

(19) **DANMARK**

(10) **DK/EP 3995960 T3**



(12)

Oversættelse af europæisk patentskrift

Patent- og
Varemærkestyrelsen

-
- (51) Int.Cl.: **G 06 F 9/54 (2006.01)** **G 05 B 19/042 (2006.01)**
- (45) Oversættelsen bekendtgjort den: **2024-11-18**
- (80) Dato for Den Europæiske Patentmyndigheds bekendtgørelse om meddelelse af patentet: **2024-09-25**
- (86) Europæisk ansøgning nr.: **21215738.2**
- (86) Europæisk indleveringsdag: **2017-11-10**
- (87) Den europæiske ansøgnings publiceringsdag: **2022-05-11**
- (30) Prioritet: **2016-11-10 LU 93300**
- (62) Stamansøgningsnr: **17794354.5**
- (84) Designerede stater: **AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR**
- (73) Patenthaver: **PHOENIX CONTACT GMBH & CO. KG, Flachsmarktstrasse 8, 32825 Blomberg, Tyskland**
- (72) Opfinder: **HEUTGER, Henning, , 31812 Bad Pyrmont, Tyskland**
KLEINOW, Rolf, , 33619 Bielefeld, Tyskland
DREWES, Tobias, , 33039 Nieheim, Tyskland
BECK, Alexander, , 86492 Egling an der Paar, Tyskland
FRIEBEN, Jens, , 33106 Paderborn, Tyskland
- (74) Fuldmægtig i Danmark: **Gauger y Asociados SLU, C/ Anabel Segura, 10, 28108 Alcobendas, Spanien**
- (54) Benævnelse: **COMPUTERPROGRAM TIL DATAUDVEKSLING MELLEM PROGRAMMODULER**
- (56) Fremdragne publikationer:
EP-A1- 2 456 124
CH-A2- 709 007
GB-A- 2 415 806

**COMPUTER PROGRAM FOR DATA EXCHANGE BETWEEN PROGRAM
MODULES**

5 The present invention pertains to the field of automation technology and relates to a computer program for data exchange between program modules in real time.

 Current topics such as Industry 4.0, the Internet of Things or the VDI CPS agenda show that automated systems will in future be characterised by a high degree of network connection and software. In this respect there will be a shift from classical IEC 61131-3 programming to modern advanced languages. By 2018, around 70% of the code used will
10 be outside the IEC-defined languages. In addition to existing requirements such as real-time operation capability and consistency, greater flexibility, more security and better maintainability are included.

 The real-time operation capability of the global system plays an important role in the network connection of the sensors. This is particularly important in an industrial
15 environment with time-critical sequences of operations. Therefore, it is necessary to improve the real-time operating capacity of all components of an automation system. In this regard, and as a result of future developments, storage equipment and the program modules that access same will be especially important, since an ever-increasing amount of data will have to be stored and processed.

20 GB 2 415 806 A discloses a dynamic software integration architecture with program modules to integrate distributed components or systems via dynamic integration connectors, using integration rule sets that define access and communication rules associated with a respective software component, and a negotiation engine that negotiates with the integration connectors to define data and/or control connectivity based on the
25 integration rule sets.

 EP 2 456 124 A1 relates to data communication engineering, in particular for encoders in an industrial automation environment, and describes a method for communicating between functional modules. CH 709 007 A2 also discloses a service bus for connecting clients to services of service providers in a data processing system.

30 An objective of the present invention is to provide a solution to improve a data exchange between program modules in real time.

According to the invention, this objective is solved by a computer program according to patent claim 1. Other configurations arise from the dependent claims.

Accordingly, the computer program is designed as part of a firmware of a control device in an automation system and comprises instructions that cause the control device to perform the following steps for data exchange between program modules which can be executed on the control device as part of an overall application for execution on the control device by a real-time operating system:

- providing a global data space in which the program modules can store variable values so that other program modules can access them,
- 10 - configuring the computer program using a description file, wherein the description file comprises meta information that describes which variables a program module offers by using ports,
- storing the meta information of the description file, wherein the stored meta information comprises at least one variable,
- 15 - providing at least one connector, wherein the connector denotes a functionality of the computer program which provides a connection from one of the program modules to another one of the program modules, wherein the connector describes the manner in which data of the at least one variable is written and read in a memory, so that a connection can be established between the program modules, so that the at least one variable can be exchanged between one of the program modules and the computer program
- 20 - publishing variables provided by one of the program modules for at least one second program module of the program modules, so that the published variables can be accessed by at least the second program module,
- 25 further comprising instructions that cause the control device to perform the following step:
providing a programming interface that enables a program module to search the meta information.

In an exemplary method for exchanging data between program modules in real time, data storage equipment and meta information storage on the data storage equipment is provided, wherein the meta information comprises at least one variable. The method

further envisages providing at least one description file and a configuration of the data storage equipment with the description file. Furthermore, the method comprises providing at least one connector, wherein a connection to the connector can be established between program modules via the data storage equipment based on a request in the description file
5 and wherein the program modules are created using the same or different programming languages and can be executed by a real-time operating system. Furthermore, the method comprises an exchange of at least one variable between one of the program modules and the data storage equipment via the connector.

The data storage equipment comprises in particular a software component, for
10 example configured as part of the firmware of a control device, wherein the control device is configured in particular as a control device for controlling an automation system. For example, the control device can be used to read sensor data and control actuators based on the sensor data, wherein several program modules are executed on the control device for this purpose.

The data storage equipment comprises a memory via which the program modules
15 can exchange data. A functionality of the data storage equipment known as a connector provides a connection from one program module to another program module via the data storage equipment. The data storage equipment can provide local device data for the exchange between program modules via the memory and can be used equally by all
20 processes.

In particular, the meta information is read by the data storage equipment and may be assigned to a program module in each case, wherein the meta information may comprise, in particular, the identification of a variable used in the program module, as well as additional information assigned to this variable, such as a data type.

The description file defines the configuration of the data storage equipment, and
25 the description file may comprise, for example, meta information or information about the meta information to be read, for example in the form of a file path. Additionally, the description file may also comprise information defining the connectors to be created.

The description file is provided, for example, in the form of a text file in a memory
30 of a control device and is read by the data storage equipment as part of the firmware of the control device, for example, when the control device is started up or switched on.

An input to create a connector in the description file can be seen as a request to create a corresponding connector, that is, to provide the corresponding functionality on the data storage equipment. For this purpose, the description file contains, for example, information about which connection or port of a program module is to be connected to which connection or port of another program module. Connections or ports of the program modules are provided, for example, via API.

