PLATFORM FOR ADVANCED CONTROL APPLICATIONS

B. Horn*, J. Beran, J. Findejs**, V. Havlena**, M. Rozložník****

**Honeywell Process Solutions*
2500 W. Union Hills Drive, Phoenix, AZ 85027, USA
brian.horn@honeywell.com

***Honeywell Laboratories Prague*
Pod vodárenskou věží 4, 182 08 Prague, Czech Republic
jaroslav.beran@honeywell.com, jiri.findejs@honeywell.com,
vladimir.havlena@honeywell.com, mikulas.rozloznik@honeywell.com

Abstract: Control applications have many requirements not provided by commercial operating systems. This paper describes the characteristics and usage of an environment for hosting process-control applications, which is implemented on a commercial operating system. It is called Unified Real Time (URT) platform, and is intended for applications that are large or complex and that may involve dynamic configuration, flexible scheduling, complex organization, etc. This paper also demonstrates the structure of a typical Advanced Control Application (ACA) designed under URT.

Keywords: Cascade control, Software components, Control applications, Data flow, Process control, Real-time.

1. INTRODUCTION

The increasing power and reliability of commercially available computers and operating systems have been a boon to advanced control applications due to the relatively inexpensive and ever-increasing processing power and memory availability, as well as the ability to leverage operating system features such as memory management and priority execution.

Control applications have many requirements not provided by commercial operating systems. Several years ago Honeywell Advanced Process Control group undertook the implementation of a platform to run on MS Windows 2000 and successors that would provide execution services, data exposure, process data access, configuration, and other application needs in as general and flexible way as possible. Unlike many of the embedded applications (Passetti, 2002), the component software based solution (Szyperski, 1999) is using standard Component Object Model (COM) and Distributed COM (DCOM) technology (Grimes, 1997). The resultant platform is called *Unified Real Time (URT)*. The design took advantage of experience with Distributed Control System (DCS) applications, sharing many characteristics with advanced control applications, as well as experience with earlier platforms for advanced applications. The platform is primarily dedicated to advanced process control and real time optimisation applications (supervisory control layer) with sampling periods one second or more.

This paper describes the URT architecture and demonstrates the typical usage of this platform. It further contains the structure of a typical Advanced Control Application (ACA) designed under URT.

2. URT ARCHITECTURE

Control applications generally need to expose data to operators and engineers so that they can view what the application is doing and provide inputs. Equally important is the ability to pass data between modular pieces of an application, or between applications. Advanced applications may need to expose large amounts of data, which implies that flexible containers and structures are needed for the organization of data. At the same time, it should be simple to configure and work with applications that require only a small amount of data.

To meet these needs, data values that need to be exposed are encapsulated in component objects called data items. Data items have as small granularity as possible: each scalar value or each container (array, list, etc.) of values is a separate data item. The main properties of a data item are the type, the value(s) and the name.

A special data type comp, which can be thought of as a reference or a pointer to another component, is the basis of data organization in URT. Containers of type comp can be nested in any order to any depth to form a tree structure. The generality of this data tree provides the flexibility needed for large applications to organize their data in a simple way. The tree structure also scales well – it can range from just a single branch with a few data items to a large structure with many branches to organize a large amount of data.

A struct is a specialized list of comp whose elements can point to data items of different types. This is similar to the concept of a struct in C++ or a record in Pascal. In URT, the distinction between a struct and a list of comp is that the elements of a struct may be of different types whereas the elements of a list of comp should all be the same type (not enforced, but simplifies programming of function blocks).

In more detail, a struct is actually obtained from a container of type node list, which is the same as a list of comp with one additional property called the node type. The node type can be any string that describes the purpose of the node in the hierarchy. A struct is a node list with node type set equal to “struct”.

Function blocks and scheduling blocks are derived from node list. Their node list underpinning allows them to be included in the tree structure and allows them to organize their data underneath them.

