements are transformed into equivalent data flow expressions; and

randomizing ordering of the data flow expressions.

12. The method of claim 7, wherein randomization of usage of the plurality of function blocks used by the PLC program comprises inserting one or more non-functional function blocks into the plurality of function blocks.

13. The method of claim 1, wherein the PLC program comprises a plurality of function blocks, each identified by a triple comprising a numerical identifier and the program structure of the PLC program is randomized by:

assigning a random value to each numerical identifier, and

sorting the plurality of function blocks by numerical identifier.

14. The method of claim 1, wherein program structure of the PLC program is randomized during compilation by reordering one or more organization blocks in the PLC program.

15. A method for programmable logic controller (PLC) program randomization, the method comprising:

receiving, by a PLC, an intermediate representation of a PLC program;

compiling, by the PLC, the intermediate representation into PLC assembly code, wherein program structure of the intermediate representation is randomized during compilation with respect to usage of one or more program blocks; and

executing, by the PLC, the PLC assembly code.

16. The method of claim 15, further comprising:

following failover of the PLC, re-compiling the intermediate representation into a new PLC assembly code, wherein program structure of the intermediate representation is randomized during re-compilation; and

executing, by the PLC, the new PLC assembly code.

17. The method of claim 15, wherein program structure of the intermediate representation is randomized during compilation by randomizing address values for one or more data blocks used by the PLC program.

18. The method of claim 15, wherein program structure of the intermediate representation is randomized during compilation by randomizing control flow of function blocks and functions used by the PLC program.

19. The method of claim 15, wherein program structure of the intermediate representation is randomized during compilation by randomizing data flow of function blocks and functions used by the PLC program.

20. The method of claim 15, wherein program structure of the intermediate representation is randomized during compilation by reordering one or more organization blocks used by the PLC program in the intermediate representation.

21. A system for programmable logic controller (PLC) program randomization, the system comprising:

a computer readable storage medium storing source code corresponding to a PLC program;

an engineering system configured to compile the source code into a plurality of functionally equivalent intermediate representations of the PLC program, wherein program structure of the PLC program is randomized during compilation such that each intermediate representation is unique among the plurality of intermediate representations; and

a plurality of programmable logic controllers, each programmable logic controller configured to:

receive an intermediate representation of the PLC program from the engineering system

compile the intermediate representation into PLC assembly code, wherein program structure of the intermediate representation is randomized during compilation with respect to usage of one or more program blocks.

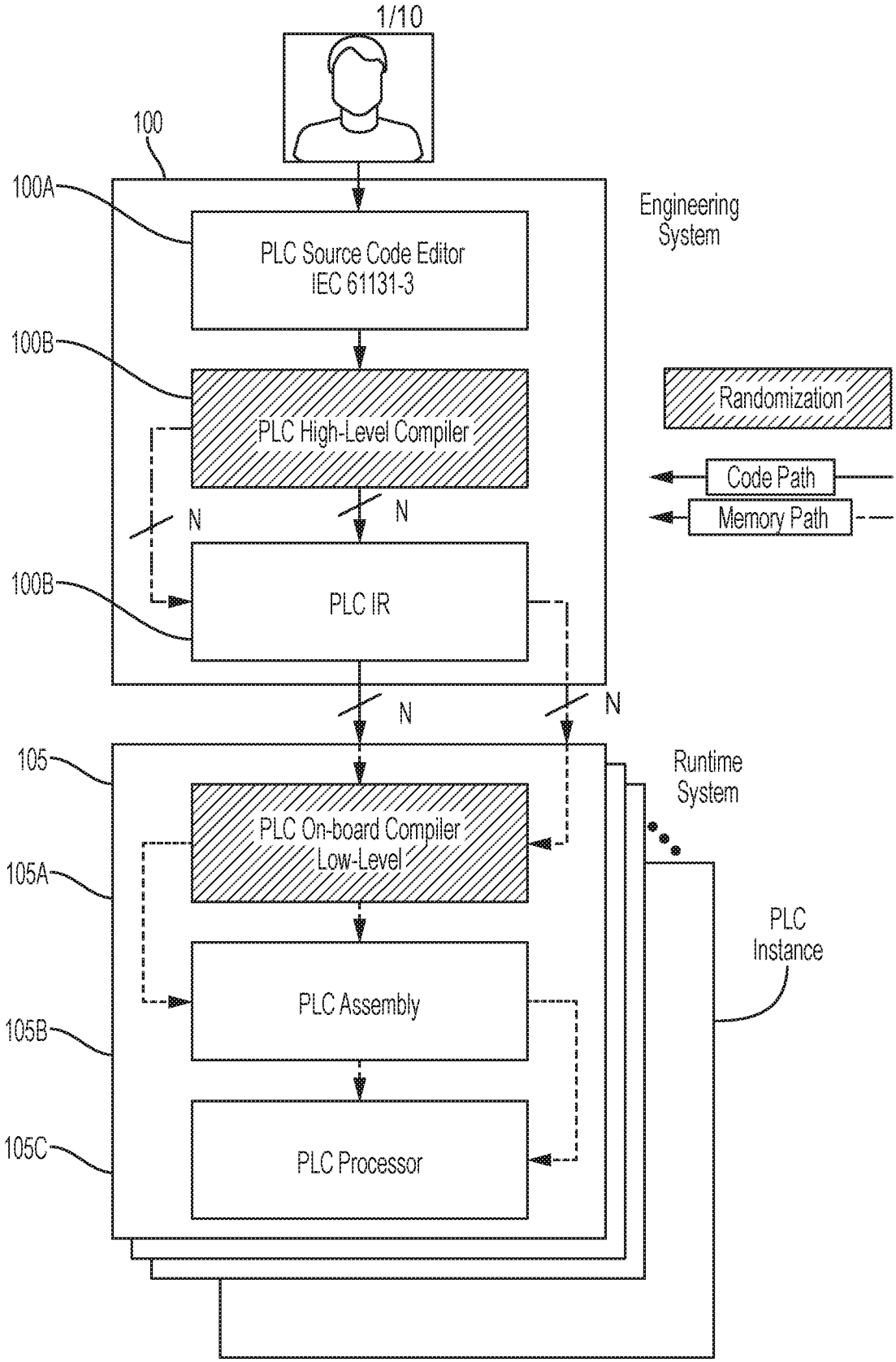
1/10
100
100A
PLC Source Code Editor
IEC 61131-3
Engineering
System
100B
PLC High-Level Compiler
Randomization
Code Path
Memory Path
N
N
100B
PLC IR
N
N
105
Runtime
System
PLC On-board Compiler
Low-Level
105A
PLC
Instance
PLC Assembly
105B
PLC Processor
105C

