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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 February 2006 (16.02.2006)

PCT

(10) International Publication Number
WO 2006/017653 A1

(51) International Patent Classification:
G06F 9/445 (2006.01) *G06F 9/46* (2006.01)
G06F 9/455 (2006.01)

(21) International Application Number:
PCT/US2005/027719

(22) International Filing Date: 4 August 2005 (04.08.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/913,119 6 August 2004 (06.08.2004) US

(71) Applicant *(for all designated States except US)*: **HONEYWELL INTERNATIONAL INC.** [US/US]; 101 Columbia Road, P.O. Box 2245, Morristown, NJ 07960 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **AGRAWAL, Mukul, B.** [US/US]; 14420 43rd Avenue North, Plymouth, MN 55446 (US). **COOPER, Saul** [US/US]; 1505 North Timber Ridge, Fridley, MN 55432 (US). **VICRAJ, Thomas, T.** [US/US]; 2365 Cavell Avenue North, Golden Valley, MN 55427 (US). **GRABA, Lee, B.** [US/US]; 4920 Emerson Avenue South, Minneapolis, MN 55409 (US).

(74) Agents: **HOIRIIS, David, Esq.** et al.; Honeywell International INC., 101 Columbia Road, P.O. Box 2245, Morristown, NJ 07960 (US).

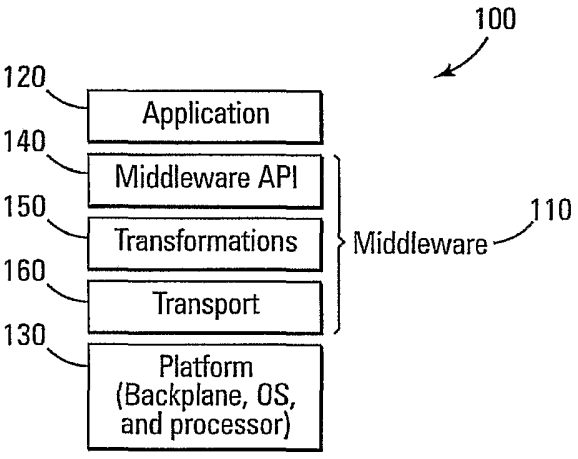
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK,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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

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

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

(54) Title: EMBEDDED SOFTWARE APPLICATION



(57) Abstract: A system includes a platform on which a plurality of platform-specific UO and fault-tolerance mechanisms are implemented. The system also includes an embedded software application operating on the platform and middleware which acts as a buffer betty en the application and the platform. In operation, the middleware logically separ~tes the embedded software application from the platform-specific UO and faulttolera~nce mechanisms, such that the application can be transferred from one platform to another without necessitating complex and time-consuming code changes.

WO 2006/017653 A1

EMBEDDED SOFTWARE APPLICATION

Technical Field

This application relates in general to high-integrity and high-availability software systems and, more specifically, to software applications embedded in specific hardware systems, such as, for example, avionics control systems.

Background

In many modern devices, the operation of the device is controlled by an advanced control system including one or more embedded software applications. For example, modern aircraft include control systems that closely monitor and control the operation of the aircraft. These control systems typically include numerous sensors and actuators that frequently collect and report data regarding the current status of the aircraft and its surroundings. The systems also include software applications that receive and analyze the data collected by the sensors and actuators, and use this data to make decisions regarding the behavior of the aircraft.

The configuration of an avionics control system can vary significantly from one aircraft to another. For example, many systems include multiple redundant sensors collecting the same data. Such a configuration enables similar data from redundant sensors to be compared to determine whether a sensor has malfunctioned. Also, in the event of a sensor malfunction, the redundant sensors enable the system to continue collecting the necessary information, thereby reducing the possibility of a total system failure. In addition, many control systems implement some form of redundancy at the application level to reduce the possibility of a system failure due to a software error or a malfunction in the underlying platform on which the application operates.

While such redundancies are common in avionics control systems, there can be considerable differences between the particular redundancy schemes adopted by different aircraft manufacturers or even in different aircraft made by the same manufacturer. For example, one aircraft may include two redundant sensors to measure airspeed, whereas another aircraft may include three redundant sensors to measure the same physical property. Further, applications themselves may have redundant copies of a given variable in order to provide fault-tolerance at the application level. In conventional avionics control systems, such differences have a significant impact on the application code

because they affect numerous I/O procedures and fault-tolerance strategies embedded in and dispersed throughout the code itself.

As a result, application developers must have a detailed understanding of the intricacies of a specific aircraft and platform configuration before creating the application code for the aircraft. In addition, even minor modifications to an aircraft computing and sensor platform design or configuration can necessitate significant reconfiguration of the corresponding application code. Such reconfiguration, if necessary, can often lead to lengthy delays in obtaining regulatory approval and certification for a given aircraft design.

Summary of the Invention

The above-mentioned drawbacks associated with existing embedded software applications are addressed by embodiments of the present invention and will be understood by reading and studying the following specification.

In one embodiment, a system comprises a platform on which a plurality of platform-specific I/O and fault-tolerance mechanisms are implemented. The system further comprises an embedded software application operating on the platform, and middleware interposed between the embedded software application and the platform. The middleware logically separates the embedded software application from the platform-specific I/O and fault-tolerance mechanisms.