Taking the idea of the tree structure organization to its logical conclusion, a URT platform consists of a single tree structure that contains the schedulers, function blocks, and data items for the application instances that have been configured on that platform. Typically, each application instance is a branch off the top node of its platform (although organizational nodes may be at the top of very large applications), and typically a scheduling block heads this branch. The function blocks and data items that comprise the

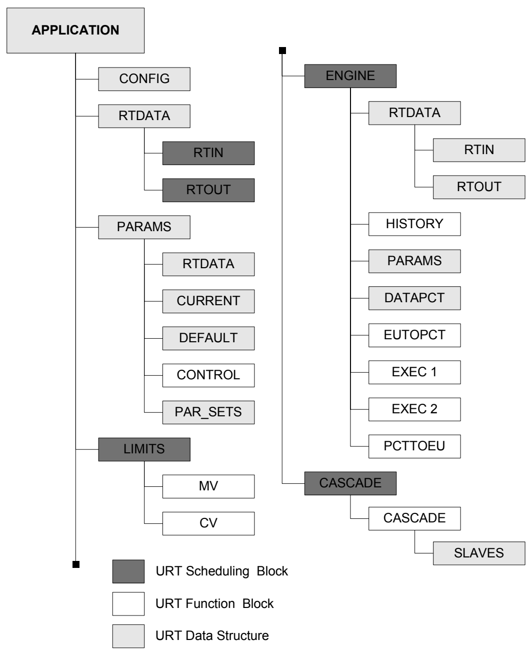


Fig. 1. The ACA Structure

application instance are in one or more branches under the scheduling block. There are some constraints imposed on the organization by the way in which the execution is propagated, but an application designer has a great deal of flexibility in deciding how to organize the scheduling, functional, and data components for the application.

To back up just a bit, a URT platform tree is composed of components called tree members. A tree member component is defined as a component that supports a few minimal interfaces that allow it to be part of a URT tree. In particular, a tree member must support an attachment to a parent, and any node that is not a leaf node must also support attachments to children. Data items, function blocks, and scheduling blocks are derived from classes that support these requisite tree member interfaces.

For unique needs, the component nature of tree members allows easy extension to URT’s capabilities – additional types of data items and scheduling blocks can be readily implemented. Of course, new types of applications will usually require writing one or more function blocks specific to that application type.

3. ADVANCED CONTROL APPLICATION (ACA) EXAMPLE – DATA STRUCTURE

Figure 1 illustrates a URT platform tree for a typical advanced control application. The application reads data from a process or underlying control system, executes a control algorithm, and writes the result back to the process or to the set points of basic control loops. The application behaviour is parameterized by a set of parameters that can be changed in run-time. Besides the main control algorithm, other tasks can run simultaneously providing data for operators, another applications or the main control algorithm.

The platform tree in this example consists of the six tree members:

CONFIG – The structure consisting of the application configuration parameters, which cannot be changed in the run-time.

RTDATA – The structure consisting of the real time application data (input and output). The data can be read from or written to the OPC (OLE for Process Control) server (remote process or application).

PARAMS – The structure consisting of all application tuning parameters. Parameters are organized in parameter sets. The sets are switched on demand in real-time in a thread-safe manner. Some data in a set are replaced by real-time data.

LIMITS – The scheduling module that calculates the actual limits of the manipulated and controlled variables. This is an example of a support task.

ENGINE – The scheduling module that implements the main control algorithm. Implementation of this algorithm is distributed to several function blocks (EXEC1, EXEC2)

CASCADE – The scheduling module that handles the application coordination in the cascade. Independent implementation of a connection to a master application and to slave applications enables creating control cascades from different applications easily. A control cascade can span across several platforms (computers).

Important features of control applications are modularity and extensibility. Control algorithms are further developed and it is essential to be able to change an algorithm as easily as possible. The decomposition of a control application to URT function blocks fulfils this requirement very well. Function blocks with the same data items can be exchanged without any problem. In the same way, it is easy to add a pre- or post-processing functionality to an existing application by adding a new function block to an appropriate place in the URT platform.

4. DATA FLOW

4.1 References (links)

The link type provides a reference to another data item in the tree. Access methods for the value property are overloaded so that the value can be read and written as the name of the target data item. Internally, the target name is resolved to a pointer whenever it is changed. A function block can use the link data item as a reference from which the target data item and data items in the branch below the target can be easily and efficiently accessed. This can be used to provide dynamic indirection.

The link and comp types are similar in that they provide a reference to another data item. The essential difference is that the comp type defines a parent-child relationship within the platform tree, which implies that entering a non-NULL value means creation and attachment of a new child, whereas the link type defines a reference to an existing component.