Fig. 1

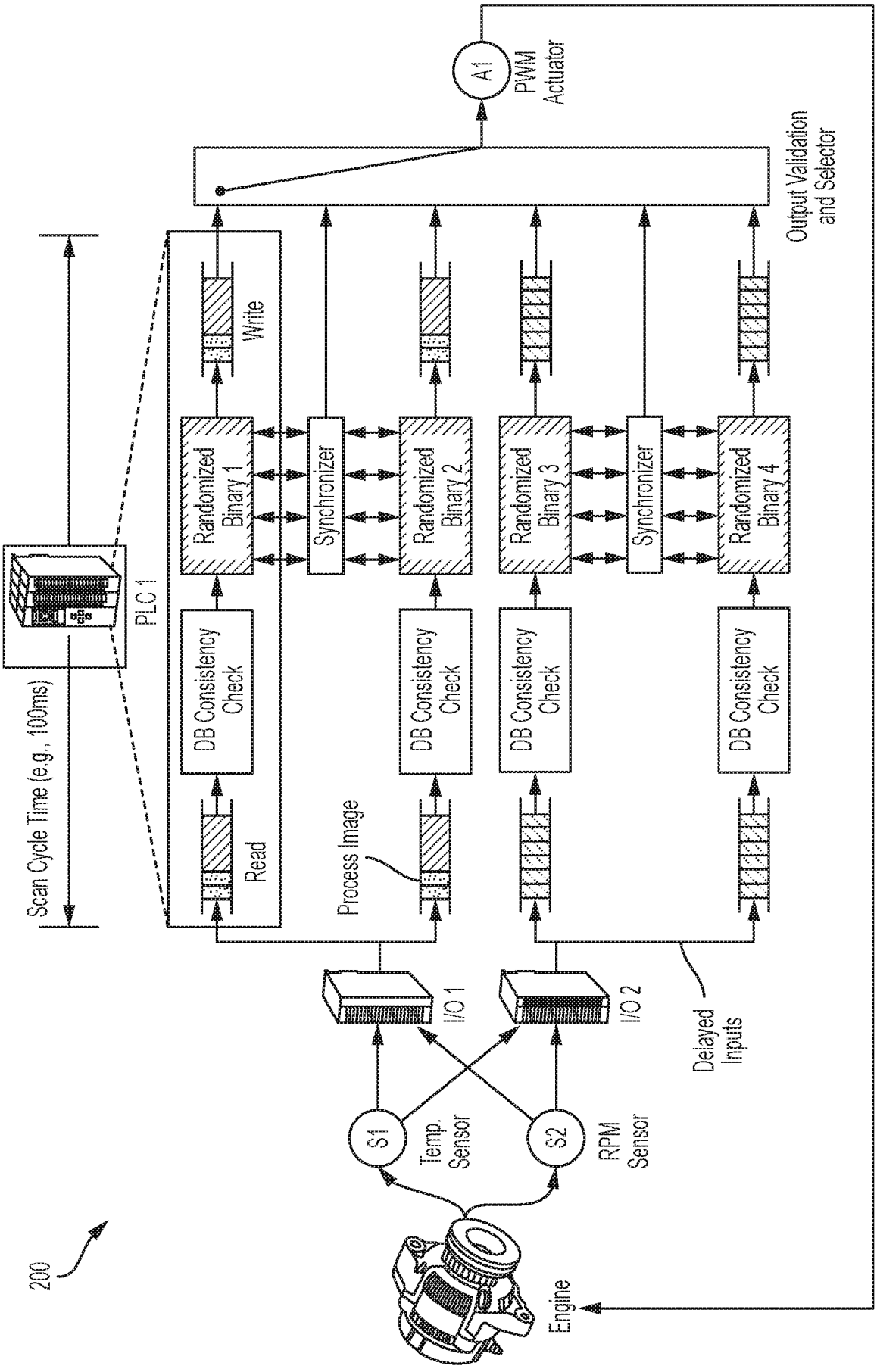
200
Engine
S1
Temp. Sensor
S2
RPM Sensor
I/O 1
I/O 2
Delayed Inputs
Process Image
Read
Write
PLC 1
Scan Cycle Time (e.g., 100ms)
DB Consistency Check
DB Consistency Check
DB Consistency Check
DB Consistency Check
Randomized Binary 1
Synchronizer
Randomized Binary 2
Randomized Binary 3
Synchronizer
Randomized Binary 4
Output Validation and Selector
A1
PWM Actuator

