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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 April 2023 (20.04.2023)

WIPO | PCT

(10) International Publication Number
WO 2023/062221 A1

(51) International Patent Classification:
G06F 8/41 (2018.01)

(21) International Application Number:
PCT/EP2022/078722

(22) International Filing Date:
14 October 2022 (14.10.2022)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10 2021 126 848.6
15 October 2021 (15.10.2021) DE

(71) Applicant: **VOLKSWAGEN AKTIENGESELLSCHAFT** [DE/DE]; Berliner Ring 2, 38440 Wolfsburg (DE).

(72) Inventor: **MOELLER, Jens-Dietrich**; Friedrich-Wilhelm-Straße 41, 38302 Wolfenbüttel (DE).

(74) Agent: **HOFSTETTER, SCHURACK & PARTNER PATENT- UND RECHTSANWALTSKANZLEI, PARTG MBB**; Balanstr. 57, 81541 München (DE).

(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, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, 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, ST, SV, SY, TH,

(54) **Title:** METHOD AND SYSTEM FOR STRIPPING DOWN A SOFTWARE TO MINIMIZE ITS CODE FOR OPERATING DEVICE FEATURES OF A DEVICE

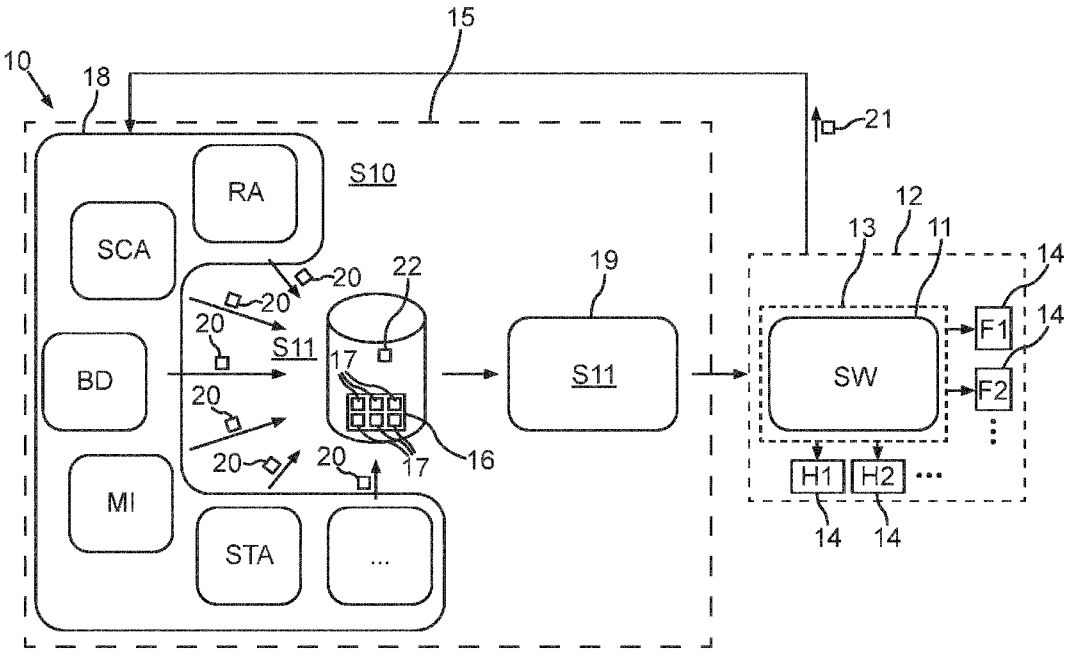

Fig.1

(57) **Abstract:** The invention is concerned with a method for stripping down a software code (16) of a software while still being able to operate a predefined feature set of a device (12), wherein device feature data (21) of the device lists several device features (14) that are to be operated by the software by executing the software on at least one microprocessor (13), wherein the software code (16) of the software comprises of several function blocks (17) for providing a respective functionality in the device and each function block (17) is listed in software configuration data (20). The method comprises that a binary code (11) of the software is generated by a compilation procedure (19), wherein the compilation procedure (19) comprises that only those of the function blocks (17) that are marked in the software configuration data (20) as being required functions blocks (17) for operating the device features (14) are actually selected and the binary code (11) is compiled from the selected function blocks (17) only.

WO 2023/062221 A1

[Continued on next page]

TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, 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, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

Description

Method and system for stripping down a software to minimize its code for operating device features of a device

The invention is concerned with a method and a system for reducing or stripping down a software code of a software in order to minimize the code while still being able to operate a predefined feature set in a device, e.g. in a vehicle. Device feature data lists the several device features that are to be operated by the software and the software shall have a minimized code footprint, such that a minimum amount of binary code stays unused during the operation of the device.

In automotive technology, vehicles (like passenger vehicles or trucks or busses) nowadays comprise microprocessors having an operating system instead of microcontrollers with monolithic control software. The operating system is often an open source system such as Linux, in particular the Linux distribution “JANTE” ®. However, those operating systems are not only limited to the use on microprocessors for vehicles and therefore comprise additional features that are, however, not needed in a vehicle.

Since any software, including the operating system kernel, in the vehicle has to be tested on safety issues and has to be supervised and maintained, the more code the software comprises, the more code has to be tested, supervised and maintained. Therefore, it is the goal of software engineers to identify those parts of the code of the operating system that are not used and to eliminate those parts of the code. So far, this is done manually, which makes the process very cumbersome.

The same counts for software that is originally written for running on a microcontroller and shall now be transferred to run on a microprocessor. On microcontrollers, it is often an embedded software that is running on a proprietary operating system that is different from chip manufacturer to chip manufacturer. Here, a compiler can be used to optimize the binary code of the software as different function blocks of the software code are linked statically resulting in a well know predefined arrangement of the binary code in the memory.