The description file can be understood as a meta-description file with which each connection or port can be specified with its respective data type and data address.

In particular, the data storage equipment thus provides a memory or a global data space in which program modules can store variables or variable values so that other program modules can access them.

For this purpose, at least one variable or a variable value is exchanged between one of the program modules and the data storage equipment via the functionality of the data storage equipment referred to as a connector, that is, a program module transmits a variable or a variable value to the data storage equipment, which is stored in the memory or global data space, or a program module retrieves a variable or a variable value stored in the memory or global data space, the communication taking place via connections of the respective program modules, which are connected by a connector.

In automation technology, a transformation from the use of classic PLC programming languages to the use of advanced languages is foreseeable. Accordingly, according to the invention it becomes possible to access a common data memory or data storage equipment in real time, even when using program modules written in different programming languages.

The IEC 61131 standard, for example, has so far regulated the syntax and semantics of programming languages for programmable controllers. In this sense, a standardized set of programming languages for programmable controllers is proposed. This set has two text languages (instruction list and structured text) and two graphical languages (ladder diagram and function block diagram). Furthermore, IEC 61131 (edition 3.0, 2013-02) defines characteristics intended to simplify communication between program controllers and other components of an automated system.

However, other programming languages can also be used according to the

invention, which are being replaced by the current classical programming languages according to IEC 61131 or are already being used in parallel. For example, program modules can be used that are each programmed in at least one of the programming languages according to IEC 61131, in Matlab/Simulink, C++, C# or JAVA or Python.

5 Other programming languages are also possible.

This allows a single programming language to be used for all program modules. It is also possible that different programming languages are used for two or more modules of the program.

Program modules are controlled via a programming interface, for example, in particular an API (application programming interface). The description file can be provided, for example, as an XML file (XML = Extensible Markup Language file).

The programming interface or API of the data storage equipment allows an application developer to look up published variables with their meta information and establish connectors at execution time. In addition to a static link via connectors, dynamic connections can also be established via connectors. This may be necessary for components whose data access changes at execution time, such as an OPC-UA server (Open Platform Communications Unified Architecture), which can offer navigation of data points at any time. For detailed analysis and troubleshooting options, the programming interface or API also provides the option to list all connectors that have been configured.

The data storage equipment and its connectors can be configured using one description file or a multitude of description files, for example in the form of one XML file or several XML files. These description files allow configuration rather than programming. This means that the function of a system can be changed at execution time without parts of the individual application having to be translated again. The use of description files or an XML-based configuration also allows the user to make adaptations directly in the controller or to use their own tools and external tool chains. The structure of a description file or XML file is specified in this respect by means of clearly defined diagram files, which allow the configuration to be checked quickly.

30 In addition to the configuration of the data storage equipment, program modules or programs and their connections or ports can also be defined with the help of description

files, in particular XML-based description files. Description files can be understood as meta-description files with which each connection or port can be specified with its respective data type and data address. Additionally, other attributes can be specified that can be used for dynamic searches on the data storage equipment. Examples of this are
5 company-specific units or labels. In order to be able to link variables from different programming domains, it is advantageous to define all data types in a common language. Using this common language, it is possible to describe programs that use self-defined primitive data types or data types specified by libraries. This general description language is advantageous for linking structures or individual elements of these structures.

10 Specifying the data storage equipment, connectors, and program metadata in multiple description files or XML configuration files allows a global application to be divided into functional modules that can be created by different groups of developers or vendors. This not only facilitates the development of a complex system control, but also the step-by-step expansion of a functioning system. The overall application does not
15 therefore have to be available from the beginning, but can be made up of submodules. In this context, a global application is understood to mean all programs or program modules, tasks and configuration settings that are necessary for execution on a control device.

 In this context, the terms "program" and "program module" are used interchangeably.

20 The description file comprises meta information. Meta information describes variables and programs, for example, which variable ports a program offers and what type of data they are. Ports are understood as variables that represent inputs or outputs of programs or components and can therefore be connected (addressed), leaving memory space available for them on the data storage equipment. Meta information describes the
25 entire program, that is, the sum of all input and output connections or ports. Meta information describes the type of variables and how they are structured, for example, boolean, integer, or complex, composite data types. An example of meta information is the boolean variable "release". A connector can transfer a variable, for example from "release" to another program to the Bit-type variable "Go". In this sense the value of the
30 variable is converted from boolean to bit.

 The variable data can be written and read in a memory of the data storage

equipment. The way this happens is described with a connector. Meta information is needed to describe which variables a program module offers by using ports. For example, a program "A" may provide the "boolean"-type variable "release". This variable is created in the memory of the data storage equipment and the value of the variable is written in
5 program "A". Other programs can read this value via a connector that knows where the variable is stored in the memory of the data storage equipment.

In an exemplary embodiment, the method further comprises providing variables to the data storage equipment by a first program module of the program modules, publishing the variables provided by the data storage equipment for at least one second
10 program module of the program modules, and an access to variables in the data storage equipment by at least the second program module.

In another design of the method, at least one buffer mechanism is provided for storing variables in the meta information on the data storage equipment.
15 Different buffer mechanisms or storage mechanisms can be used, such as a quad buffer or a single buffer as a single storage. The buffer mechanism can, for example, be adapted to scale to systems with a small working memory.

According to the invention, in another design, the method further comprises the use of a programming interface in a program module, wherein the programming interface
20 enables searching of meta information on the data storage equipment.

Furthermore, the method may advantageously provide for the dynamic generation of connectors.

For example, searching in meta information enables dynamic creation of connectors (subscription), for example by an OPC UA server. In this context, dynamic
25 means that this connection is not configured through a description file, but can be generated and also terminated at execution time via an API of the data storage equipment.

Furthermore, the method advantageously envisages that the data storage equipment is configured to supply at least some of the program modules with meta information in response to the description file, so that they can be executed in different
30 processes managed by the real-time operating system.

In another embodiment of the method, the connector may be used to convert

variables from a first data type to a second data type.

Furthermore, the method may advantageously include performing a functional operation with the connector.

Another option for connectors is the ability to perform functional operations similar to a type conversion. For example, a temperature can be converted from Fahrenheit to Celsius without additional programming.

In another embodiment of the method, an expansion of the data storage equipment by an additional program module without restarting a control system can be envisaged.

A control system denotes a sequential controller in which program modules can be loaded, started, debugged, stopped, and downloaded. The control system is based on the real-time operating system.

It is an advantage that the control system can be expanded without having to interrupt the control system for a restart. Operation can thus continue without interruption.