Fig. 2

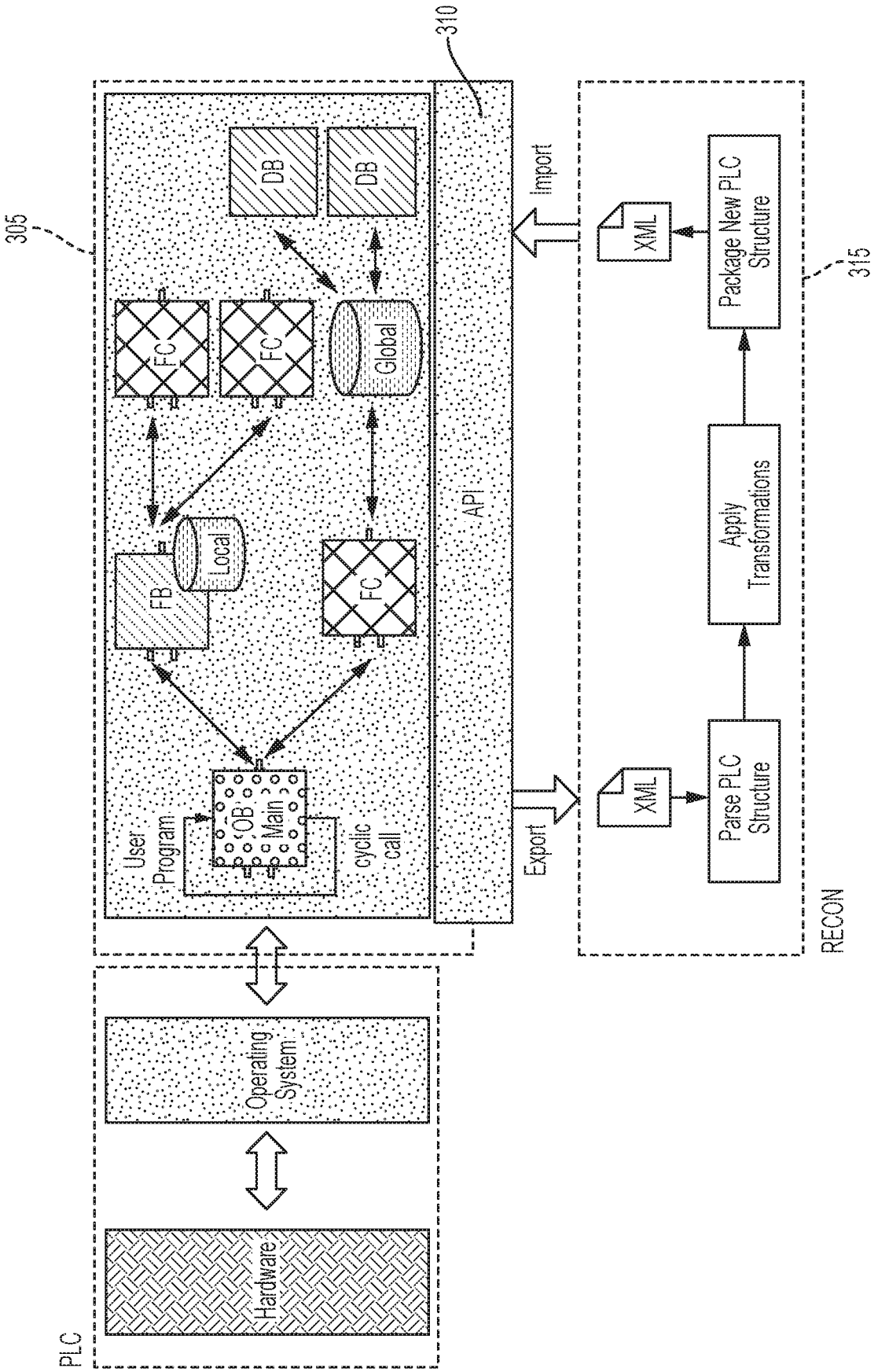
310
305
315
DB
DB
FC
FC
Global
Local
FB
FC
OB
Main
cyclic call
User Program
API
Import
Export
XML
XML
Package New PLC Structure
Apply Transformations
Parse PLC Structure
RECON
Operating System
Hardware
PLC

Fig. 3

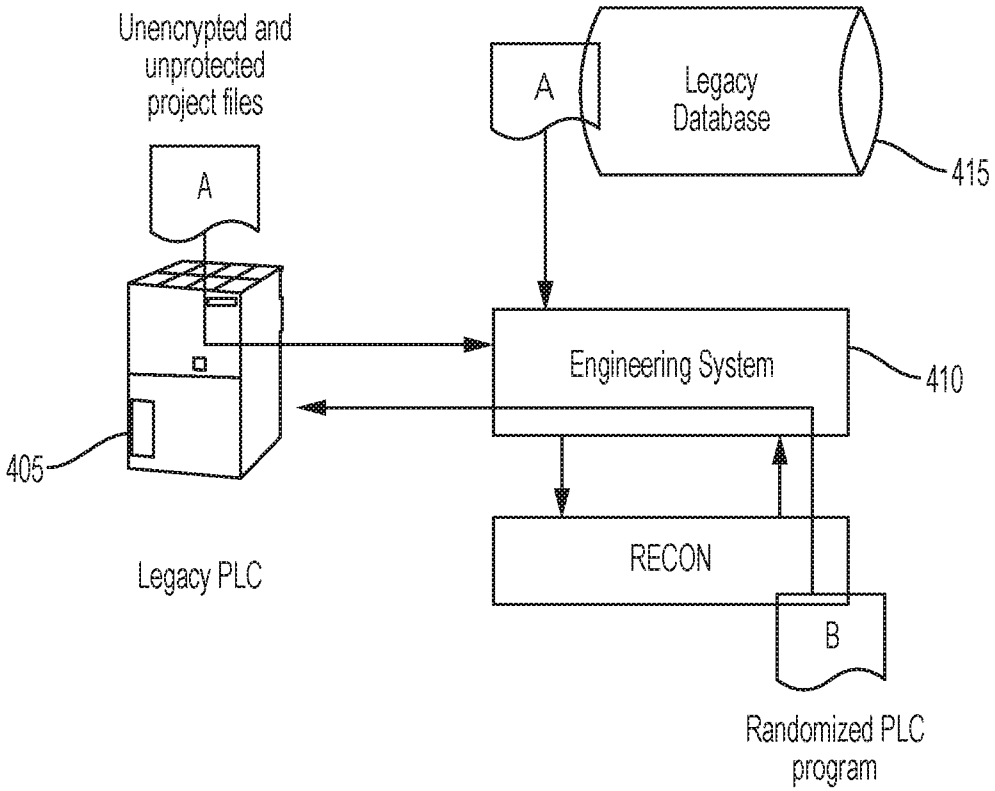
Unencrypted and unprotected project files
A
Legacy PLC
405
A
Legacy Database
415
Engineering System
410
RECON
B
Randomized PLC program