This is not possible when transferring the functionality into a shared library on microprocessors. Thus, if the software shall be directly transferred to a microprocessor, the software may also comprise code that is not needed. This is not easy to detect as the shared library may be linked dynamically in different contexts. This software therefore also needs an optimization in order to minimize the amount of software code that is finally included in the binary code of a software for a vehicle.

Document US 2008 133 598 A discloses a method for reducing a memory food print of an embedded system. As has already been described, within an embedded system, it is much easier to identify unused code, as an embedded system is running on a monolithic binary code with no dynamic linking. Transferring this method to a system using a microprocessor and dynamic linking is not trivial.

Document US 2012 102 473 A describes a method for reducing the code size of compiled code by taking advantage of the specific structure of object oriented code where abstract method definitions may be deleted without rendering the remaining code inoperable.

Document US 2015 234 652 A describes a software for an internet server that determines automatically which parts of the software have been accessed by clients. Those parts of the software that are not accessed by any client within a certain time period are deleted from the server. This statistical approach comes with the risk that code may be deleted that might become relevant in a very rare specific case that may have been overlooked.

Document US 10956138 B describes a method for reducing memory usage of a software that is based on a scripting language. This solution requires the programming of functionalities based on scripts instead of binary code.

Document US 9436449 B describes a method for removing portions of a software by means of a trial and error method in that code is disabled and a behavior of the resulting software is compared to a desired behavior. If the resulting behavior matches the desired behavior, the code is removed, otherwise the code is re-enabled. This method makes it difficult to determine non-required code in a deterministic straight forward way and code might remain in the software that is not used at all but that has not been identified in the trial and error phase.

Document US 2009 327 639 A discloses a method for identifying unneeded software features by analyzing the memory data from a memory of an embedded system. This method relies on the fact that in embedded systems static linking is used such that unneeded software features can be traced easily from a memory data dump of a running embedded system. This is not possible in a microprocessor system that is using an operating system and dynamic linking, as software features are located in different regions of the memory every time the system is restarted.

It is an objective of the present invention to provide a tool for reducing the amount of code that is included in the binary version of a software given a specific software source code that comprises a large amount of function blocks.

The objective is accomplished by the subject matter of the independent claims. Advantageous developments with convenient and non-trivial further embodiments of the invention are specified in the following description, the dependent claims and the figures.

One solution comprises a method for stripping down or pruning a software to reduce or minimize the software code of the software for operating predefined device features of a device, e.g. a vehicle. Device feature data provides a list of the several device features (i.e. a feature set) that are to be operated by the software. For operating these device features, binary code of the software will be executed on at least one microprocessor (e.g. of the device). In particular on the at least one microprocessor, at least one operating system will be running together with software libraries for dynamic linking. A software code of the software (like e.g. source code and/or libraries) comprises of several build blocks or function blocks for providing different functionalities in the vehicle. For example, such a function block could provide the functionality of converting raw image data of a camera of the device into an MPEG video data stream (MPEG – moving picture expert group). Another function block might enable or implement a printing functionality. Each function block is listed in software configuration data, i.e. the software configuration data provide an overview or classification of the available function blocks in the software code.

Thus, there is provided device feature data that is listing the device features and there is software configuration data that is listing the available function blocks.

The method comprises that a processing unit performs the following steps:

- Determining a respective demand, that each of the device features has with regard to each of the function blocks. This may be performed by a predefined code customization procedure that may be performed by the processing unit, i.e. the corresponding instructions may be provided by the code customization procedure. The demand indicates that the respective function block is required to operate the respective device feature. For example, a function block providing a specific graphics driver routine might be needed to run a specific display of the device. In this case the display is a device feature of the device.
- Marking or flagging, in the software configuration data, each function block for which a respective demand by at least one device feature is determined. In other words, each marked or flagged function block is a required function block that is required to operate or run at least one device feature.

- Generating a binary code of the software for operating the device. The binary code is a machine language version (e.g. X86 instructions or RISC instructions) of the software for execution by the device processing hardware. This "device processing hardware" comprises the entire set of available microprocessors, i.e. the network of electronic control units and/or the central computer which in total form the device electronic system. The binary code is generated by a compilation procedure that comprises that only those of the function blocks are selected that are marked in the software configuration data as being required functions blocks. The binary code is then compiled and/or linked from these selected function blocks. In particular, the binary code is generated from only the selected ones of the function blocks. The remaining function blocks are not required and are therefore excluded from the binary code.

In other words, a system that is performing the described method will generate binary code for the at least one microprocessor in the device electronic system by performing the code customization procedure and the compilation procedure. The method is based on the list of all available function blocks, i.e. the software configuration data on one side and on the other side the device feature data listing all device features that need to be controlled or operated by the resulting binary code of the software. The code customization procedure matches the feature set and the software configuration such that only required function blocks are marked or identified or flagged in the software configuration data. One way of flagging can also be that the list entries of the non-required function blocks are deleted from the software configuration data. Another way of flagging can be that the required or the non-required function blocks are marked by setting a "select bit" or an "un-select bit" in the software configuration data.

Then, the compilation procedure selects only the required function blocks, i.e. the corresponding source code, for compiling and linking which will result in binary code for at least one operating system and/or binary code for at least one dynamic library or shared object library and/or at least one application software that is to executed by the at least one operation system and is based on at least one dynamic library and/or shared object library. A dynamic library is a binary code module that an application program is linked to at runtime (e.g. at start-up), like is known from the prior art. The corresponding software for this dynamic linking is the well-known dynamic linker.