In another embodiment, a system includes an embedded software application utilizing application-specific I/O signals. A method of configuring the system comprises calling an initialization procedure and, during the initialization procedure, referencing configuration data regarding the platform on which the embedded software application operates. The method further comprises utilizing the referenced data to instantiate a signal map in which platform-specific I/O signals are correlated with application-specific I/O signals.

In another embodiment, a system comprises a plurality of redundant sensors configured to measure data representing multiple estimates of a given physical property and a plurality of redundant applications representing one or more ways of computing an intermediate value. The system further comprises a transformation module configured to integrate the outputs of the redundant sensors and/or the redundant applications into a uniform signal passed to an embedded software application.

In another embodiment, a method of controlling the operation of a system comprises receiving at least one system input signal in a first format over an input port and transforming the system input signal(s) into at least one application input signal having a second format. The method further comprises creating at least one application output signal in the second format, converting the application output signal(s) into at least one system output signal having the first format, and sending the system output signal(s) over an output port.

In another embodiment, a control system comprises a platform including a backplane, an operating system, and one or more processors. The control system further comprises middleware operating on the platform and an application interacting with the middleware. The middleware receives a plurality of system input signals in a first format and transforms the system input signals into a plurality of application input signals having a second format expected by the application. The middleware also transforms a plurality of application output signals in the second format into a plurality of system output signals having the first format and sends the system output signals over an output port.

The details of one or more embodiments of the claimed invention are set forth in the accompanying drawings and the description below. Other features and advantages will become apparent from the description, the drawings, and the claims.

Brief Description of the Drawings

Figure 1 is a block diagram of one embodiment of an avionics control system having a middleware layer.

Figure 2 is a diagram illustrating signal I/O mapping for an application operating within a partition.

Figure 3 is a diagram illustrating the configuration of one embodiment of the middleware.

Figure 4 is a flow chart illustrating the process for creating the structure of the middleware during partition startup in one embodiment.

Figure 5 is a diagram illustrating the relationships between the main classes in the middleware.

Figure 6 is a diagram illustrating one embodiment of a fault-management transform.

Like reference numbers and designations in the various drawings indicate like elements.

Detailed Description of the Preferred Embodiment

In the following detailed description, reference is made to the accompanying drawings that form a part hereof, and in which is shown by way of illustration specific illustrative embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that logical, mechanical, and electrical changes may be made without departing from the spirit and scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense.

Figure 1 is a block diagram of one embodiment of an avionics control system node 100 having a middleware layer 110. In the illustrated embodiment, the system node 100 comprises an application 120 operating on a platform 130 comprising a backplane, an operating system, and one or more processors. Figure 1 shows a single node 100 of a multi-node system communicating through a network or a backplane. In operation, the node 100 typically exchanges data with other nodes 100 in the system. Whereas in conventional avionics control systems the application interacts directly with the underlying platform, in the system illustrated in Figure 1, the middleware 110 acts as a buffer between the application 120 and the platform 130 on which it operates.

In operation, the middleware 110 exports to the application 120 an application program interface (API) that provides for high-level I/O interaction at the application level. The API also brokers I/O calls from the application 120 to platform-specific interactions. The form of those interactions is preferably determined through configuration of the middleware 110, rather than through compiled code. Thus, changes in the platform 130 on which the application 120 operates or even in the aircraft associated with the application 120 (assuming the same requirements between aircraft) can advantageously be handled through configuration of the middleware 110, rather than through complex and time-consuming re-coding of the application 120. In a preferred embodiment, the middleware 110 is sufficiently parameterized such that the expected types of variations across platforms 130 and aircraft can be handled through configuration.

In the illustrated embodiment, the middleware 110 comprises an API layer 140, a transformation layer 150, and a transport layer 160. The API layer 140 is exposed to the applications 120. In some embodiments, the API layer 140 comprises three elements.

The first element is a set of classes that define a small set of signal types, each representing a different type of value, referred to as application signals. In operation, applications 120 refer to the definitions stored in the first element of the API layer 140 to create instances of the defined signal types corresponding to the inputs and outputs with which the applications 120 need to interact. An application 120 can also query the signals about their characteristics, such as their current validity, freshness, update rate, expected latency, allowable range, and for floating point types, resolution and precision.

The second element of the API layer 140 comprises two classes that are used to group together application signals that should be refreshed or written all at the same time. In one particular exemplary embodiment, these classes are referred to as InputSignalGroup and OutputSignalGroup. To provide an example, if an application 120 has a 10 Hz periodic task, with a number of input signals used by that task, including redundant input signals, the application 120 would create the appropriate signals, create an InputSignalGroup, associate the signals with the group, and register the group with the middleware 110, as discussed in more detail below. Then, on every cycle, the refresh method would be called on the InputSignalGroup, causing the middleware 110 to refresh the appropriate signals to be updated with current values. A similar arrangement could be used with output signals, except that an OutputSignalGroup would be used, and the write method would be used to send the current values of the associated output signals out into the platform 130.

The third element of the API layer 140 comprises two registration functions that are used to register an Input/OutputSignalGroup with the middleware 110. In one particular exemplary embodiment, these functions are referred to as registerInputSignalGroup and registerOutputSignalGroup. In operation, these registration functions enable the middleware 110 to set up the data paths between the signals created by the application 120 and the system signals that exist at the platform layer.

In a preferred embodiment, the API layer 140 of the middleware 110 is sufficiently rich that applications 120 can be written to interact only with the API layer 140. As a result, applications 120 can specify I/O data without platform-specific information, such as addresses, board type, board number, etc., and need not be written to be platform- and aircraft-specific. Rather, a standard application 120 can be moved from one aircraft to another, without necessitating changes to the I/O interaction code of the application 120.