Fig. 4

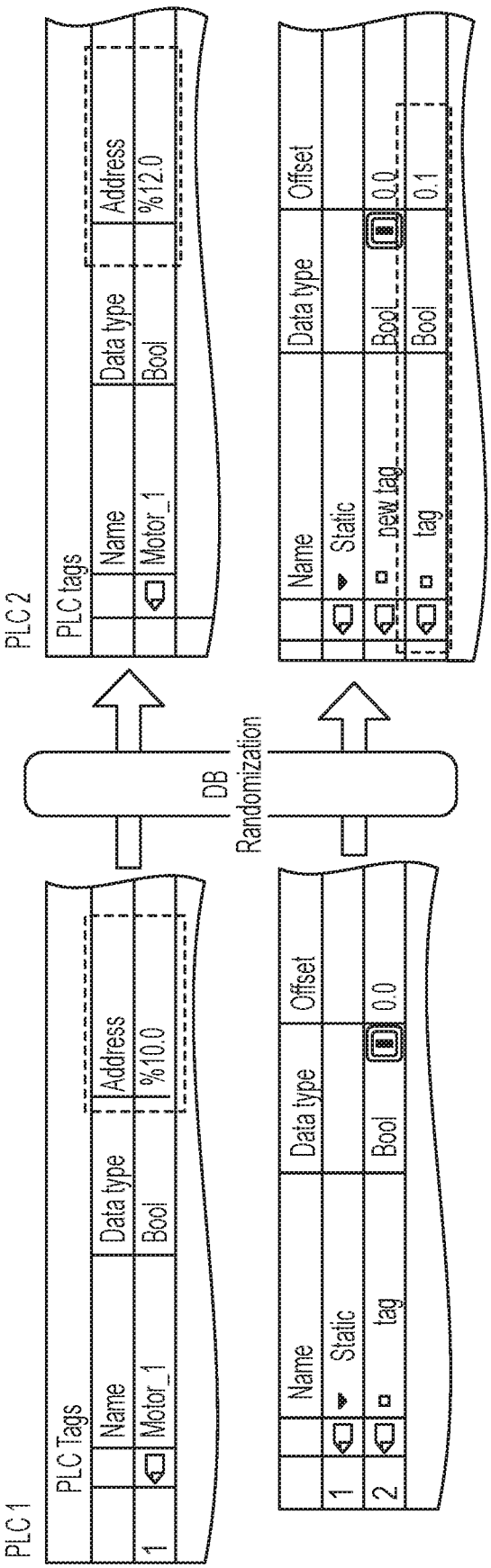
PLC 1
PLC Tags
Name
Data type
Address
1
Motor_1
Bool
%10.0
Name
Data type
Offset
1
Static
2
tag
Bool
0.0
DB
Randomization
PLC 2
PLC tags
Name
Data type
Address
Motor_1
Bool
%12.0
Name
Data type
Offset
Static
new tag
Bool
0.0
tag
Bool
0.1