The method provides the advantage that no software parts or function blocks of the software code need to be deleted, as the software configuration data identifies the required function blocks, and the compilation procedure will actively select only these required function blocks. In other words, the software code may be updated, for example, a new version may be

provided, and this will not require a new adaptation of the software code as the software configuration data still hold the information which of the function blocks shall be selected from the updated software code. For example, if the software code comprises the code of an operating system, for example, a Linux operating system, an update of this software code of the operating system may be provided, and no adaption of this general operating system code is necessary as the compilation procedure knows which function blocks of the operating system need to be selected based on the software configuration data. This allows to efficiently handle software code that forms the basis for a binary code of a software for a device. The original or initial software code can be stripped down or pruned to a reduced amount or even a minimum amount that is required to operate the device features.

The binary code may then be installed in the respective device, e.g. a motor vehicle, where at least one microprocessor of the device may execute the binary code such that the device features, e.g. a human machine interface (HMI) and/or an electronic control unit and/or a display, may be operated or controlled by the at least one microprocessor according to the binary code.

The invention also comprises further developments that provide features which afford additional technical advantages.

One development comprises that the code customization procedure comprises:

- generating a preliminary version of the binary code using all function blocks or using a preliminary version of the software configuration data for selecting function blocks and
- executing the preliminary version of the binary code in a runtime analysis environment and
- generating logging data of observed system calls and/or functions calls and/or program counter states using a tracing functionality of the runtime analysis environment and
- adapting the software configuration data by marking those functions blocks that have been used according to the logging data and/or by unmarking those functions blocks that stayed unused according to the logging data.

This provides the advantage that the code customization procedure may determine automatically or by itself which function blocks of the software code are necessary for generating the binary code. The code customization procedure may start with a full version or with the whole software code, i.e., all function blocks may be used for generating the preliminary version of the binary code. This will result in a binary code that will also comprise the binary equivalent of the function blocks that are not necessary or not required in the

software for the device. However, by observing the interactions or signal exchange between, for example, applications on one side and operating systems on the other side (i.e., the system calls) and/or the interactions between single function routines in an application and/or an operating system (i.e., the function calls) using the tracing functionality of a runtime analysis environment, those parts of the binary code that are not accessed throughout the operation or execution of the software binary code, can be identified as function blocks that are not required. Using the program counter states for identifying those regions in the memory that are not covered or accessed by the program counter of a microprocessor even allows to identify single program steps that are not used or that are not active in the software code. For example, if a certain IF-condition distinguishes between two specific states (state_1, state_2) of the device (in pseudo-code: IF state_1 is true THEN execute code for state_1, ELSE execute code for state_2 ENDIF) and one of the states is never reached, then the code for the state that is never occupied by the device can be identified as a non-required function block. Thus, by observing system calls, function calls and/or program counter states, a different granularity for observing non-required parts of the binary code can be provided. An appropriate runtime analysis environment can be taken from the prior art, for example, based on a compiled version of the binary code that additionally provides so-called debugging symbols.

One development comprises that the code customization procedure comprises:

- matching memory addresses that where accessed according to the logging data with memory addresses associated with a respective function block and
- marking those function blocks as required that have been accessed by the program counted according to the comparison of memory addresses.

Within a runtime analysis environment, the so-called “code coverage” can be determined, i.e., even if a dynamic library is used, during runtime the position of this dynamic library in memory is known. Thus, by observing the content of the program counter of a microprocessor, it can be determined, which code is executed by the microprocessor system. The binary code can be compared using debug symbols such that it can be identified which memory address belongs to which function block of the software code. A memory region or a segment of the memory that is not accessed by the program counter can then be identified as belonging to a non-required function block and the corresponding marking in the software configuration data can be done. Observing the program counter provides the most detailed analysis of unused code.

The code customization procedure not only needs to analyze the running binary code. Alternatively or additionally, the given source code, i.e. the software code providing all the function blocks, can also be analyzed.

One development comprises that the code customization procedure comprises

- performing a static code analysis is performed by generating a dependency graph of software functions of the software code and
- identifying those functions blocks that are not contained in the dependency graph and
- setting those function blocks in an unmarked state in the software configuration data.

This makes use of the fact that function blocks comprising function routines are identified by a function name or function header that is used for calling or accessing the respective function routine from another part of the software code. Thus, if a certain function header or function name is not mentioned or accessed from anywhere in the software code, this function routine obviously is not necessary or needed in the binary code. This can be identified by generating a dependency graph that identifies the function calls in the source code starting from the main routine or start routine of each application and/or operating system.

One development comprises that the code customization procedure comprises

- storing a dependency list of build dependencies during at least one preliminary execution of the compilation procedure and
- marking those the function blocks in the software configuration data, that are mentioned as part of the software build of the binary code in the dependency list.

This can make use of the tools that are also used in the compilation procedure, i.e. a pre-compiler, a compiler and/or a linker. These tools also provide information about the function blocks in the software code that are actually needed starting from the main program or main function or entry point of the software. In other words, not the unused or non-required function blocks are identified, but a positive list of all those function blocks that are included in the dependency list is identified and marked in the software configuration data. Using the built dependency provides the advantage that all those function blocks are considered that actually contribute to the binary code.

One development comprises that the code customization procedure comprises that

- receiving a manual input from a user by a user interface and
- marking those function blocks in the software configuration data that are indicated as necessary according to the manual input.

In other words, at least one user, preferably several users (preferably several users from several different development departments) request or reserve those function blocks that they plan to include in their application software code. Thus, even experimental inclusions of function blocks that would normally not be considered by any automatic algorithm can still be included by providing the corresponding manual input. For example, a data base with all the manually marked function blocks can be stored such that any user or developer that is participating in the development of the software for the device can mark or identify function blocks that the user or developer is planning to include or rely upon when developing an application. The user interface can be provided, for example, based on a network browser and/or a database access GUI (graphical user interface).