Furthermore, the method may advantageously envisage providing a substitute value via connector in the event that no update of a published variable has been made within a defined time interval.

To meet different program requirements, connectors can have different properties. This gives the user the option to choose between different connectors that focus on real-time capacity, data consistency, memory consumption, or a combination of these properties, for example, depending on the design objectives of the application. Connectors can additionally be responsible for providing replacement values if programs do not update their published variables within a defined time interval.

In another advantageous design, the method presents a configuration of the data memory with the description file by defining a connection, a data type for the connection and a data address of the connection.

In this sense the term "connection" should be equated with the term "port". The data address of a connection can be defined as an input port or an output port. The connection between an output port of a publication program and an input port of a subscription program is made by using a connector. The connector offers a plurality of functionalities. For example, the connector can be used to perform conversions between different data types of the program domains involved. For example, an INT in an IEC

61131-3 program can easily be converted to an int16 in a C++ program.

Data from other components, such as data from fieldbus systems, can also be reproduced on the data storage equipment as linkable ports. This allows the user to easily connect IO data (IO = Input Output) with programs from different programming domains.

5 In this way, for example, devices of a fieldbus can be structured as variables and given user-defined names.

An exemplary data storage equipment for exchanging data between program modules of the same or different programming languages in a real-time operating system comprises a memory for carrying out the method.

10 The data storage equipment is preferably configured as global data storage (GDS) equipment. Furthermore, the data storage equipment can be installed as a centrally implemented software component of a control system in an automation system. In this sense data storage equipment can provide local data of the device for exchange between programs via a memory and can be used equally by all processes.

15 In a design of the data storage equipment, an exchange area is provided in the memory of the data storage equipment, into which data can be written and read, and where program modules can access the data storage equipment via interfaces.

The data storage equipment and its connectors can support a consistent exchange of variables within a task and across task limits. The principle of conventional IEC 61131-
20 3 program execution is transferred to task execution. This means that input data is available to programs in other tasks in a task-consistent manner. In addition, meta information or data can be exchanged directly between programs in a task.

An exemplary device for exchanging data between program modules of the same or different programming languages in a real-time operating system comprises a data
25 storage equipment, at least one processor and at least one interface to an automation device.

An exemplary real-time operating system for exchanging data between program modules of the same or different programming languages comprises a data storage equipment, at least one stored description file with which the data storage equipment can
30 be configured, and several program modules that can be executed by the real-time operating system and that are created using the same or different programming languages.

In this regard, the data storage equipment is configured to connect the program modules to each other by means of connectors in order to exchange at least one variable between program modules, the meta information with the at least one variable being stored on the data storage equipment.

5 Meta information with the at least one variable can be used by connectors to establish a connection. In addition, one or more description files are used, which can be designed, for example, as an XML file. In case of a multitude of description files, these can be combined to form a configuration or global configuration. This can be done by using a configuration file that is above the description files.

10 The computer program advantageously comprises program code means for carrying out the described method, wherein the computer program is executed on a computer system in a real-time operating system.

Some exemplary embodiments of the invention are shown purely schematically in the drawings and are described in more detail below. Where:

15 Figure 1 shows an exemplary embodiment of a device for exchanging data between program modules in a real-time operating system;

Figure 2 shows an exemplary embodiment of a port link with different types of data for controlling a machine; and

Figure 3 shows an exemplary embodiment for executing tasks with different program
20 modules in a real-time operating system.

Figure 1 shows an exemplary embodiment of a device 10 for exchanging data between program modules 11, 12, 13 in a real-time operating system. In this regard, data is exchanged between program modules 11, 12, 13 in real time via a data storage equipment 20. Program modules 11, 12, 13 with the programs CPP_Filter, IEC_Sync and
25 SIM_CTRL can be connected to each other via connectors for data exchange. An exemplary connector 14, also referred to as a "smart connector", is shown schematically as an arrow between the program modules 11 and 12, which may be referred to as a "smart connector". Connector 14 provides a connection from the program module 11 to the program module 12 via data storage equipment 20.

30 In this regard, in Figure 1, the connection from the program module 11 with the CPP_Filter to the global memory 20 and from there to the program module 12 with the

IEC_Sync program is a simplified representation of the manner in which the data is accessed or the manner in which it is saved. The connector 20 defines the manner in which the data source (CPP_Filter) can write the variables/data and the manner in which the data sink (IEC_Sync) can read them. In this respect, "smart connector" means that, among
5 other things, information about data conversions can also be specified.

In addition to the name of the variable, further meta information can be stored on the data storage equipment 20. This comprises, among others, the time of the last modification, access rights, visibility, data type, user-defined attributes or descriptive text.

10 In addition to the basic data types known in all programming languages, the connectors 14 can also link user-defined data types, such as structures, arrays, or combinations of both. In this regard, a link may apply to the entire data type or to parts of it.

Figure 1 also schematically shows other components that are implemented, for
15 example, as software components, such as a fieldbus 15, an OPC-UA server 16, a web server 17 and any other component 18 of an automation installation that in each case have read and write access to data 21-31 provided on the data storage equipment 20.

Furthermore, the data storage device 10 comprises a processor 32 and an interface 33 for an automation equipment. In this example, the data storage equipment 20 is
20 connected to the processor 32 as well as to the interface 33. Connections can be wired or wireless. Furthermore, in the present exemplary embodiment, the interface 33 is connected to a component of the fieldbus 15.

The data storage equipment 11 of Figure 1 is a global memory for storing data, wherein the data 21-31 are implemented as metadata or meta information and contain at
25 least one variable. These variables may be published in the data storage equipment 11 and made available to other program modules 11, 12, 13 or components 15, 16, 17, 18 by subscribing to them and receiving them in real time.

Each program module 11, 12, 13, regardless of the programming domain, has the option of storing variables in the data storage equipment 20 as a global data space by
30 using a predefined API. Variables can be stored and published using a static configuration at the start-up of the system or dynamically at execution time. In this respect, the variables

are given a unique name with which they can be searched in the data storage equipment 20. After publication, other modules of the program can subscribe to the variable and, therefore, its values can be used. A published variable is defined as either an input port or an output port, depending on the write or read data address.

5 Figure 2 shows an exemplary embodiment for linking ports with different types of data to control a machine, in particular to control a drive train of the machine. In this respect, the connections 41 to 46 represent connectors, as shown in Figure 1. The ports that are connected in Figure 1 are not shown in the other figures. The connector 41 shows a boolean to BIT conversion, the connector 44 shows the interconnection of structures and the elements thereof, and the connector 46 shows the interconnection of vector
10 elements including byte exchange, scale/shift correction, and type conversion. In this regard, a link may apply to the entire data type or to parts of it.