In the embodiment illustrated in Figure 1, the second layer of the middleware 110 is the transformation layer 150. In operation, this layer converts signals generated at the application level to signals that are meaningful at the platform level and vice versa. In general, application-level signals referred to as "application signals," and platform-level signals are referred to as "system signals." Although there is typically some relationship or correspondence between application signals and system signals, the particular characteristics of the application signals are often different from those of the system signals. This is especially likely if the application 120 is ported from one aircraft to another in which similar signals are defined differently.

To provide one example, a signal for airspeed may be defined in units of knots in one aircraft, but in units of meters/second in another aircraft. In a conventional avionics control system, a set of code changes would be required to transfer an application developed for the first aircraft to the second to account for the difference in units. In the system illustrated in Figure 1, however, the transformation layer 150 of the middleware 110 can be used to transform system signals into the forms that are expected by the application 120, and vice versa. In this particular example, i.e., a units mismatch, a scaling transformation can be used to convert the airspeed system signal into whatever units are expected by the application 120, thereby advantageously eliminating the need for a set of code changes in the application 120.

In a preferred embodiment, a number of standard transformations are constructed in advance. These standard transformations are preferably parameterized, meaning that their characteristics can be determined by parameters read at startup time from configuration data. Such an arrangement greatly reduces the number of code changes that must be performed to transfer an application 120 from one aircraft or platform 130 to another, although as a practical matter, it is likely that the need for such code changes cannot be eliminated entirely.

In addition to addressing mismatches in signal format, the transformation layer 150 also addresses the correlation and matching of system signals and application signals. For example, a first aircraft or platform 130 may assign a first ID to a given system signal, e.g., airspeed, whereas a second aircraft or platform 130 may assign a different ID to the same system signal. The transformation layer 150 of the middleware 110 can be used to map the appropriate system signals onto the corresponding application signals, and vice versa, as illustrated in Figure 2.

In the illustrated embodiment, the application operates within a discrete partition 210, as set forth in the ARINC 653 specification, which is described in more detail below. In operation, the transformation layer 150 of the middleware 110 matches the system input signals 220 received at the partition boundary with their equivalent application input signals 230. The transformation layer 150 also matches the application output signals 240 with their equivalent system output signals 250 to be transferred beyond the partition 210 over an ARINC 653 channel. The characteristics of the mapping function performed by the transformation layer 150 are preferably determined by parameters read at startup time from configuration data. As a result, code changes are not required to account for differences between IDs assigned to the same system signals by different aircraft or platforms 130.

Similar techniques can be used to account for differences in the redundancy schemes implemented on different aircraft or platforms 130. For example, one aircraft may include two redundant sensors to measure airspeed, whereas another aircraft may include three redundant sensors to collect the same data. In either case, the accepted airspeed signal value can be determined using any of a wide variety of well-known "sensor voting" algorithms. Once such a determination is made, the aircraft may include multiple exact replicas of the accepted airspeed signal value in redundant programs executing on identical computers in accordance with a variety of software redundancy schemes.

In some embodiments, these redundancy schemes are implemented in a series of "fault-management" transforms, one example of which is illustrated in Figure 6. In the illustrated embodiment, the transform 610 comprises one element of the transformation layer 150 of the middleware 110. In operation, the transform 610 accepts a plurality of redundant input signals 620 and produces a single transformed signal 630 that is in turn used by the application 120 to generate one or more output signals 640. In some embodiments, a fault-management transform 610 is defined for a specific aircraft platform and is independent of the actual sensors being used. To provide one example, one transform 610 may choose the first valid input from a plurality of sensors, whereas another transform 610 may choose the median value of all valid inputs.

The details of the particular hardware and software redundancy schemes implemented on a given platform 130 are ultimately irrelevant to the applications 120 operating on the platform 130. For example, an application 120 referencing airspeed

typically does not need to know how many sensors gather airspeed data or how an airspeed signal value is determined; rather, the application 120 simply needs to know the accepted value of the airspeed signal. Thus, in a preferred embodiment, the transformation layer 150 of the middleware 110 combines all of the signals or inputs that relate to a single physical or computed property into one "redundant signal group," and passes the accepted signal value to requesting applications 120 in the proper form. Applications 120 preferably do not distinguish between inputs from sensors and inputs from other applications 120.

Moreover, in some embodiments, a platform 130 includes a plurality of redundant applications 120 representing multiple ways of computing a given intermediate value. The transformation layer 150 of the middleware 110 can integrate the outputs of the redundant applications 120 into a uniform output signal passed to other (possibly redundant) applications 120. The redundant applications 120 need not be aware of the existence of their replicas.

The transformation layer 150 of the middleware 110 enables applications 120 to receive input signals and generate output signals in the form they expect, while the details of the particular hardware and software redundancy schemes implemented on the underlying platform 130 remain transparent. Accordingly, the transformation layer 150 advantageously enables a degree of separation between the development of an application 120 and the development of the overall system, and enables the application 120 to be relatively independent of the fault-tolerance and I/O details of the overall control system.

Referring again to Figure 1, the third layer of the middleware 110 is the transport layer 160, which interacts with the platform 130 and with the transformation layer 150. In operation, the transport layer 160 acquires input system signals from the platform 130 and delivers these signals to the transformation layer 150. The transport layer 160 also receives output system signals from the transformation layer 150 and delivers these signals to the platform 130.