One development comprises that the code customization procedure comprises that a static target analysis is defined by the steps of:

- providing an association list associating possible device features of the device with necessary function blocks that are needed to operate the respective device feature,
- identifying device features comprised in the device according to the device feature data and identifying the respective associated necessary function blocks in the association list,
- marking those function blocks in the software configuration data that are necessary according to the identified device features.

By analyzing or looking at the actual target, for example, a specific electronic control unit and/or a specific sensor and/or a specific actuator, for example, their technical specification data, it can be determined which function blocks may be needed, for example, a specific temperature sensor and/or a specific communication protocol and/or a specific power supply unit. While specific binary code might not be accessed or executed during a test phase of the software, for example, in a simulated environment and/or a test platform, a specific binary code might become necessary once the software is installed in the device itself such that software code for the specific device feature, for example, the power supply unit and/or temperature sensor, might be become relevant or necessary. As this might not be detectable during a test phase using the above-described runtime analysis environment, using the association list associating the actually available hardware device features and/or software-based functionalities that will be available or present in the device itself, with the necessary function blocks, it can be prevented that such a function block is overlooked or left out when generating the binary code.

One development comprises that software configuration data is adapted iteratively in that the customization procedure is repeated at least once and for each repetition, the marked

function blocks from the respective software configuration data resulting from the previous iteration are provided in the customization procedure as the set of function blocks that need to be compiled and/or linked by the compilation procedure. This iterative approach has proven advantageous, as removing or unmarking a specific function block may have an effect on the overall dependency graph or the overall dependencies of function blocks. For example, if one function block is marked as non-required or is removed from the software customization data, any other function block that has so far been exclusively used only by this now-removed function block, will be identified as also being non-required or unnecessary in the next iteration.

One development comprises that in the device feature data at least of the following types of device features is listed:

- hardware components of the device,
- software-based functionalities of the device.

In the case of a device that is a vehicle, exemplary hardware components of a vehicle can be: an electronic control unit, a sensor, an actuator, a display of a specific display type, a brake control system. Exemplary software-based functionality of a vehicle can be: a controller for an electric engine, a body control system, a driver assistance system, an autonomous driving function. The corresponding function blocks for hardware components may be designed for controlling and/or accessing at least one signal of the hardware component.

One development comprises that a driver software module comprised in the software code is removed by unmarking it in the software configuration data, if the corresponding hardware component to which the driver software is dedicated, is non-existed in the device. It has proven that this is a easily implemented way of reducing the size of the software code as a software driver can be directly associated with a specific hardware component, for example, a specific graphic chip and/or signal processing chip and/or a specific data bus. If such a hardware component is not present in the device, the corresponding driver software is not required in the binary code for the device.

One development comprises that in the software configuration data at least one of the following types of function blocks is listed:

- a whole software functionality comprising several software functions and/or at least one software library (e.g. printing functionality),
- a single function routine (function header and function body in source code) or system call routine,

- at least one single line of code within one of a function routine and a system call routine.

These three possibilities provide different granularity regarding the selection of necessary parts or data of the software code. By identifying whole software functionalities, for example, a printing functionality and/or a data conversion functionality and/or a communication protocol, selecting or marking one such software functionality as a function block provides a way of selecting all those parts of the software code that exclusively relate to this software functionality, and thus removing or ignoring this software functionality provides an effective way of reducing the size of the binary code. A single function routine can be identified by its function header and its function body in the source code. Within a whole software functionality single function routines have proven to be unnecessary with regard to operating the device features, such that such a single function routine or (in the case of an operating system) a system call routine may be removed without breaking or destroying the whole software functionality that comprises this function routine or system call routine. Removing single lines of code can be advantageous, for example, in the case that a conditional execution of such single lines or a set of single lines, for example, as part of an if-statement or a case-statement is not necessary in the device.

One development comprises that the software code comprises

- an operating system kernel and/or
- at least one dynamic library and/or at least one shared object file that is designed to be dynamically linked during execution of the binary code in the device.

In other words, the method may be applied to the software code of an operating system kernel and/or to the software code of a dynamic library/shared object library or shared object file. Such software code has proven to be difficult to analyze with prior art methods as their arrangement in memory and/or their usage by of numerous or several applications running on the operating system kernel and/or the dynamic linking to these libraries are difficult to identify.

Accordingly, one development comprises that the software code comprises of several different, independent application programs for the device. As different application programs may all run on the same operating system kernel and/or may use the same dynamic libraries/shared object libraries, and/or several different developer teams may develop these application programs, this method provides the specific advantage that the development of these independent application programs may be coordinated such that for all these application programs the underlying operating system kernel and/or dynamic libraries/shared

object libraries are adapted to provide the necessary function blocks for all application programs.

One development comprises that the binary code is transferred to the device and the device is operated by executing the binary code. The advantage is that the binary code can be run or executed with minimal requirement of storage space as the binary code is tailored to the available device features of the device. No “dead” binary code for non-existent device features has to be stored.

In the case that the device that is a vehicle, one development comprises that the software code comprises at least one software function that performs a driving function in the vehicle, if performed by at least one electronic control unit of the vehicle. The method can be used to provide software in the vehicle for the actual driving function of the vehicle, for example, a driver assistance functionality and/or a not-autonomous driving functionality.

One further solution comprises a system for generating a binary code of a software for operating device features in a device, wherein the system comprises a processing unit for performing a code customization procedure and a compilation procedure and a database storage for storing a device feature data and a software configuration data, wherein the system is configured to perform an embodiment of the described method.