Figure 3 shows an exemplary embodiment for executing tasks 50, 51 with different program modules 11, 12, 13 in a real-time operating system. The program
15 modules 11, 12, 13 receive their data, in particular the subscribed variables, from the data storage equipment 20, as shown and described in Figure 1.

Figure 3 shows an exemplary structure of tasks 50, 51, programs and connectors, taking into account a sequence control. The task-consistent data exchange between the task 50 as the main cyclic task and the task 51 as the Ctrl cyclic task or control task is
20 performed by buffer mechanisms 54 of the data storage equipment 20. Data consistency is independent of individual tasks 50, 51 or tasks, the priority thereof, and the temporal behaviour thereof.

The first task 50 comprises program modules 11 and 12 with the programs CPP_Filter and IEC_Sync, the programming languages of the program modules 11 and
25 12 being different from each other. In addition, the second task 51 comprises the program module 13 with the SIM_CTRL program. The program module 13 is also written in a different programming language compared to the programming languages of the program modules 11 and 12. This means that all program modules comprise a different programming language.

30 Figure 3 shows a schematic structure of the connections between ports. In this regard, there are ports for input and output, referred to as input ports 52 and output ports

53, via which input data and output data are sent.

The connectors are the arrows drawn in figure 3 inside and outside the tasks 50, 51. In this regard, the connectors are synchronised via buffer mechanisms 54 such as a quad buffer, for non-blocking and cycle-coherent data exchange between the program
5 modules 11, 12, 13. The connectors are used to define the manner in which data is subsequently copied to the destination. An example of a connector is the complete path from the output 53 of the task 51 to the input 52 of the task 50 through the buffer mechanism 54 of the data storage equipment 20.

In Figure 3, there is also a direct connection from the program module 11 with
10 CPP_Filter to the program module 12 with IEC_Sync in the task 50, without using a path through the connectors shown below. This connection is not shown in Figure 1.

By providing the data storage equipment 20 according to the invention as a global data memory or global data space (GDS), and as part of a data storage equipment 10, there is the possibility of performing equivalent communication between programs or
15 program modules 11, 12, 13 of different programming domains. A data exchange in real time is possible between the program modules 11, 12, 13 or the programs, which can also be written in different programming languages. In this respect, the data storage equipment 20 can assume the function of a data reserve that allows linking primitive and complex data types of different programming languages. High demands are placed on data
20 consistency and data exchange performance, particularly in real-time operation, and are met here. The data storage equipment 20 allows the management of the data connections established between programs and meta information, which enables navigation and dynamic connection to the available data.

In general, according to the invention, a functionality is made possible as part of
25 a new firmware architecture on the basis of which new industrial control systems can be implemented. This architecture is initially designed for use in a single device or control device.

In this sense, the aim is to support the incorporation of advanced languages into the conventional programming of control systems. The aim is to achieve a smooth
30 transition between the IEC 61131-3 standard and the world of advanced languages. In addition to the support for such languages, seamless integration of different programming

domains is also necessary. These include Matlab/Simulink, C++, C#, JAVA, Python and the IEC 61131-3 standard languages. For this seamless integration, the control system must be able to transfer the sequence of operations known from IEC 61131-3 with program instances and tasks to the sequence of operations of the other programming domains. Despite these more complex requirements for technical implementation, additional aspects of data security, ease of use, flexibility and, above all, performance are guaranteed.

Below is an example of the possible content of a description file for configuring a data storage equipment in XML format:

```
<?xml version="1.0" encoding="utf-8"?>
<GdsConfigurationDocument>

  <GdsComponentSettings>
    <SharedDataSettings Size="1000000" />
    <SharedMetaDataSettings Size="1000000" />
    <SharedMetaConfigSettings Size="1000000" />
    <MetaDataPaths>
      <MetaDataPath ConfigPath="./CppCounter.meta.config" />
      <MetaDataPath ConfigPath="./CppConfig.meta.config" />
      <MetaDataPath ConfigPath="./PizController.meta.config" />
    </MetaDataPaths>
  </GdsComponentSettings>

  <TaskBufferRelations>
    <Task-BufferRelation Task="TaskConf" BufferType="Single"/>
    <Task-BufferRelation Task="TaskMain" BufferType="Default"/>
    <Task-BufferRelation Task="TaskCount" BufferType="Quad"/>
  </TaskBufferRelations>

  <Connections>
    <Connection
      StartPort="Arp.Demonstrator.CppCounter-1 /CppCounter-1: Counter16"
      EndPort="Arp.Fbi.Profinet.Pnc-0/3.3:DS16"
      Min="" Max="" Log="True" RangeCheck="false"/>
    <Connection
      StartPort="Arp.Fbi.Profinet.Pnc-0/2.2:FreigabeTaster"
      EndPort="Arp.Demonstrator.CppConfig-1/CppConfig-1:FreigabeTaster"
      Min="" Max="" Log="True" RangeCheck="false"/>
    <Connection
      StartPort="Arp.Demonstrator.CppConfig-1/CppConfig-1:Freigabe"
      EndPort="Arp.Fbi.Profinet.Pnc-0/2.2:FreigabeLed"
      Min="" Max="" Log="True" RangeCheck="false"/>
    <Connection
      StartPort="Arp.Demonstrator.CppConfig-1/CppConfig-1:Freigabe"
      EndPort="Arp.Demonstrator.PizController-1/PizController-1:Freigabe_Regler"
      Min="" Max="" Log="True" RangeCheck="false"/>
    <Connection
      StartPort="Arp.Demonstrator.CppConfig-1/CppConfig-1:theta_soll"
      EndPort="Arp.Demonstrator.PizController-1/PizController-1:theta_soll"
      Min="" Max="" Log="True" RangeCheck="false"/>
    <Connection
      StartPort="Arp.Demonstrator.CppConfig-1/CppConfig-1:psi_soll"
```

```

EndPort="Arp.Demonstrator.PizController-I/PizController-1:psi_soll"
Min="" Max="" Log="True" RangeCheck="false"/>
<Connection
StartPort="Arp.Demonstrator.PizController-I/PizController-1:U_G_converted"
EndPort="Arp.Fbi.Profinet.Pnc-0/2.2:U_G_converted"
Min="" Max="" Log="True" RangeCheck="false"/>
<Connection
StartPort="Arp.Demonstrator.PizController-I/PizController-1:U_N_converted"
EndPort="Arp.Fbi.Profinet.Pnc-0/2.2:U_N_converted"
Min="" Max="" Log="True" RangeCheck="false"/>
<Connection
StartPort="Arp.Fbi.Profinet.Pnc-0/1.1:theta" EndPort="Arp.Demonstrator.PizController-
I/PizController-1:theta"
Min="" Max="" Log="True" RangeCheck="false"/>
<Connection
StartPort="Arp.Fbi.Profinet.Pnc-0/1.1:psi" EndPort="Arp.Demonstrator.PizController-
I/PizController-1:psi"
Min="" Max="" Log="True" RangeCheck="false"/>
<Connection
StartPort="Arp.Fbi.Profinet.Pnc-0/1.1:dot_theta"
EndPort="Arp.Demonstrator.PizController-I/PizController-1:dot_theta"
Min="" Max="" Log="True" RangeCheck="false"/>
<Connection
StartPort="Arp.Fbi.Profinet.Pnc-0/1.1:dot_psi"
EndPort="Arp.Demonstrator.PizController-I/PizController-1:dot_psi"
Min="" Max="" Log="True" RangeCheck="false"/>
</Connections>

</GdsConfigurationDocument>