Fig. 5

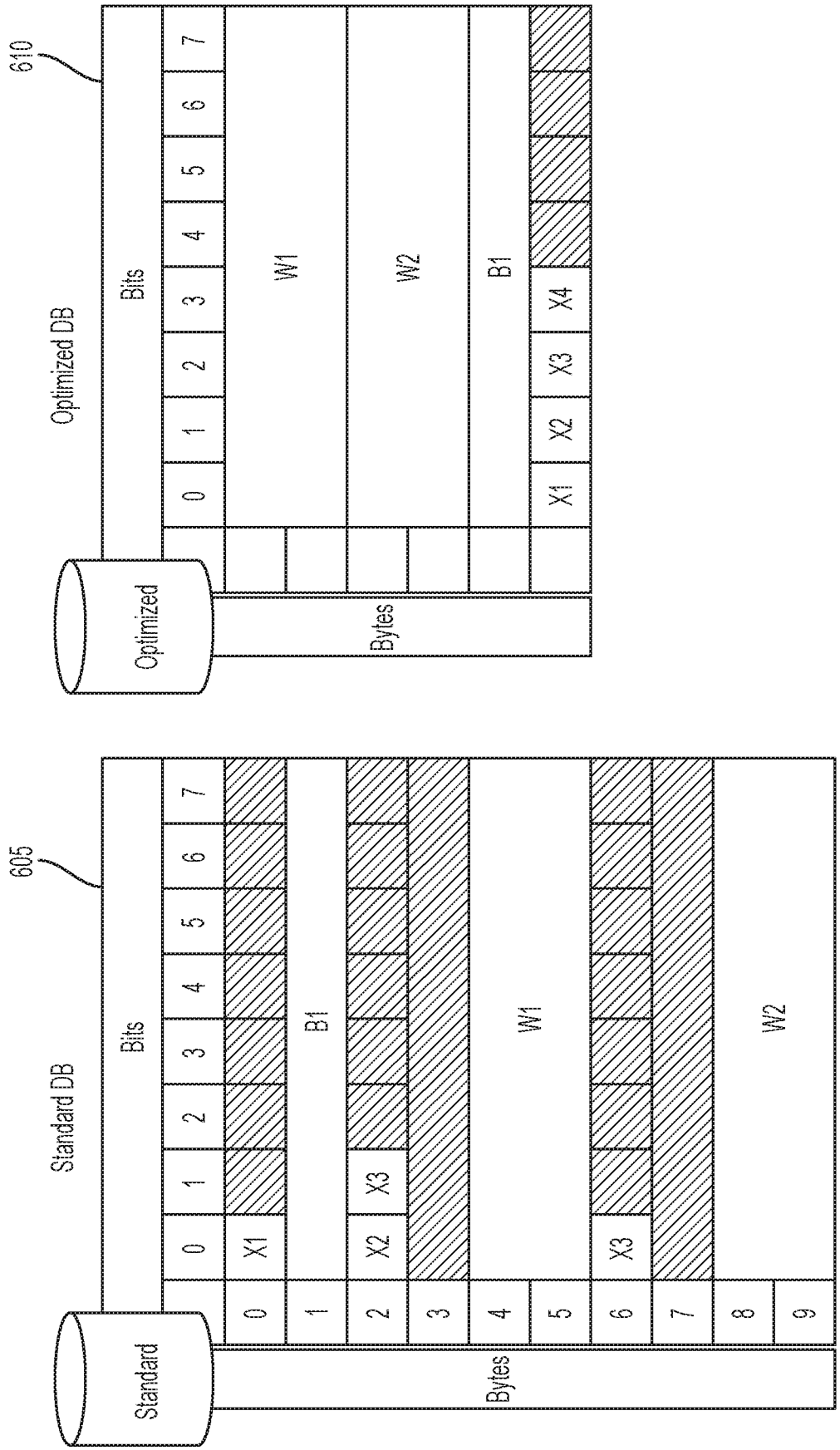
610
Optimized DB
Bits
0
1
2
3
4
5
6
7
W1
W2
B1
X1
X2
X3
X4
Optimized
Bytes
605
Standard DB
Bits
0
1
2
3
4
5
6
7
B1
W1
W2
X1
X2
X3
X3
Standard
Bytes
0
1
2
3
4
5
6
7
8
9

Fig. 6

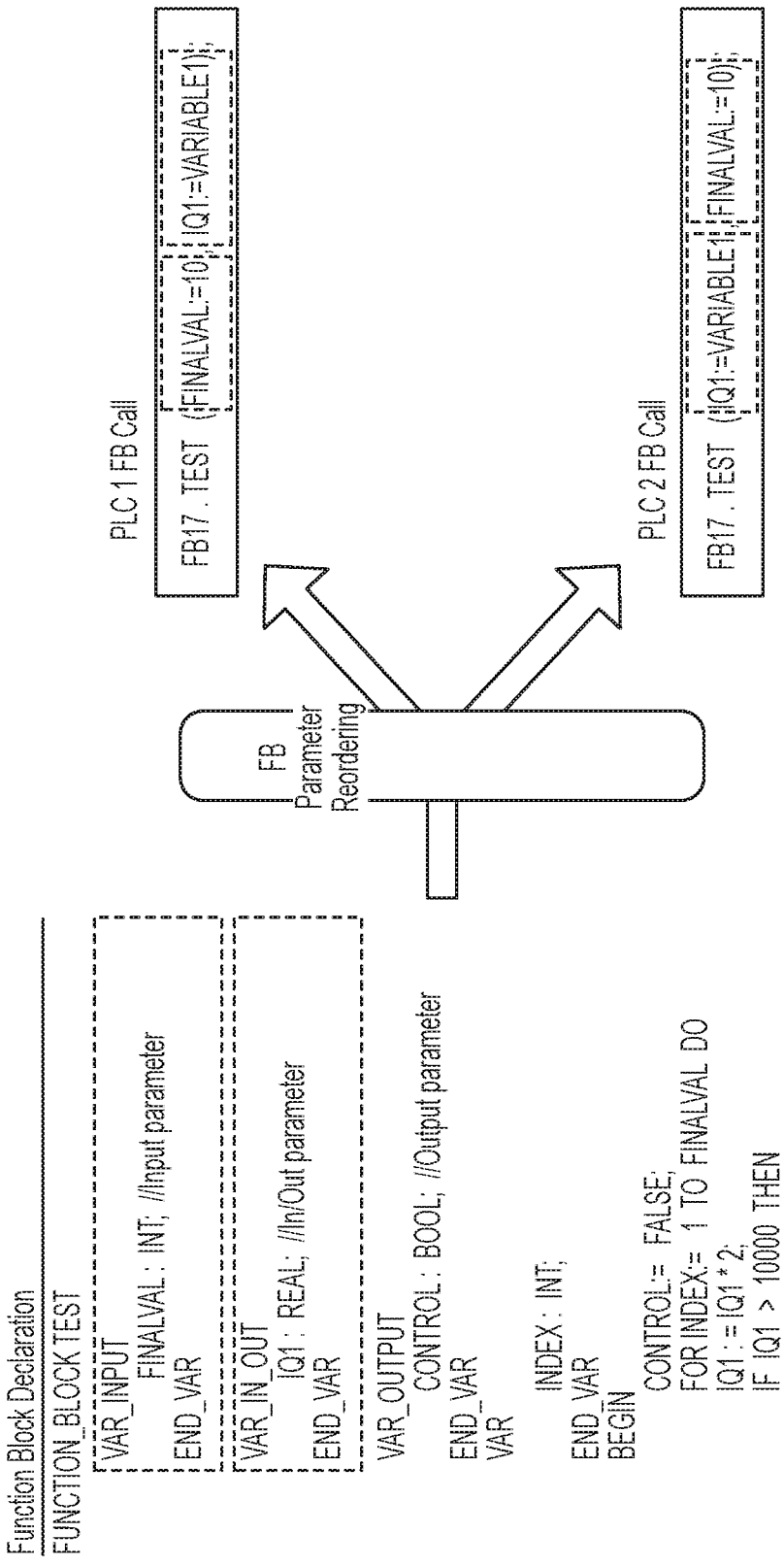
Function Block Declaration
FUNCTION_BLOCK TEST
VAR_INPUT
FINALVAL : INT; //Input parameter
END_VAR
VAR_IN_OUT
IQ1 : REAL; //In/Out parameter
END_VAR
VAR_OUTPUT
CONTROL : BOOL; //Output parameter
END_VAR
VAR
INDEX : INT;
END_VAR
BEGIN
CONTROL:= FALSE;
FOR INDEX:= 1 TO FINALVAL DO
IQ1 : = IQ1 * 2;
IF IQ1 > 10000 THEN
CONTROL:= TRUE;
END_IF;
END_FOR;
END_FUNCTION_BLOCK
FB Parameter Reordering
PLC 1 FB Call
FB17 . TEST (FINALVAL:=10, IQ1:=VARIABLE1);
PLC 2 FB Call
FB17 . TEST (IQ1:=VARIABLE1, FINALVAL:=10);