The processing unit may comprise a processor circuit adapted to perform the embodiment of the method according to the invention. For this purpose, the processor circuit may comprise at least one microprocessor and/or at least one FPGA (field programmable gate array) and/or at least one DSP (digital signal processor). Furthermore, the processor circuit may comprise program code which comprises computer-readable instructions to perform the embodiment of the method according to the invention when executed by the processor device. The program code may be stored in a data memory of the processor device. The system, especially the processor circuit, may be implemented as a computer or a computer network that is provided for developing the software for the device. It may be provided, for example, in a laboratory a network interconnecting the developers of the software for the device.

The device is preferably designed as vehicle, e.g. a motor vehicle, in particular as a passenger vehicle or a truck, or as a bus or a motorcycle. The device can also be designed as a building control system (e.g. for climatizing the building) or as a compute server system, just to name examples.

The invention also comprises the combinations of the features of the different embodiments.

In the following an exemplary implementation of the invention is described. The figures show:

Fig. 1 a schematic illustration of an embodiment of the inventive system; and

Fig. 2 a schematic illustration of an embodiment of the inventive method.

The embodiment explained in the following is a preferred embodiment of the invention. However, in the embodiment, the described components of the embodiment each represent individual features of the invention which are to be considered independently of each other and which each develop the invention also independently of each other and thereby are also to be regarded as a component of the invention in individual manner or in another than the shown combination. Furthermore, the described embodiment can also be supplemented by further features of the invention already described.

In the figures identical reference signs indicate elements that provide the same function.

Fig. 1 illustrates a system 10 that can be used to provide binary code 11 that may be provided in a device that is designed as a vehicle 12 to run at least one microprocessor 13 in the vehicle 12 such that by the at least one microprocessor 13 based on the binary code 11 at least one device feature 14 may be provided in the vehicle 12. The at least one microprocessor 13 may be provided by one or preferably several different electronic control units and/or at least one central computer of the vehicle 12. As device features 14 hardware components H1, H2 and/or software-based functionalities F1, F2 may be provided, wherein the examples shown for device features 14 are only an exemplary number of device features 14. That can be more or less device features 14.

The binary code 11 can be designed on the basis of system 10 such that an amount of non-required or unused binary code is minimized. The binary code 11 constitutes a software SW for operating or controlling the device features 14.

For generating the binary code 11, the system 10 may comprise a processing unit 15 that may be implemented on the basis of at least one computer and/or processor circuit. For example, the processing unit 15 can be the computer network of a software development laboratory. For generating the binary code 11, the software code 16 of at least one operating system (i.e. at least one operating system kernel or OS-kernel) and/or the source code or at least one dynamic library DL and/or the source code of at least one application APP may be provided, for example, in a database DB. An example of a dynamic library DL is the so-called dynamic link library DLL as it is used in the Microsoft ® operating system or a shared object library (.so) as it is known from the Linux ® operating system.

By the software code 16, function blocks 17 may be defined that each provide a specific functionality that can be included in the binary code 11. Starting from the software code 16, based on a code customization procedure 18, one of several of the function blocks 17, especially not all of the function blocks 17, may be selected and only those selected function blocks 17 of the software code 16 may be used in a compilation procedure 19 for compiling and/or linking the binary code 11.

The code customization procedure 18 may generate software configuration data 20 that indicate which of the function blocks 17 shall be part of the compilation procedure 19 for generating the binary code 11. Possible elements or steps of the compilation procedure 19 comprise: a runtime analysis RA, a static code analysis SCA, a build dependency analysis BD, a reception of manual inputs MI, a static target analysis STA, wherein one or several or all of these steps or even further other analysis steps may be part of the compilation procedure.

For generating the software configuration data 20, i.e. for indicating or marking, which device feature 14 shall be considered or supported by the binary code 11, device feature data 21 can be provided to the code customization procedure 18. The device feature data can indicate, which device feature 14 and/or which development platform is to be supported or considered by the binary code 11.

The resulting software configuration data 20 provide or constitute a rule set 22 for selecting function block 17 from the software code 16 for the compilation procedure 19 that will then compile and/or link the selected function blocks 17 to generate the binary code 11.

Thus, in a step S10, the code customization procedure 18 may identify or determine an amount of the device features for certain function blocks 17, and the code customization procedure 18 may in a step S11 mark in the software configuration data 20 each function block 17 for which a respective demand has been determined. In a step S12, the binary code 11 may be generated by the compilation procedure 19.

Fig. 2 illustrates, how the code customization procedure 18 may identify unnecessary or non-required function block 25 in a preliminary version 26 of the binary code 11. The binary code 11 in its preliminary version 26 may provide an operating system OS providing an operating system kernel. Also, dynamic libraries DL may be provided. Running on the operating system and using dynamic libraries DL, at least one application APP may be executed as part of the software for the vehicle 12. During runtime of this preliminary version 26 of the binary code 11, access to different parts of the memory footprint 27 of the binary code 11 may be

observed, for example, by tracing function calls 28 and/or system calls 29 in the running software. Alternatively or additionally, the content of a at least one program counter 30 of the microprocessor 13 running the binary code 11 may be observed. From these observations, it may be identified that an unused code 31 may exist that is not executed during runtime. This non-executed code 31 can be associated with a function block that consequently can be identified as an non-required function block 25. For example, during the dynamic linking, the dynamical library DL will be loaded into memory contributing to the memory footprint 27 and a dynamic linker can be prepared such that the memory address or memory pages used for this dynamic library DL may be stored, for example, in a list. Thus, identifying a memory region of non-executed code 31 can be associated with the corresponding function block 25. Alternatively or additionally, the binary code 11 in the preliminary version 26 may be provided with debugging symbols that can also be used for identifying which function block a non-executed code 31 belongs to.