```

Below is an example of the possible content of a file with meta information about a program module in XML format:

```

<MetaConfigurationDocument
xmlns:td="http://www.phoenixcontact.com/schema/typedomain"
xmlns="http://www.phoenixcontact.com/schema/metaconfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.phoenixcontact.com/schema/metaconfig"
schemaVersion="1.3" >
< Program
name="PizController"
version="1.0"
typeDomain="com.phoenixcontact.arp.basetypes.cpp">

<Descriptions>
<Description languageId="DE" text="Eine Programmbeschreibung" />
<Description languageId="EN" text="A program description" />
</Descriptions>
<!-- List of events that can be triggered by this program -->
<Events>
<Event name="ExampleEventThrownByAProgram" >
<Descriptions>
<Description languageId="EN" text="An example Event that is thrown by this program"
/>
</Descriptions>
</Event>
</Events>

<Ports>
<Port name="CompPerson" type="Arp.Demonstrator.PizControllerComponent:Person"
kind="Output" multiplicity="1">
<Descriptions>
<Description languageId="DE" text="Typ wurde in Komponente definiert" />

```

```

</Descriptions>
</Port>
<Port name="FunktionsStatus" type="float" kind="Output" multiplicity="1">
<Descriptions>
<Description languageId="DE" text="Typ wurde in externer Typdomain definiert" />
</Descriptions>
</Port>
<Port name="theta_soll" type="float" kind="Input" multiplicity="1">
<Attributes>
<Attribute name="Farbe" value="Rot" />
</Attributes>
<Descriptions>
<Description languageId="DE" text="Typ gehoert zur typeDomain des Programms.
Dazu kommt noch ein Attribut." />
</Descriptions>
</Port>
<Port name="psi_soll" type="float" kind="Input" multiplicity="1" />
<Port name="theta" type="float" kind="Input" multiplicity="1" />
<Port name="psi" type="float" kind="Input" multiplicity="1" />
<Port name="dot_theta" type="float" kind="Input" multiplicity="1" />
<Port name="dot_psi" type="float" kind="Input" multiplicity="1" />
<Port name="Freigabe_Regler" type="bool" kind="Input" multiplicity="1" />
<Port name="IAnteil_Aktiv" type="bool" kind="Input" multiplicity="1" />
<Port name="U_N" type="float" kind="Output" multiplicity="1" />
<Port name="U_G" type="float" kind="Output" multiplicity="1" />
<Port name="Sine" type="float" kind="Output" multiplicity="1" />
<Port name="U_G_converted" type="int16" kind="Output" multiplicity="1" />
<Port name="U_N_converted" type="int16" kind="Output" multiplicity="1" />
</Ports>

</Program>

</MetaConfigurationDocument>

```

Exemplary embodiments of a method, a data storage equipment, a device, a real-time operating system and a computer program product are described below with reference to the figures.

- 5 An exemplary method for exchanging data between program modules in real time, such as the program modules 11, 12, 13, comprises: providing a data storage equipment 20, storing meta information 21-31 in the data storage equipment 20, wherein the meta information 21-31 comprises at least one variable, providing at least one description file, configuring the data storage equipment 20 with the description file, providing at least one
- 10 connector 14, wherein the connector 14 can establish a connection between program modules can be established via the data storage equipment 20 based on a request in the description file, wherein the program modules are created using the same or different programming languages and can be executed by a real-time operating system, and exchanging at least one variable between one of the program modules and the data storage
- 15 equipment 20 via the connector 14.

Another exemplary method for exchanging data between program modules 11,

12, 13 in real-time, wherein the program modules is executed on a control device of an automation system, and wherein the program modules form part of a global application for execution on the control device, comprises providing data storage equipment 20 that provides a global data space in which the program modules can store variable values so that other program modules can access them, wherein the data storage equipment has a software component that forms part of the firmware of the control device. The exemplary method further comprises providing at least one description file, wherein the description file comprises meta information describing which variables a program module offers by using ports, a configuration of the data storage equipment 20 with the description file, wherein the meta information of the description file is stored in the data storage equipment, wherein the stored meta information 21-31 has at least one variable, and wherein at least one connector 14 is provided, wherein the connector designates a functionality of the data storage equipment which provides a connection from one of the program modules to another of the program modules via the data storage equipment, wherein the connector describes the manner in which data of the variable is written and read in a memory of the data storage equipment such that a connection can be established between the program modules 11, 12, 13 via the data storage equipment 20, wherein the program modules 11, 12, 13 can be executed by a real-time operating system, and an exchange of the variables between one of the program modules 11, 12, 13 and the data storage equipment 20 via the connector 14. Furthermore, the exemplary method envisages providing variables in the data storage equipment 20 via a first program module of the program modules 11, 12, 13, as well as publishing the variables provided by the data storage equipment 20 for at least one second program module of the program modules 11, 12, 13, and accessing the variables in the data storage equipment 20 via at least the second program module.

Preferably, the method comprises providing variables in the data storage equipment 20 via a first program module of the program modules, as well as publishing the variables provided by the data storage equipment 20 for at least one second program module of the program modules, and accessing the variables in the data storage equipment 20 via at least the second program module.

Preferably, the method further comprises providing at least one buffer mechanism

34 for storing variables in the data storage equipment 20.

According to the invention, the method provides for the use of a programming interface in a program module, for example one of the program modules 11, 12, 13, wherein the programming interface allows searching the meta information in the data storage equipment 20.

In an advantageous embodiment, the method may further comprise a dynamic generation of connectors 14.