In some embodiments, the system operates in accordance with the ARINC 653 specification. As will be understood by those of ordinary skill in the art, this specification sets forth standards for a time and space partitioning operating system in which applications 120 are separated into discrete partitions on a single processor. Each partition is guaranteed a slice of time for execution as specified by the system designer.

The specification provides for the transfer of data between partitions over 653 channels through which arrays of bytes can be transferred. There are no industry-wide standards, however, for the format of the data passed over ARINC 653 channels. As a result, the format of the data may vary from aircraft to aircraft. Accordingly, the definitions for data packets transmitted over the ARINC 653 channels are implemented in the transport layer 160 of the middleware 110. This configuration advantageously enables signal data to be transferred efficiently across a channel by including several related signals in the same packet and sending only that type of data across the channel.

To provide an example, a navigation application 120 might produce three position signals (e.g., latitude, longitude, and altitude) at a rate of 10 Hz. The transport layer 160 for this application 120 would pack all three signal values into a single packet, and send them over a given channel at 10 Hz. Applications 120 that need to use any or all of those signals would be consumers of that channel, and pull the necessary signals out of the packet, as defined by the navigation application 120. The details related to the packet definition would preferably be determined by parameters read at startup time from configuration data.

In some embodiments, this approach is implemented by constructing a set of system signal classes representing the same data types as the application signal classes. These classes have the capability to serialize their values and attributes (e.g., ID, type, size, CRC) into a byte array, as well as to update their values from a byte array. Two classes are constructed to interact with the channel, as well as hold the specification of the ID, type, and order of signals included in each packet. In one particular exemplary embodiment, these classes are referred to as SignalListReader and SignalListWriter.

Figure 3 is a diagram illustrating the configuration of one embodiment of the middleware 110. In the figure, arrows denote containment. For example, in the illustrated embodiment, the middleware 110 comprises a PartitionConfig class 305, which contains an InputChannels list 310, an OutputChannels list 315, an ApplicationConfig object 320, and a Linkups list 325. The InputChannels list 310 and the OutputChannels list 315, in turn, contain a number of ChannelConfig objects 330, each of which contains a plurality of SystemSignalConfig objects 335.

The PartitionConfig class 305 contains all the configuration information needed by the partition 210 to startup. Specifically, the lists of input and output channels, along with configuration information about those channels, enable the middleware 110 to create

the input and output ports for those channels. In addition, each ChannelConfig object 330 contains a list of SystemSignalConfig objects 335, which provide the information about the signal data that will be transferred over the channels. This configuration provides the middleware 110 with all of the information needed to send a list of signals over a channel, and to read a list of signals from the channel.

In the illustrated embodiment, the ApplicationConfig object 320 contains an InputSignals list 340 and an OutputSignals list 345. The InputSignals list 340, in turn, contains an AppSignalConfig object 350. This configuration provides the ApplicationConfig object 320 with all of the information necessary to create and start the application 120.

In addition, in the illustrated embodiment, the Linkups list 325 contains LinkupConfig objects 355, each of which contains a TransformConfig object 360, a SystemSignalList 365, and an AppSignalList 370. This information is used to map between the system signals 220, 250 read into or written out of the partition 210, and the application signals 230, 240. The information is also used to create the transformations needed to perform the mapping functions.

In some embodiments, the system signals and application signals are defined in abstract base classes holding a number of common attributes, such as the signal ID (which is unique across the system), the type of the object, and the validity and freshness of the signal. The system signal abstract base class also preferably has the ability to convert its ID, type, value, and a CRC value into a byte array, and to update these attributes from a byte array.

Transformation classes, which are also abstract base types, provide a link between the system signals and the application signals. In some embodiments, the middleware 110 comprises three transformation classes, which are referred to as InboundTransform, OutboundTransform, and RedirectTransform. In these embodiments, an InboundTransform converts an incoming system signal into an application input signal, an OutboundTransform converts an outbound application signal into a system signal, and a RedirectTransform routes an application output signal back into the application 120 as an input signal. As discussed above, in a preferred embodiment, there are a number of standard transformations that would be chosen at configuration time, which do not need to be re-coded and re-compiled. For example, a general polynomial transform, with both input range limits and output range limits, and a polynomial transformation may be able

to handle a large percentage of transformations between like (e.g., float-to-float, double-to-double, etc.) signals.

Figure 4 is a flow chart illustrating the process for creating the structure of the middleware 110 during partition startup in one embodiment. In a first step 405, a configuration tree representing the configuration of the partition 210 is created. In a next step 410, a determination is made as to which ARINC 653 ports need to be created. Then, in a step 415, the system signals to be transported across each channel are identified. In a step 420, the information needed to create the system signals is determined. In a next step 425, the SignalListReader and SignalListWriter classes are constructed with the appropriate ports and signals. In some embodiments, each SignalListReader is assigned to one port, and is able to pull packets out of the port and use the packets to refresh the values of the corresponding signals. Similarly, each SignalListWriter is assigned to one port, and is able to write the values for its list of system signals into a byte array, which is sent over the output port.