Thus, a method is provided that is shrinking the OS-Kernel by performing a code-footprint optimization. An extension to libraries also possible.

A preferred embodiment of the method is a close to full automatic way of shrinking software configuration whilst proofing the code in use has been tested. This means feeding a knowledge database DB with lots of information from runtime, build time dependencies as well as static code analysis and manual input from required parts or software parts that are absolutely forbidden. The algorithm then creates a ruleset for optimizing the code footprint with the aim to shrink it as much as possible on next compile iteration.

The aim of shrinking code is interesting for other businesses as well. Reducing code footprint size has not been in focus on non-embedded systems (i.e. microprocessor systems using an OS and DLs), but it is essential to minimize per-unit costs and to maximize usability. Programming languages can also profit. Assuming nowadays microprocessor systems offer much more compute power, in the end this can lead to slower user interfaces and long boot times simply because it leads to more and more waste of compute power due to code execution of unnecessary code without taking care about resource consumption.

Overall, a tool is provided that is shrinking an OS-Kernel providing code-footprint optimization or reduction. Such a tool may implement a method for stripping down a software code (16) of a software for operating predefined device features (14) of a device , wherein device feature data (21) of the device lists the several device features (14) that are to be operated by the software by executing the software on at least one microprocessor (13), wherein the software code (16) of the software comprises of several function blocks (17) for providing a respective functionality in the device and each function block (17) is listed in software configuration data

(20), wherein the method comprises that a processing unit (15) of a system (10) performs the following steps: determining a respective demand of the device features (14) with regard to each of the function blocks (17) in a predefined code customization procedure (18), wherein the demand indicates that the respective function block (17) is required to operate the respective device feature (14), and marking in the software configuration data (20) each function block (17) for which a respective demand by at least one device feature (14) is determined, wherein each marked function block (17) is a required function block (17) and generating a binary code (11) of the software for operating the device by a compilation procedure (19), wherein the compilation procedure (19) comprises that only those of the function blocks (17) that are marked in the software configuration data (20) as being required functions blocks (17) are selected and the binary code (11) is compiled from the selected function blocks (17).

Reference numbers

10	system
11	binary code
12	vehicle
13	microprocessor
14	device feature
15	processing unit
16	software code
17	function block
18	code customization procedure
19	procedure
20	software configuration data
21	device feature data
22	rule set
25	required function block
26	preliminary version
27	code footprint
28	function call
29	system call
30	program counter
31	unused code

Claims

1. Method for stripping down a software code (16) of a software for operating predefined device features (14) of a device , wherein device feature data (21) of the device lists the device features (14) that are to be operated by the software by executing the software on at least one microprocessor (13), wherein a software code (16) of the software comprises several function blocks (17) for providing a respective functionality in the device and each function block (17) is listed in software configuration data (20), wherein the method comprises that a processing unit (15) of a system (10) performs the following steps:
   - by a predefined code customization procedure (18): determining a respective demand of the device features (14) concerning each of the function blocks (17), wherein the demand indicates that the respective function block (17) is required to operate the respective device feature (14), and
   - marking in the software configuration data (20) each function block (17) for which a respective demand by at least one device feature (14) is determined, wherein each marked function block (17) is a required function block (17) and
   - generating a binary code (11) of the software for operating the device by a compilation procedure (19), wherein the compilation procedure (19) comprises that only those of the function blocks (17) that are marked in the software configuration data (20) as being required functions blocks (17) are selected and the binary code (11) is compiled from the selected function blocks (17).

2. Method according to claim 1, wherein the code customization procedure (18) comprises:
   - generating a preliminary version (26) of the binary code (11) using all function blocks (17) or using a preliminary version (26) of the software configuration data (20) and
   - executing the preliminary version (26) of the binary code (11) in a runtime analysis environment and
   - generating logging data of observed system (10) calls and/or functions calls (28) and/or program counter (30) states using a tracing functionality of the runtime analysis environment and
   - adapting the software configuration data (20) by marking those functions blocks (17) that have been used according to the logging data and/or by unmarking those functions blocks (17) that stayed unused according to the logging data.

3. Method according to claim 2, wherein the code customization procedure (18) comprises:
   - matching memory addresses that where accessed according to the logging data with memory addresses associated with a respective function block (17) and
   - marking those function blocks (17) as required that have been accessed by the program counted according to the comparison of memory addresses.

4. Method according to any of the preceding claims, wherein the code customization procedure (18) comprises
   - performing a static code analysis is performed by generating a dependency graph of software functions of the software code (16) and
   - identifying those functions blocks (17) that are not contained in the dependency graph and
   - setting those function blocks (17) in an unmarked state in the software configuration data (20).

5. Method according to any of the preceding claims, wherein the code customization procedure (18) comprises
   - storing a dependency list of build dependencies during at least one preliminary execution of the compilation procedure (19) and
   - marking those the function blocks (17) in the software configuration data (20), that are mentioned as part of the software build of the binary code (11) in the dependency list.

6. Method according to any of the preceding claims, wherein the code customization procedure (18) comprises that
   - receiving a manual input from a user by a user interface and
   - marking those function blocks (17) in the software configuration data (20) that are indicated as necessary according to the manual input.

7. Method according to any of the preceding claims, wherein the code customization procedure (18) comprises that a static target analysis is defined by the steps of:
   - providing an association list associating possible device features (14) of the device with necessary function blocks (17) that are needed to operate the respective device feature (14),
   - identifying device features (14) comprised in the device according to the device feature data (21) and identifying the respective associated necessary function blocks (17) in the association list,

- marking those function blocks (17) in the software configuration data (20) that are necessary according to the identified device features (14).