Advantageously, the method may envisage that the data storage equipment 20 is configured to supply at least some of the program modules, for example the program modules 11, 12, 13, with meta information, for example the meta information 21-31, in response to the description file, so that they can be executed in different processes managed by the real-time operating system.

Advantageously, the method may further comprise converting the variables from a first data type to a second data type via the connector 14.

Furthermore, the method may advantageously comprise an embodiment of a functional operation with the connector 14.

Furthermore, the method can advantageously envisage an expansion of the data storage equipment 20 by an additional program module without restarting a control system.

Advantageously, the method envisages that the connector 14 provides a substitute value in the event that a published variable has not been updated within a defined time interval.

Preferably, the method may envisage that the data storage equipment 20 is configured with the description file by defining a connection, a data type for the connection and a data address of the connection.

An exemplary data storage equipment 20 for exchanging data between program modules of the same or different programming languages in a real-time operating system, for example between program modules 11, 12, 13, comprises a memory and is configured to carry out the method described above, wherein the data storage equipment 20 preferably comprises a software component for this purpose, which can preferably be part of a firmware of a control device.

Another example of a data storage equipment 20 for exchanging data between program modules in a real-time operating system, for example between program modules 11, 12, 13, comprises a memory and a software component, wherein the program modules can be executed by the real-time operating system on a control device in an automation system as part of a global application for execution on the control device, and wherein the software component forms part of the firmware of the control device, and wherein the data storage equipment provides a global data space in which the program modules can store variable values so that other program modules can access them, and wherein the data storage equipment can be configured using a description file, wherein the description file comprises meta information that describes which variables a program module offers by using ports, and wherein the meta information of the description file is stored in the data storage equipment, wherein the stored meta information 21-31 has at least one variable, and wherein the data storage equipment is configured to provide at least one connector 14, wherein the connector designates a functionality of the data storage equipment which provides a connection from one of the program modules to another of the program modules via the data storage equipment, wherein the connector describes the manner in which data of the variable is written and read in a memory of the data storage equipment such that a connection can be established between the program modules via the data storage equipment 20, and wherein the data storage equipment is configured to exchange the variables between one of the program modules and the data storage equipment 20 via the connector 14, and wherein the data storage equipment is further configured to publish variables provided by a first program module of the program modules in the data storage equipment 20 for at least one second program module of the program modules so that the variables in the data storage equipment 20 can be accessed by at least the second program module.

Preferably, the data storage equipment 20 has an exchange area in the memory in which data can be written and read, wherein program modules, for example program modules 11, 12, 13, can access the data storage equipment 20 via interfaces.

An exemplary device 10 for exchanging data between program modules of the same or different programming languages in a real-time operating system, for example program modules 11, 12, 13, comprises a data storage equipment 20 described above, at

least one processor 32, and at least one interface 33 to an automation device.

An exemplary real-time operating system for exchanging data between program modules of the same or different programming languages, for example between program modules 11, 12, 13, comprises a data storage equipment 20 or a software component of a data storage equipment 20, at least one stored description file with which the data storage equipment 20 or its software component can be configured, wherein several program modules 11, 12, 13 can be executed by the real-time operating system, which are created using the same or different programming languages, wherein the data storage equipment 20 is configured to connect the program modules 11, 12, 13 with each other using at least one connector 14 in order to exchange at least one variable 21-31 between the program modules 11, 12, 13, wherein meta information with the at least one variable 21-31 is stored in the data storage equipment 20.

An exemplary computer program product comprises program code means for carrying out the method described above, wherein the computer program product is executed on a computer system in a real-time operating system.

List of references

- 10 device
- 11 program module
- 12 program module
- 13 program module
- 14 connector
- 15 fieldbus
- 16 OPC-UA server
- 17 data storage equipment
- 18 component
- 20 data storage equipment
- 21-31 data
- 32 processor
- 33 interface
- 41-46 connectors

- 50 task
- 51 task
- 52 input port or gate
- 53 output port or gate
- 54 buffer mechanism

COMPUTERPROGRAM TIL DATAUDVEKSLING MELLEM PROGRAMMODULER

Krav

1. Dette computerprogram er designet som en del af en firmware til en styreenhed i et automatiseringssystem, omfattende instruktioner, der får styreenheden til at udføre følgende trin til dataudveksling mellem programmoduler (11, 12, 13), som kan udføres på kontrolenhed som en del af en samlet applikation til udførelse på kontrolenheden af et realtidsoperativsystem:

- Giver et globalt datarum, hvori programmodulerne kan lagre variable værdier, således at andre programmoduler kan få adgang til dem;
- konfigurering af computerprogrammet ved brug af en beskrivelsesfil, hvor beskrivelsesfilen omfatter metainformation, der beskriver hvilke variabler et programmodul tilbyder ved at bruge porte;
- lagring af metainformationen af beskrivelsesfilen, hvor den lagrede metainformation (21-31) omfatter mindst én variabel;
- Giver mindst én konnektor (14), hvor konnektoren angiver en funktionalitet af computerprogrammet, der giver en forbindelse fra et af programmodulerne til et andet af programmodulerne, hvor konnektoren beskriver, hvordan data for mindst den ene variabel skrives og læses i en hukommelse, således at der kan etableres en forbindelse mellem programmodulerne (11, 12, 13), således at mindst den ene variabel kan udveksles mellem et af programmodulerne og computerprogrammet;
- publicering af variabler leveres af et af programmodulerne (11, 12, 13) for mindst ét andet programmodul af programmodulerne (11, 12, 13), således at de offentliggjorte variabler kan tilgås af mindst det andet program modul, der også omfatter instruktioner, der får styreenheden til at udføre følgende trin:
 - giver en programmeringsgrænseflade, der gør det muligt for et programmodul (11, 12, 13) at søge i metainformationen (21-31).

2. Computerprogrammet ifølge krav 1, kendetegnes ved, at programmodulerne (11, 12, 13) er oprettet under anvendelse af forskellige programmeringssprog.

3. Computerprogramet kan ifølge et hvilket som helst af kravene 1 eller 2, give yderligere omfattende instruktioner, der får styreindretningen til at udføre følgende trin:

- giver mindst én buffermekanisme (34) til lagring af variabler (21 31).

4. Computerprogramet kan ifølge et hvilket som helst af de foregående krav, give yderligere omfattende instruktioner, der får styreindretningen til at udføre dynamisk generering af konnektorer (14).