Fig. 7

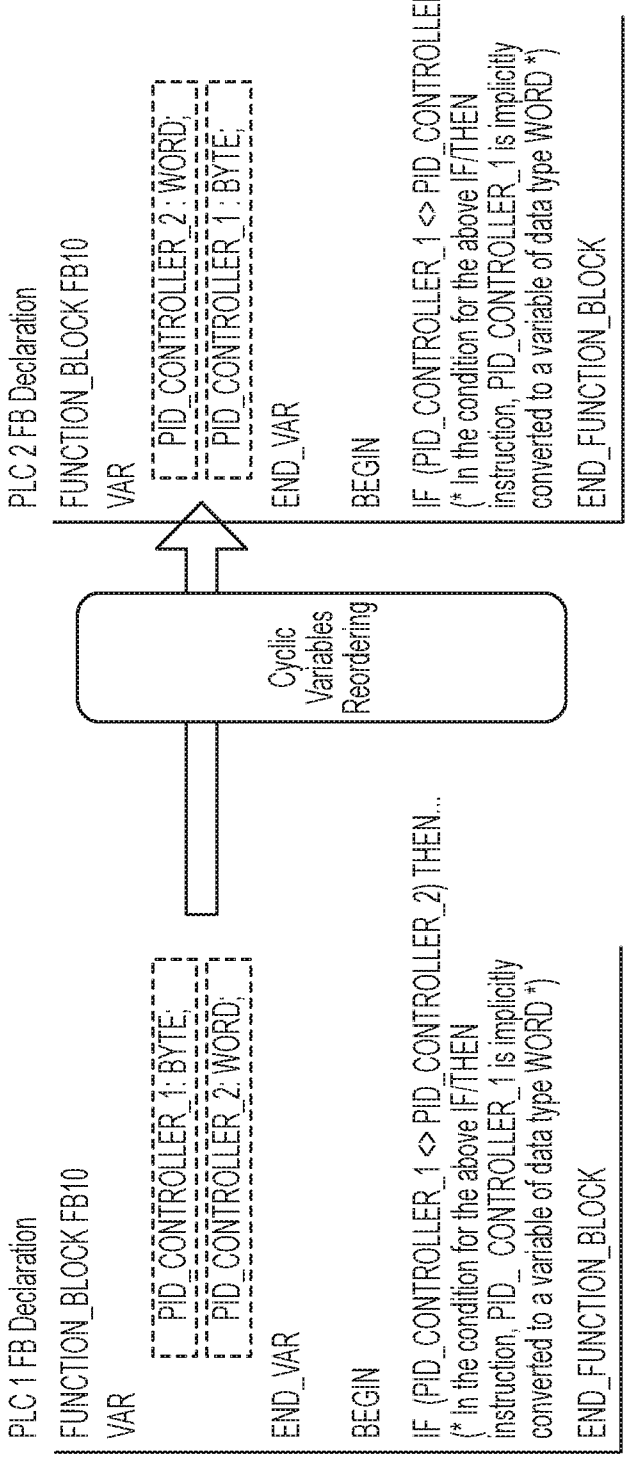
PLC 1 FB Declaration
FUNCTION_BLOCK FB10
VAR
PID_CONTROLLER_1: BYTE;
PID_CONTROLLER_2: WORD;
END_VAR
BEGIN
IF (PID_CONTROLLER_1 <> PID_CONTROLLER_2) THEN...
(* In the condition for the above IF/THEN
instruction, PID_ CONTROLLER_1 is implicitly
converted to a variable of data type WORD *)
END_FUNCTION_BLOCK
Cyclic
Variables
Reordering
PLC 2 FB Declaration
FUNCTION_BLOCK FB10
VAR
PID_CONTROLLER_2: WORD;
PID_CONTROLLER_1: BYTE;
END_VAR
BEGIN
IF (PID_CONTROLLER_1 <> PID_CONTROLLER_2) THEN...
(* In the condition for the above IF/THEN
instruction, PID_CONTROLLER_1 is implicitly
converted to a variable of data type WORD *)
END_FUNCTION_BLOCK