8. Method according to any of the preceding claims, wherein software configuration data (20) is adapted iteratively in that the customization procedure (18) is repeated at least once and for each repetition, the marked function blocks (17) from the respective software configuration data (20) resulting from the previous iteration are provided in the customization procedure (18) as the set of function blocks (17) that need to be compiled and linked by the compilation procedure (19).

9. Method according to any of the preceding claims, wherein in the device feature data (21) at least of the following types of device features (14) is listed:
   - hardware components of the device,
   - software-based functionalities of the device.

10. Method according to any of the preceding claims, wherein a driver software module comprised in the software code (16) is removed by unmarking it in the software configuration data (20), if the corresponding hardware component to which the driver software is dedicated, is non-existent in the device.

11. Method according to any of the preceding claims, wherein in the software configuration data (20) at least one of the following types of function blocks (17) is listed:
   - a whole software functionality comprising several software functions and/or at least one software library,
   - a single function routine or system (10) call routine,
   - at least one single line of code within one of a function routine and a system (10) call routine.

12. Method according to any of the preceding claims, wherein the software code (16) comprises
   - an operating system (10) kernel and/or
   - at least one dynamic library and/or at least one shared object file that is designed to be dynamically linked during execution of the binary code (11) in the device.

13. Method according to any of the preceding claims, wherein the software code (16) comprises of several different, independent application programs for the device.

14. Method according to any of the preceding claims, wherein the device is a vehicle and the software code (16) comprises as least one software function that performs a driving

function in the vehicle (12), if performed by at least one electronic control unit of the vehicle (12).

15. Method according to any of the preceding claims, wherein the binary code (11) is transferred to the device and the device is operated by executing the binary code (11).

16. System (10) for generating a binary code (11) of a software for operating device features (14) in a device, wherein the system (10) comprises a processing unit (15) that is adapted to perform a code customization procedure (18) and a compilation procedure (19) and a database storage for storing device feature data (21) and software configuration data (20), wherein the system (10) is configured to perform a method according to one of the preceding claims.

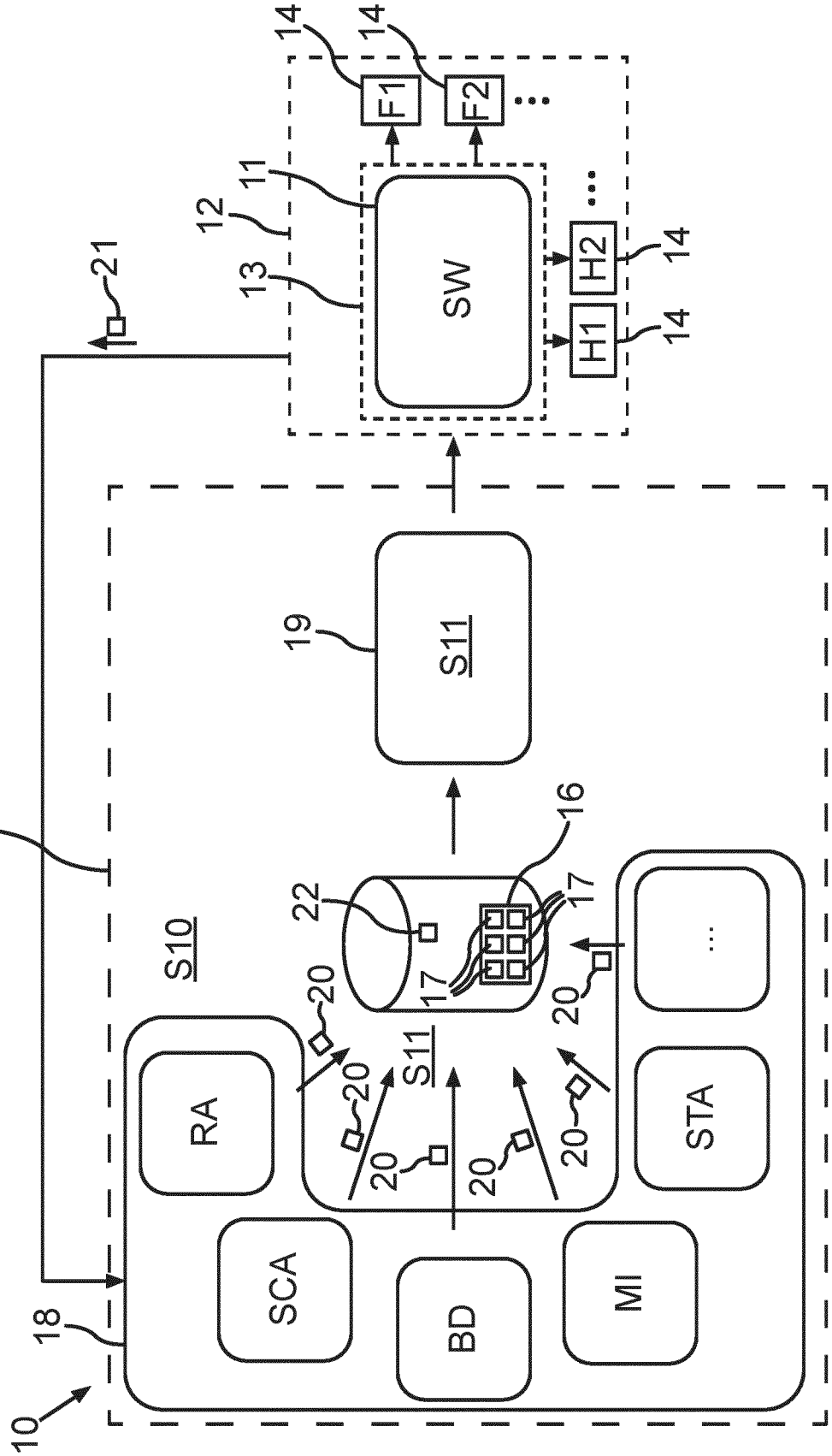
10
18
15
21
S10
RA
SCA
BD
MI
STA
S11
20
22
17
16
19
S11
SW
11
12
13
14
F1
F2
H1
H2