In a step 430, the transformation objects are created. As discussed above, these objects provide a link between the system signals and the application signals. In a next step 435, the application object is created, and in a final step 440, an init method is called. The init method creates the appropriate application signals, associates them with InputSignalGroups and OutputSignalGroups, and registers these groups. During the registration process, the application signals are matched up to the corresponding transformation objects by referencing the signal names and information stored in the LinkupConfig object loaded by the partition 210. As a result, the application signals are implicitly matched up to the equivalent system signals, and each application signal is associated with the appropriate SignalListReader or SignalListWriter.

Figure 5 is a diagram illustrating the relationships between the main classes in the middleware 110 following the construction process illustrated in Figure 4. In the illustrated embodiment, the middleware 110 comprises a Partition 505 object in communication with an Application 510 object through a series of InputPort 515, SignalListReader 520, and InputSignalGroup 525 objects, as well as through a series of OutputPort 530, SignalListWriter 535, and OutputSignalGroup 540 objects. Both the InputSignalGroup 525 and the OutputSignalGroup objects 540 are in communication with a number of SystemSignal 545, Transformation 550, and Signal 555 objects. The Signal 555 objects are also in communication with the Application 510 object. Similarly, the

SystemSignal 545 objects are in communication with the SignalListReader 520 and SignalListWriter 535 objects, respectively.

In operation, on each cycle, the Application 510 calls the refresh method on its associated InputSignalGroups 525. The InputSignalGroups 525, in turn, call the refresh methods on their associated SignalListReaders 520, which causes the SignalListReaders 520 to read their InputPorts 515, extract data packets, and use the packets to refresh the system signals. Each Transformation 550 then calls a doTransform method to process any conversions between system signals and application signals, if necessary. Once this process is complete, the values of the application signals are updated into the appropriate form expected by the Application 510. The Application 510 then uses a getValue method to get the current value of each signal.

With respect to outbound signals, the Application 510 calls the setValue methods of its associated output signals, and then calls the write method of the corresponding OutputSignalGroups 540. The OutputSignalGroups 540 execute the Transformations 550 necessary to convert application signal outputs into system signal outputs. The appropriate SignalListWriters 535 are then called to convert the system signals into byte arrays to be sent out over the OutputPorts 530 in accordance with the ARINC 653 specification.

The systems and methods described above present a number of distinct advantages over conventional embedded software applications. For example, using the systems and methods described above, common I/O operations and fault-tolerance strategies can be abstracted out, meaning that duplicate, dissimilar, platform-specific mechanisms for I/O interaction and fault-tolerance can be pulled out of the application code and placed into the common middleware layer. This abstraction advantageously reduces the amount of code that is duplicated (in sometimes inconsistent ways) across applications and standardizes I/O and fault-tolerance interaction techniques across applications.

This approach also advantageously enhances the portability of applications across platforms and insulates applications from changes in a given platform. As a result, applications can be transferred more efficiently between different aircraft, or between different models of the same aircraft having different sensors, actuators, or redundancy schemes. In addition, aircraft designers can be given greater flexibility when contemplating changes in the platform configuration of a given aircraft.

The systems and methods described above also advantageously insulate application developers from the I/O and fault-tolerance complexities of the underlying platforms on which their applications operate. As a result, application developers do not have to be concerned with the specifics of where signals originate and how they are transported through the system. Rather than incorporating complex, platform-dependent I/O and fault-tolerance operations into their applications, developers can expect that incoming signals will be delivered in the proper form to their application boundary, and that output signals will be delivered from their application to the appropriate destinations.

Moreover, while the foregoing description refers primarily to an avionics control system as one exemplary embodiment, the systems and methods described herein can be applied to a wide variety of other devices having embedded software applications. For example, these systems and methods can be implemented in applications embedded within other mobile vehicles (e.g., tanks, automobiles, ships, space vehicles), medical devices (e.g., MRI and X-Ray machines), industrial control applications (e.g., oil and gas refining, paper and pulp, pharmaceuticals, chemicals), power-transmission, and high-availability enterprise applications (e.g., financial systems).

Although this invention has been described in terms of certain preferred embodiments, other embodiments that are apparent to those of ordinary skill in the art, including embodiments that do not provide all of the features and advantages set forth herein, are also within the scope of this invention. Accordingly, the scope of the present invention is defined only by reference to the appended claims and equivalents thereof.

WHAT IS CLAIMED IS:

1. A system 100 comprising:

a platform 130 on which a plurality of platform-specific I/O and fault-tolerance mechanisms are implemented;

an embedded software application 120 operating on the platform 130; and

middleware 110 interposed between the embedded software application 120 and the platform 130,

wherein the middleware 110 logically separates the embedded software application 120 from the platform-specific I/O and fault-tolerance mechanisms.

2. The system 100 of Claim 1, wherein the system 100 comprises an avionics control system.

3. The system 100 of Claim 1, wherein the middleware 110 comprises an API layer 140, a transformation layer 150, and a transport layer 160.

4. The system 100 of Claim 1, wherein the middleware 110 comprises a plurality of standard parameterized transformations.

5. The system 100 of Claim 1, wherein the middleware 110 is configured to combine multiple signals or inputs related to a single physical property or computed value into a redundant signal group.

6. A method of configuring a system 100 having an embedded software application 120 utilizing application-specific I/O signals, the method comprising:

calling an initialization procedure;

during the initialization procedure, referencing configuration data regarding the platform 130 on which the embedded software application 120 operates; and

utilizing the referenced data to instantiate a signal map in which platform-specific I/O signals are correlated with application-specific I/O signals.

7. The method of Claim 6, wherein the configuration of I/O code and transformation code is performed independently of the configuration of the embedded software application 120.