Fig. 8

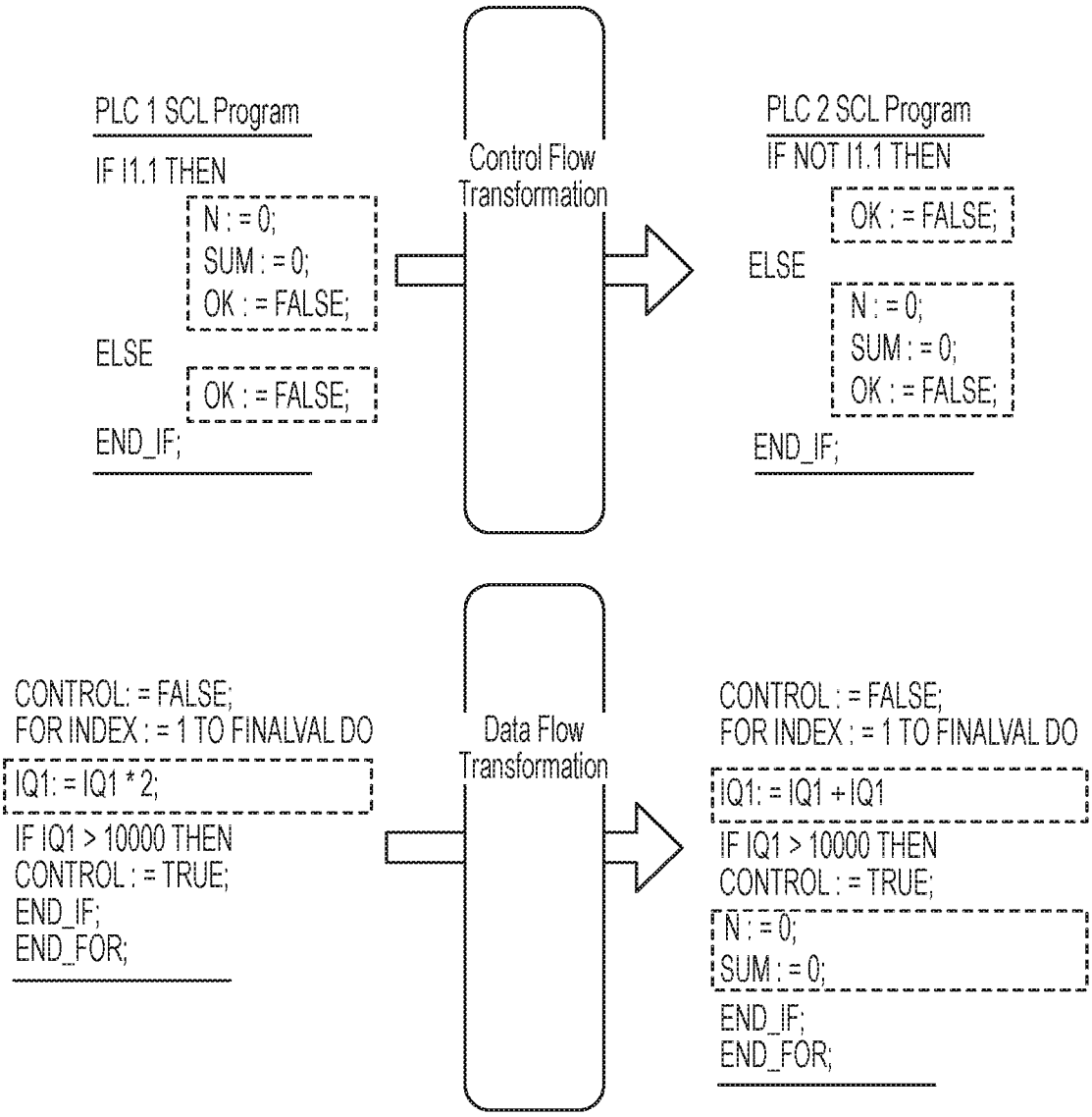
PLC 1 SCL Program
IF I1.1 THEN
N : = 0;
SUM : = 0;
OK : = FALSE;
ELSE
OK : = FALSE;
END_IF;
Control Flow Transformation
PLC 2 SCL Program
IF NOT I1.1 THEN
OK : = FALSE;
ELSE
N : = 0;
SUM : = 0;
OK : = FALSE;
END_IF;
CONTROL: = FALSE;
FOR INDEX : = 1 TO FINALVAL DO
IQ1: = IQ1 * 2;
IF IQ1 > 10000 THEN
CONTROL : = TRUE;
END_IF;
END_FOR;
Data Flow Transformation
CONTROL : = FALSE;
FOR INDEX : = 1 TO FINALVAL DO
IQ1: = IQ1 + IQ1
IF IQ1 > 10000 THEN
CONTROL : = TRUE;
N : = 0;
SUM : = 0;
END_IF;
END_FOR;