5. Computerprogramet kan ifølge et hvilket som helst af de foregående krav, give yderligere omfattende instruktioner, der får styreindretningen til at reagere på beskrivelsesfilen ved at give mindst nogle af programmodulerne (11, 12, 13) med metainformation (21, 31), f.eks. at de kan udføres i forskellige processer, der styres af realtidsoperativsystemet.

6. Computerprogramet kan ifølge et hvilket som helst af de foregående krav, give yderligere omfattende instruktioner, der får styreindretningen til at udføre konvertering af variablerne fra en første datatype til en anden datatype ved brug af konnektoren (14).

7. Computerprogramet kan ifølge et hvilket som helst af de foregående krav, give yderligere omfattende instruktioner, der får styreindretningen til at udføre en funktionel operation ved brug af konnektoren (14).

8. Computerprogramet kan ifølge et hvilket som helst af de foregående krav, give yderligere omfattende instruktioner, der får styreindretningen til at udføre følgende trin:

- udvidelse af programmet med et ekstra programmodul (11, 12, 13) uden genstart af styresystemet.

9. Computerprogramet kan ifølge et hvilket som helst af de foregående krav, give yderligere omfattende instruktioner, der får styreindretningen til at udføre følgende trin:

- giver en erstatningsværdi af konnektoren (14) i tilfælde af, at en offentliggjort variabel ikke er blevet opdateret inden for et foruddefineret tidsinterval.

10. Computerprogramet kan ifølge et hvilket som helst af de foregående krav, yderligere omfattende instruktioner, der får styreindretningen til at udføre følgende trin:

- konfiguration af computerprogrammet med beskrivelsesfilen ved at definere en port, en datatype for porten og en dataretning for porten.

11. Dette computerprogram er designet som et realtidsoperativsystem, omfattende et computerprogram ifølge et hvilket som helst af kravene 1 til 10.