8. The method of Claim 6, wherein the system 100 comprises an avionics control system.

9. The method of Claim 6, wherein the system 100 comprises middleware 110 including an API layer 140, a transformation layer 150, and a transport layer 160.

10. The method of Claim 9, wherein the middleware 110 is configured to correct mismatches in signal format between platform-specific signals and application-specific signals.

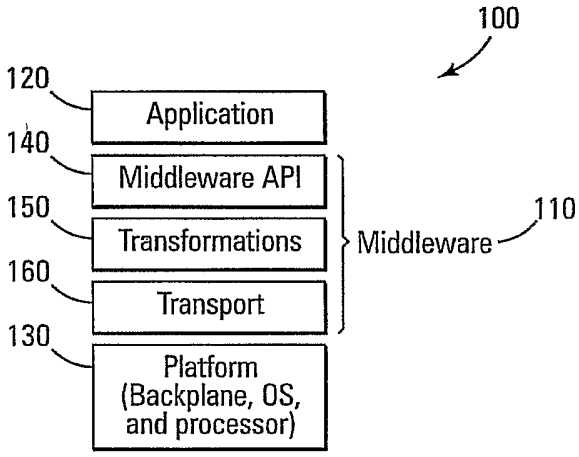

100
120
Application
140
Middleware API
150
Transformations
160
Transport
130
Platform
(Backplane, OS,
and processor)
Middleware
110

FIGURE 1

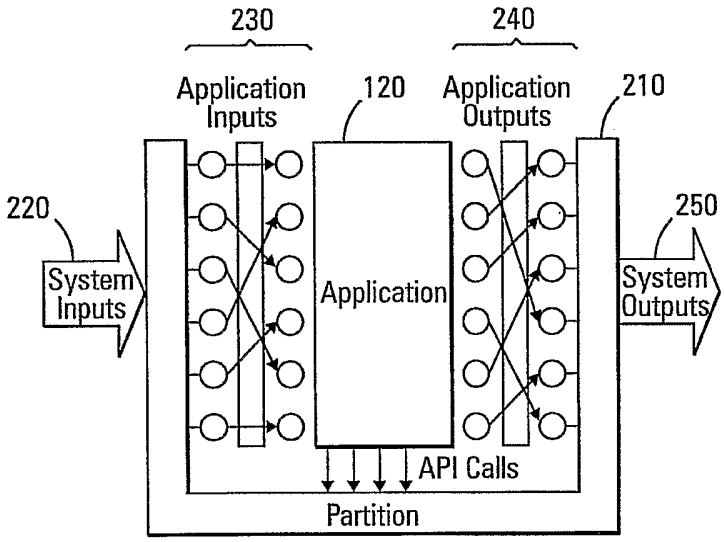

230
240
Application Inputs
120
Application Outputs
210
220
250
System Inputs
Application
System Outputs
API Calls
Partition