Fig. 9

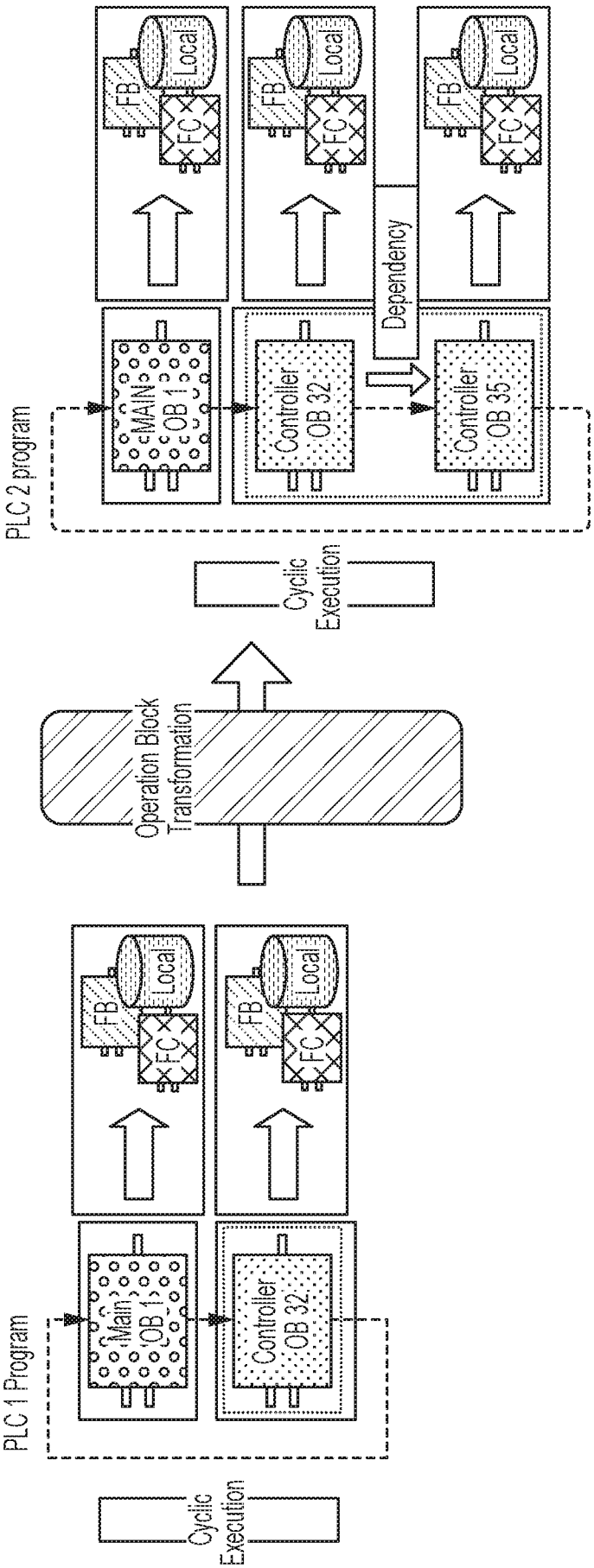
PLC 1 Program
Main
OB 1
Controller
OB 32
FB
FC
Local
Cyclic Execution
Operation Block Transformation
PLC 2 program
MAIN
OB 1
Controller
OB 32
Controller
OB 35
Dependency
Cyclic Execution

Fig. 10

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2016/066295

A. CLASSIFICATION OF SUBJECT MATTER
INV. G05B19/05 G06F21/12
ADD.

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 G05B

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 practicable, 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	EP 2 169 547 A1 (ICS TRIPLEX ISAGRAF INC [CA]) 31 March 2010 (2010-03-31) figures 1,7,10 paragraph [0055] paragraph [0056] -----	1-21
A	SCHWARTZ MOSES ET AL: "Emerging Techniques for Field Device Security", 1 November 2014 (2014-11-01), SECURITY & PRIVACY, IEEE, IEEE SERVICE CENTER, LOS ALAMITOS, CA, US, PAGE(S) 24 - 31, XP011570172, ISSN: 1540-7993 [retrieved on 2015-01-12] page 25 page 29 - page 30 ----- -/--	1-21

[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 application or patent 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
24 April 2017	16/05/2017

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, Fax: (+31-70) 340-3016	Antonopoulos, A

2

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	MICHAEL FRANZ: "E unibus pluram", NEW SECURITY PARADIGMS, ACM, 2 PENN PLAZA, SUITE 701 NEW YORK NY 10121-0701 USA, 21 September 2010 (2010-09-21), pages 7-16, XP058312861, DOI: 10.1145/1900546.1900550 ISBN: 978-1-4503-0415-3 page 8 figure 3 -----	1-21
A	Dimitrios Hristu-Varsakelis ET AL: "Handbook of Networked and Embedded Control Systems" In: "Handbook of Networked and Embedded Control Systems", 1 January 2005 (2005-01-01), Birkhäuser Boston, Boston, MA, XP055363099, ISBN: 978-0-8176-4404-8 pages 259-278, page 268 -----	1-21
A	FORREST S ET AL: "Building diverse computer systems", OPERATING SYSTEMS, 1997., THE SIXTH WORKSHOP ON HOT TOPICS IN CAPE COD, MA, USA 5-6 MAY 1997, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, US, 5 May 1997 (1997-05-05), pages 67-72, XP010226847, DOI: 10.1109/HOTOS.1997.595185 ISBN: 978-0-8186-7834-9 page 68 page 69 -----	1-21
A	Siemens Ag Siemens Ag: "Programming Guideline for S7-1200/1500", , 1 March 2014 (2014-03-01), XP055362727, Retrieved from the Internet: URL:https://www.industry.siemens.nl/automation/nl/nl/industriele-automatisering/industrial-automation/simatic-controller/modulaire-controllers/simatic-s7-1500/Documents/81318674_Programming_guideline_DOKU_v12_en.pdf [retrieved on 2017-04-06] page 10 ----- -/--	6

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	Anonymous: "Address space layout randomization - Wikipedia", , 2 December 2015 (2015-12-02), XP055363110, Retrieved from the Internet: URL:https://en.wikipedia.org/w/index.php?title=Address_space_layout_randomization&oldid=693416768 [retrieved on 2017-04-07] the whole document -----	3
A	ANDREA HÖLLER ET AL: "Patterns for automated software diversity to support security and reliability", PATTERN LANGUAGES OF PROGRAMS, ACM, 2 PENN PLAZA, SUITE 701 NEW YORK NY 10121-0701 USA, 8 July 2015 (2015-07-08), pages 1-13, XP058079545, DOI: 10.1145/2855321.2855360 ISBN: 978-1-4503-3847-9 page 7 -----	1-21

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2016/066295

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☒ As all searchable claims could be searched without effort justifying an additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. claims: 1-21

A method and a system for the development of programmable logic controller (PLC) programs by using PLC program randomization technique.

1.1. claims: 15-20

A method for randomizing software for a programmable logic controller (PLC) during program execution

---

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2016/066295

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 2169547 A1	31-03-2010	CN 101763280 A	30-06-2010
		EP 2169547 A1	31-03-2010
		US 2010083223 A1	01-04-2010