Fig.1

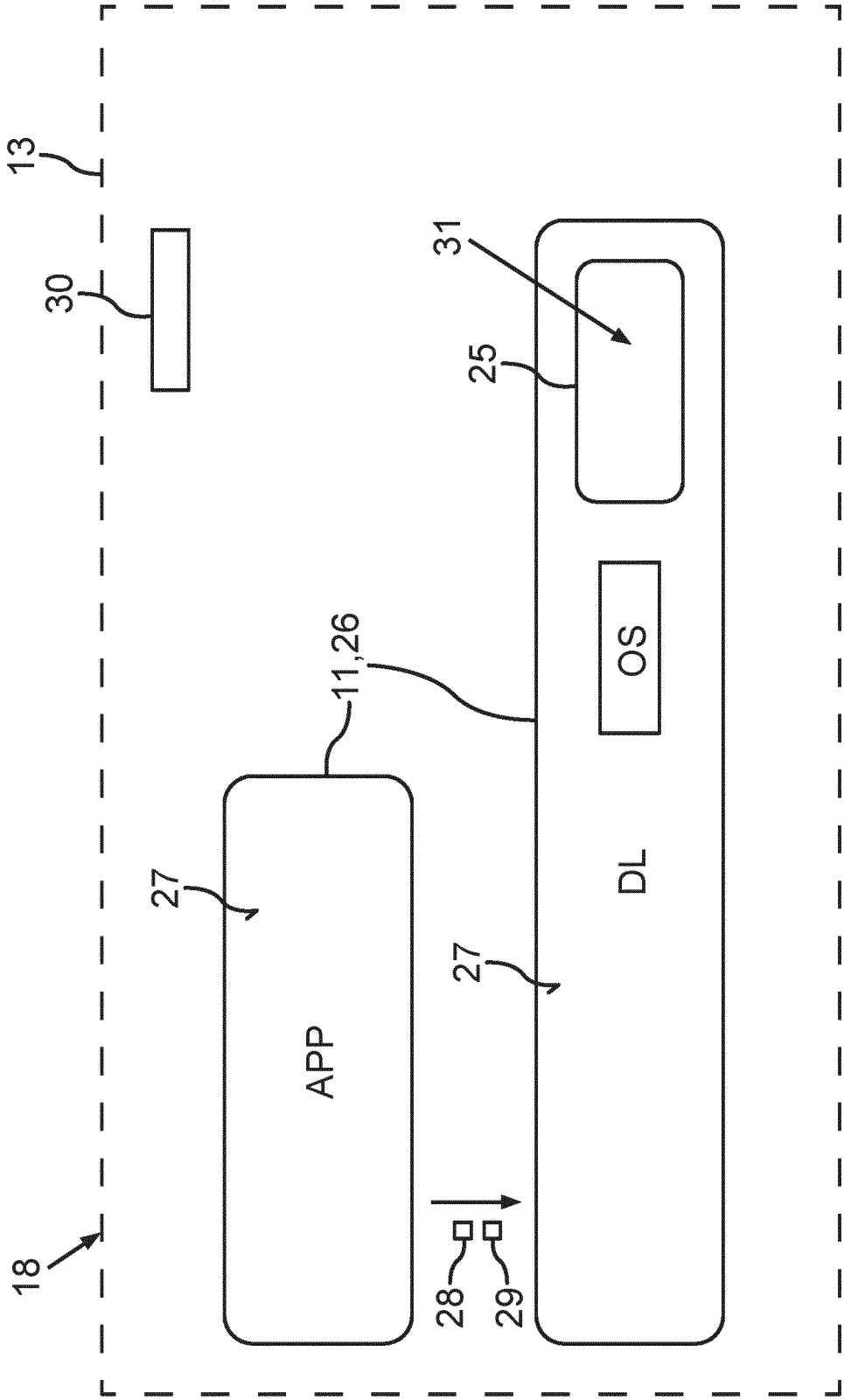

Fig.2

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2022/078722

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F8/41
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

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	ZIEGLER ANDREAS ZIEGLER@CS FAU DE ET AL: "Honey, I Shrunk the ELFs", ACM TRANSACTIONS ON EMBEDDED COMPUTING SYSTEMS, ACM, NEW YORK, NY, US, vol. 18, no. 5s, 8 October 2019 (2019-10-08), pages 1-23, XP058677280, ISSN: 1539-9087, DOI: 10.1145/3358222 abstract page 102:2, paragraph 3 - paragraph 5 page 102:4, section 3, paragraph 2 - page 102:5, paragraph 1 page 102:5, paragraph 5 - page 102:6, paragraph 4 page 102:5; figure 3 page 102:6, section 3.2, paragraph 3 - page 102:7, paragraph 1 page 102:17, section 5, paragraph 2 - paragraph 3 -/--	1-16

[x] Further documents are listed in the continuation of Box C. [ ] 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
12 January 2023	23/01/2023

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	Milasinovic, Goran

1

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	Anonymous: "Dead-code elimination - Wikipedia", , 8 October 2021 (2021-10-08), pages 1-7, XP093013068, Retrieved from the Internet: URL:https://en.wikipedia.org/w/index.php?title=Dead-code_elimination&oldid=1048901820 [retrieved on 2023-01-11] page 1, paragraph 1 page 2, section "Dynamic dead code elimination", paragraph 1 - page 3, paragraph 4	1-16