FIGURE 2

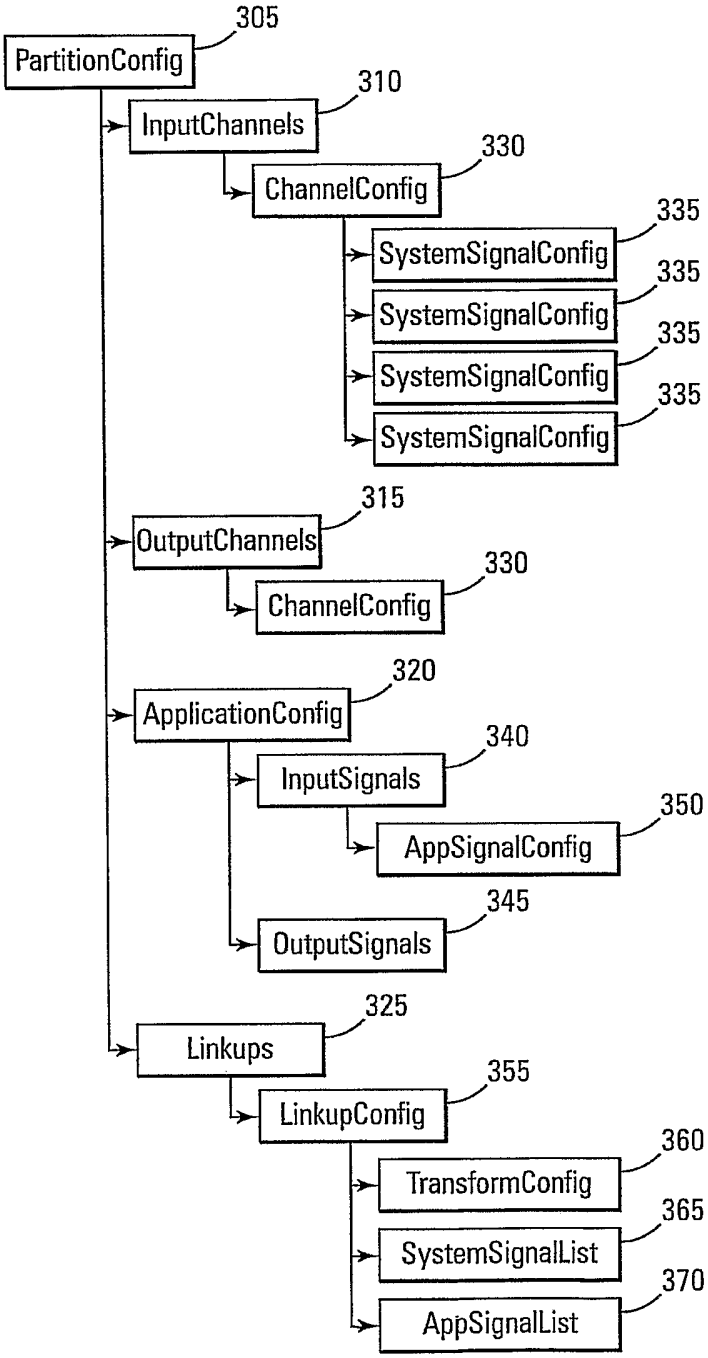

PartitionConfig
305
InputChannels
310
ChannelConfig
330
SystemSignalConfig
335
SystemSignalConfig
335
SystemSignalConfig
335
SystemSignalConfig
335
OutputChannels
315
ChannelConfig
330
ApplicationConfig
320
InputSignals
340
AppSignalConfig
350
OutputSignals
345
Linkups
325
LinkupConfig
355
TransformConfig
360
SystemSignalList
365
AppSignalList
370

FIGURE 3

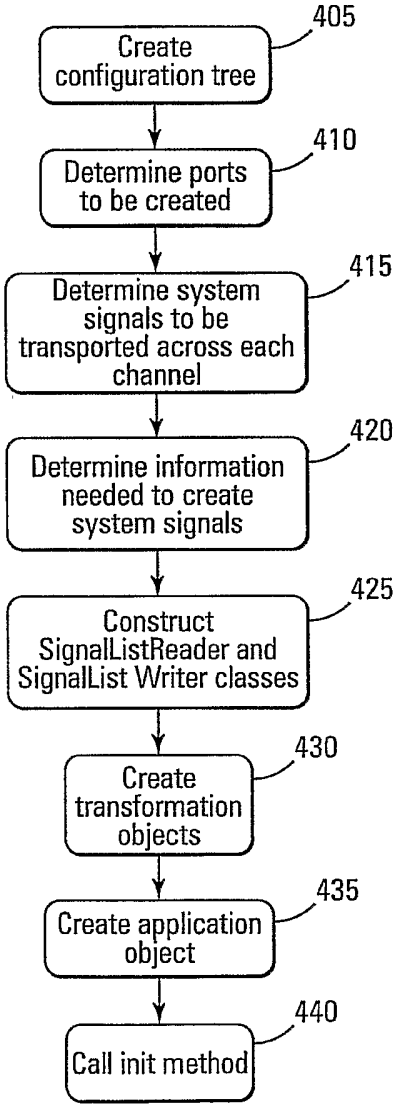

405
Create configuration tree
410
Determine ports to be created
415
Determine system signals to be transported across each channel
420
Determine information needed to create system signals
425
Construct SignalListReader and SignalList Writer classes
430
Create transformation objects
435
Create application object
440
Call init method

FIGURE 4

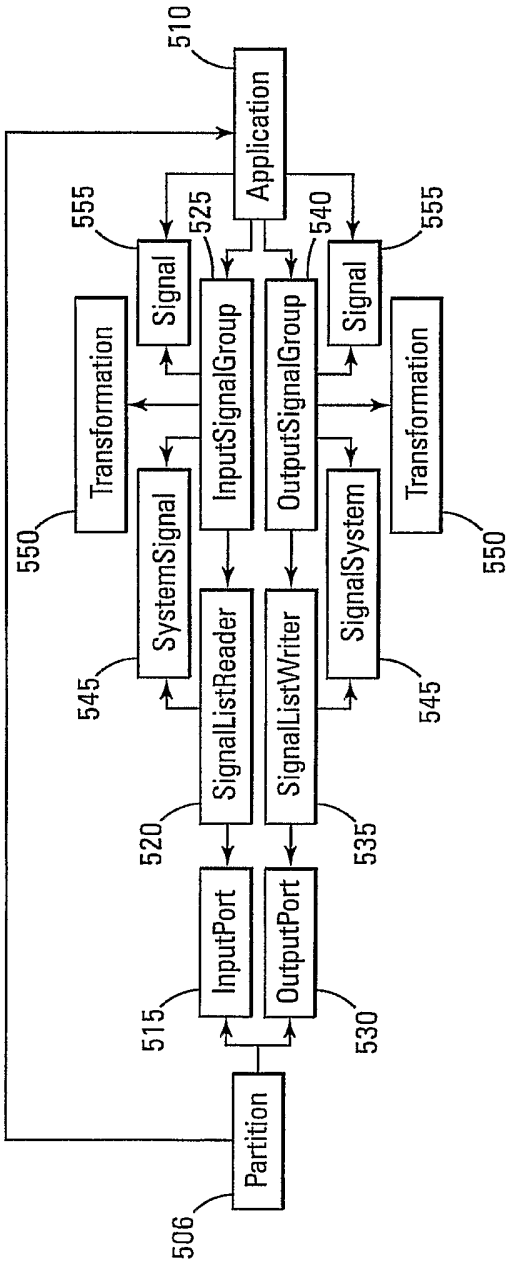

510
Application
555
525
540
555
Signal
InputSignalGroup
OutputSignalGroup
Signal
Transformation
Transformation
550
550
SystemSignal
SignalSystem
545
545
SignalListReader
SignalListWriter
520
535
InputPort
OutputPort
515
530
Partition
506