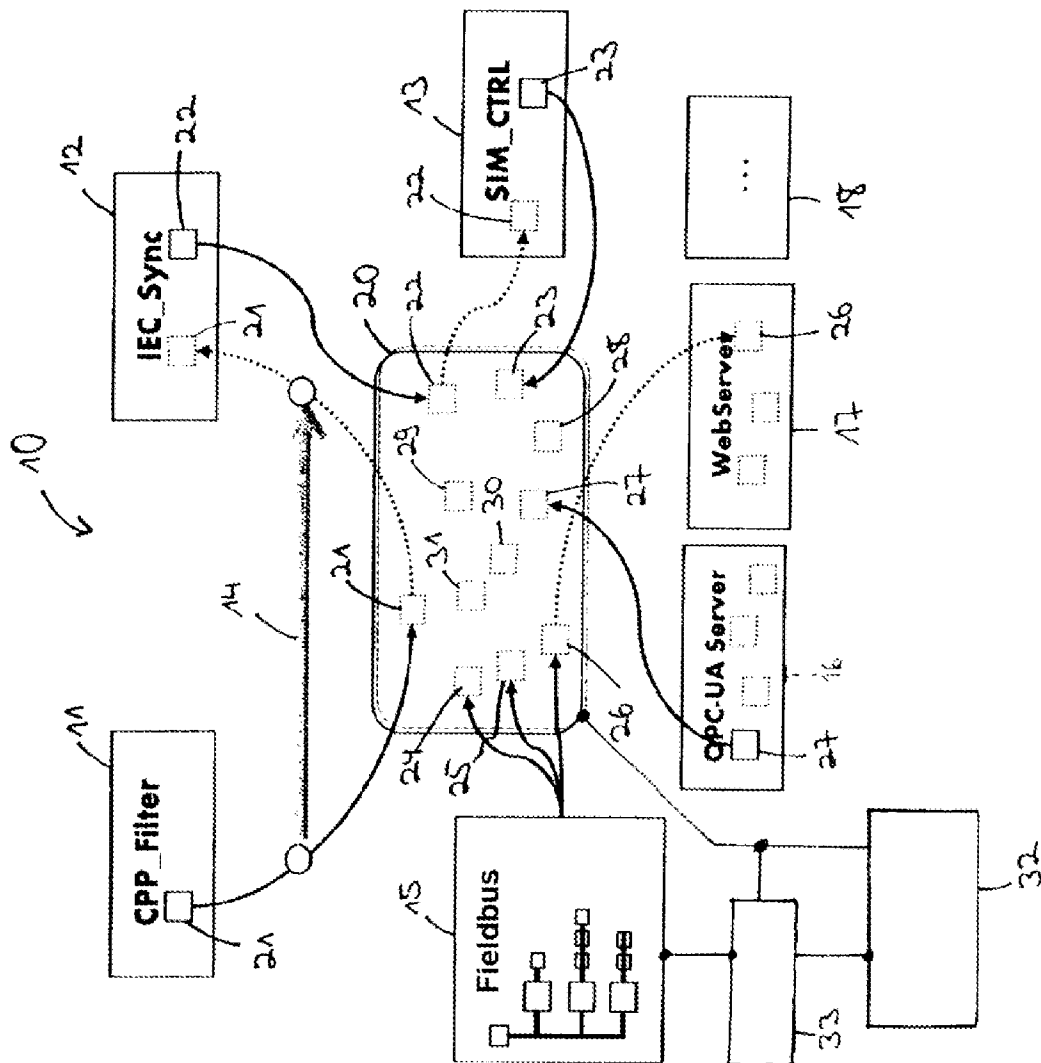


Fig. 1

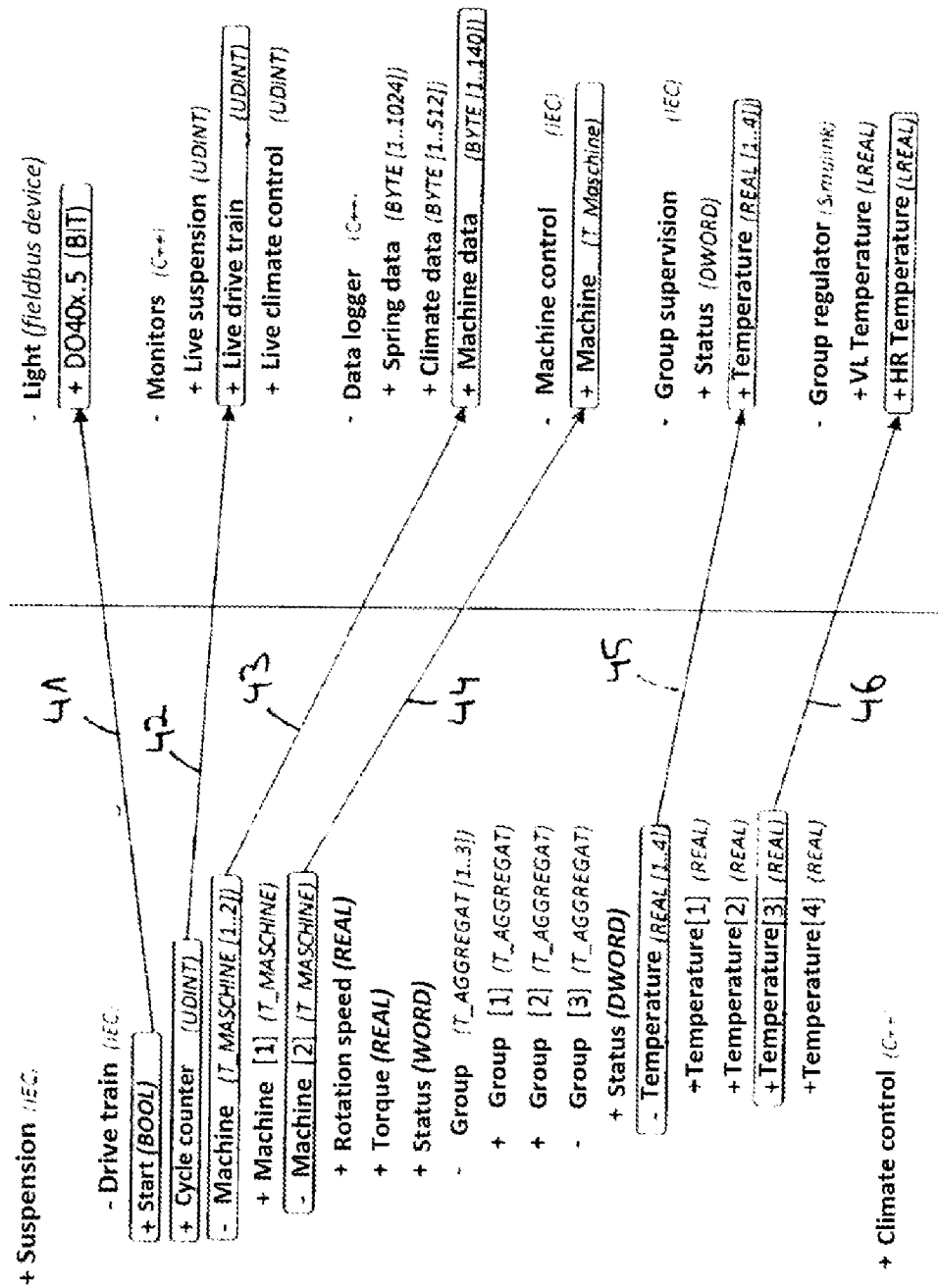


Fig. 2

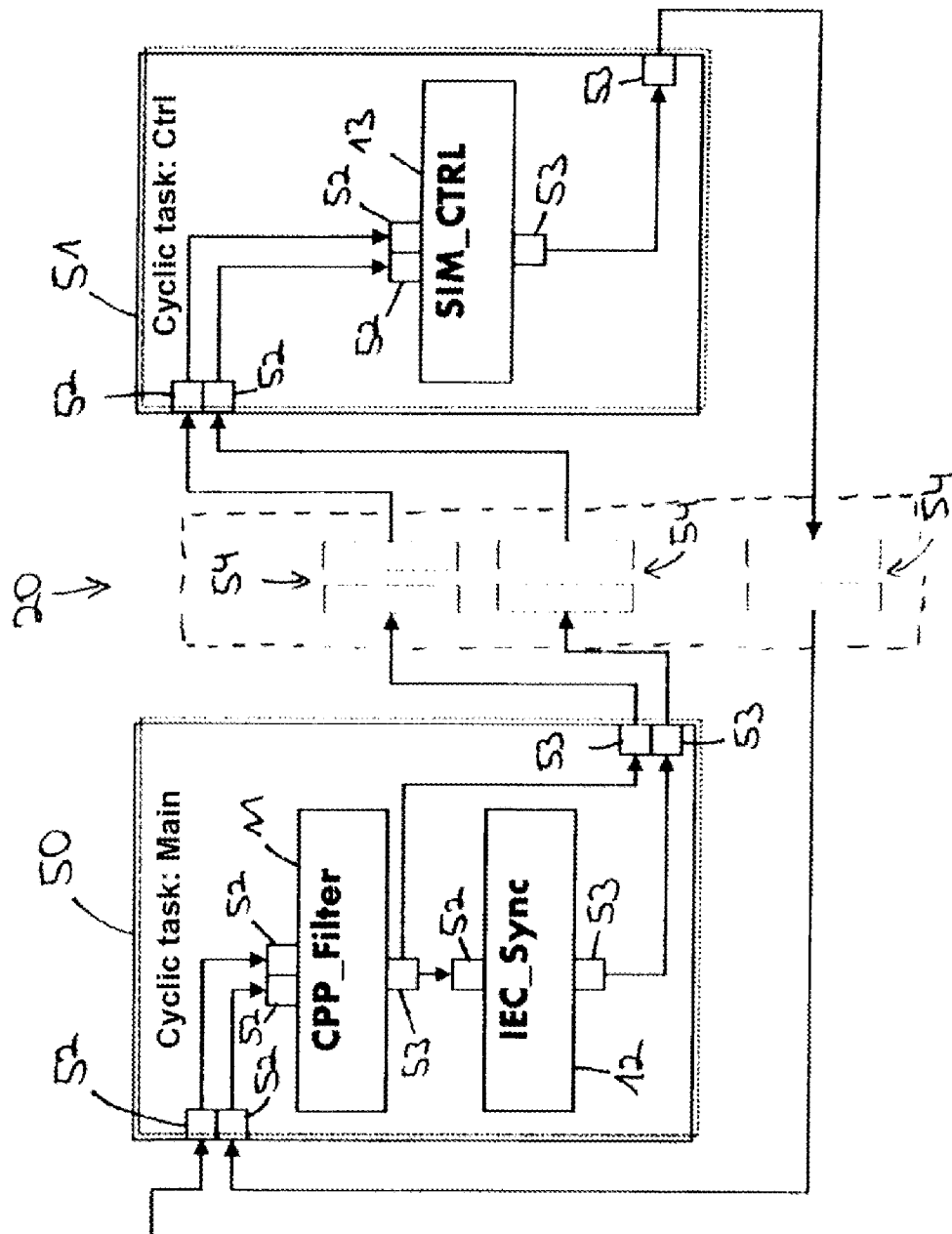


Fig. 3