FIGURE 5

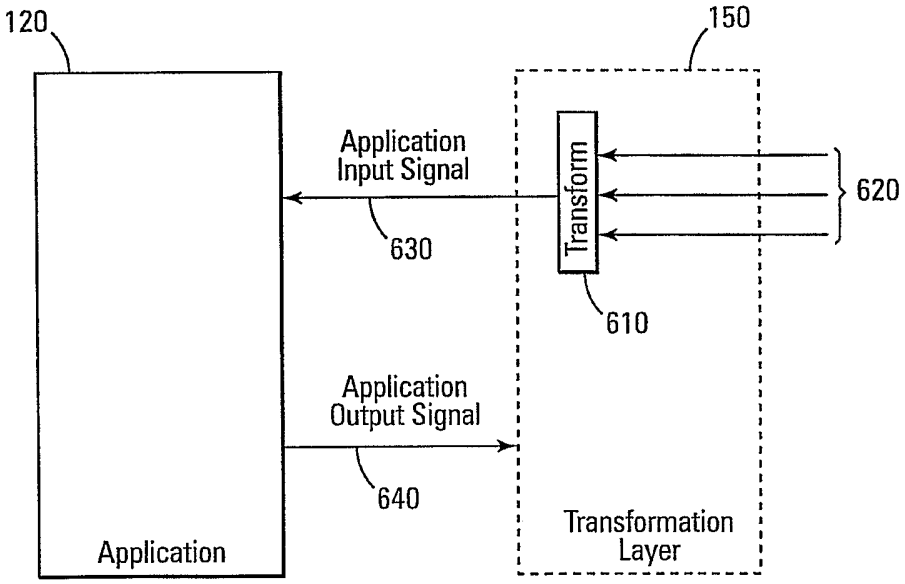

120
150
Application Input Signal
630
Transform
620
610
Application Output Signal
640
Application
Transformation Layer

FIGURE 6

INTERNATIONAL SEARCH REPORT

International Application No
IB/US2005/027719

A. CLASSIFICATION OF SUBJECT MATTER
G06F9/445 G06F9/455 G06F9/46

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

B. FIELDS SEARCHED

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

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

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, PAJ, WPI Data, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
	-/--	

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

[X] Patent family members are listed in annex.

° *Special categories of cited documents:*

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
9 December 2005	30/12/2005

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

6

Form PCT/ISA/210 (second sheet) (January 2004)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	SCHMIDT D C ET AL: "Patterns, fireworks, and middleware: their synergistic relationships" PROCEEDINGS OF THE 25TH. INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. ICSE 2003. PORTLAND, OR, MAY 3 - 10, 2003, INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, LOS ALAMITOS, CA : IEEE COMP. SOC, US, vol. CONF. 25, 3 May 2003 (2003-05-03), pages 694-704, XP010640174 ISBN: 0-7695-1877-X abstract page 695, right-hand column, section 2, paragraph 1 - page 696, left-hand column, paragraph 2 page 696, right-hand column, paragraph 3 - paragraph 5 page 697, section 3, paragraph 1 - paragraph 3 page 698, right-hand column, paragraph 5 page 699, section 5.1, paragraph 1 - page 700, left-hand column, paragraph 2 page 700, right-hand column, paragraph 2 - page 701, left-hand column, paragraph 3 page 701, left-hand column, last paragraph - right-hand column, paragraph 3 page 702, right-hand column, section 6, paragraph 1 - paragraph 2	1-10
X	US 2003/070061 A1 (WONG HOI LEE CANDY ET AL) 10 April 2003 (2003-04-10) paragraph '0044! - paragraph '0046! paragraph '0060! - paragraph '0064! paragraph '0069! paragraph '0076! paragraph '0097! - paragraph '0100! paragraph '0122! paragraph '0137! - paragraph '0141!	1-10

-/--

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US2005/027719

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	KARSAI G ET AL: "A modeling language and its supporting tools for avionics systems" 21TH. DASC. THE 21TH. DIGITAL AVIONICS SYSTEMS CONFERENCE PROCEEDINGS. IRVINE, CA, OCT. 27 - 31, 2002, DIGITAL AVIONICS SYSTEMS CONFERENCE, NEW YORK, NY : IEEE, US, vol. VOL. 1 OF 2. CONF. 21, 27 October 2002 (2002-10-27), pages 6a31-6a313, XP010616381 ISBN: 0-7803-7367-7 abstract page 6.A.3-3, right-hand column, paragraph 1- paragraph 4 page 6.A.3-7, left-hand column, paragraph 3 - page 6.A.3-8, left-hand column, paragraph 2 page 6.A.3-10, left-hand column, paragraph 2	1-10
A	DAVID FLANAGAN: "Java in a Nutshell: A Desktop Quick Reference" November 1999 (1999-11), O'REILLY & ASSOCIATES , USA , XP002357048 chapter 4, section 4.6., paragraph 1	6-10
A	ROBERT ROBSON: "Using the STL: The C++ Standard Template Library" February 1998 (1998-02), SPRINGER VERLAG NEW YORK INC. , 175 FIFTH AVENUE, NEW YORK, NY 10010, USA , XP002357049 ISBN: 0-387-98204-3 pages 213-217 - page 213, section 8.5, paragraph 1 - page 214, paragraph 1 -	6-10

6

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
T /US2005/027719

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2003070061 A1	10-04-2003	US 2003067485 A1	10-04-2003
		US 2003067489 A1	10-04-2003