4.2 Connections

Every data item may have an input and/or output connection configured. A connection reads or writes the data item’s value from some target. There are two basic types of connections, internal and external; the target of an internal connection is another data item (in the same platform, or on a different URT platform, either local or remote), and the target of an external connection is a value in another system, typically a DCS.

External. External connections provide the means to input and output process data from DCSs or other types of process-connected systems. At present, external connections use the OPC standard, but external connections are designed for easy extensibility to different APIs for access to servers that do not support OPC.

URT provides an OPC Client function block that can be inserted in applications along with function blocks that use input process data and generate output process data. An OPC Client function block receives a configuration-change event call-back when an external connection is (re)configured, and if the data item is within its scope, OPC Client adjusts its OPC groups accordingly. OPC Client reads inputs just prior to execution of function blocks that use the inputs, and writes outputs just after execution of function blocks that generate the outputs.

Internal. Internal connections allow easy connection between inputs and outputs of function blocks that provide generic functionality. Typically, a function block exposes data items for input and output values. A connection on an input data item can be configured to pull its value from the output data item of another function block, or a connection on an output data item can be configured to push the value to the input data item of another function block.

5. ADVANCED CONTROL APPLICATION (ACA) EXAMPLE – DATA FLOW

Data flow in an application is shown in Figures 2 and 3. Two types of data sharing are distinguished – connections (solid line) and links (broken line).

5.1 Real-time input data

Real-time data read from a DCS using external OPC connections are stored in defined structures. Function blocks copy required data using internal connections, or data items shared by several function blocks under the same scheduler are copied to common data structures. A different approach can be used for, e.g., storing data to function block 'history', where the operation of storing values and time-stamps is executed in context of data reading.

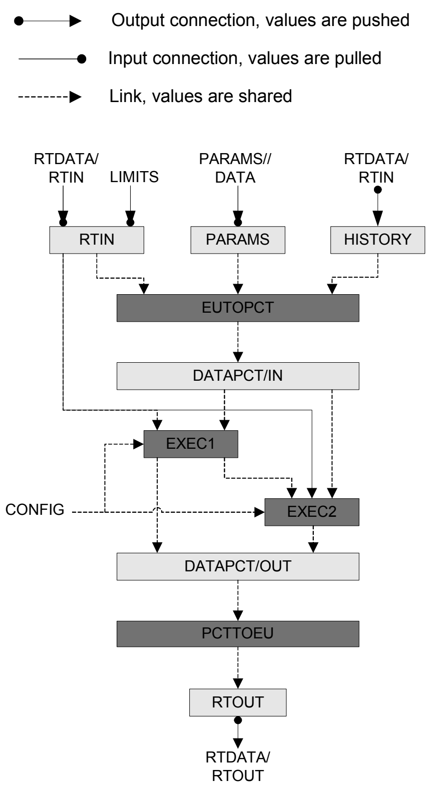


Fig. 2. The engine module data flow.

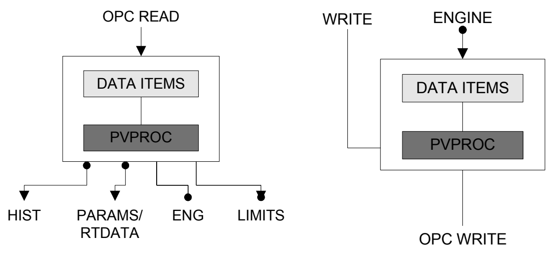


Fig. 3. The RTIN and RTOUT module data flow.

5.2 Application parameters

A control application has typically several sets of tuning parameters influencing its behaviour. Just one of these sets of parameters is active and a special mechanism is implemented to switch among the sets. These data are copied using connections. More details about the organization of the sets and the switching mechanism are described below.

5.3 Real-time output data

Real-time output data written to DCS using external OPC connections are stored similarly to real-time input data. Function blocks copy their results to output data using output internal connections. Writing to the OPC server is aperiodically scheduled by the application when data is ready.

5.4 OPC details

Data from the process and other external sources are delivered to a URT platform via an OPC Client function block. This block allows getting data from several OPC servers. Data retrieval period is given by the scheduling module that the OPC function

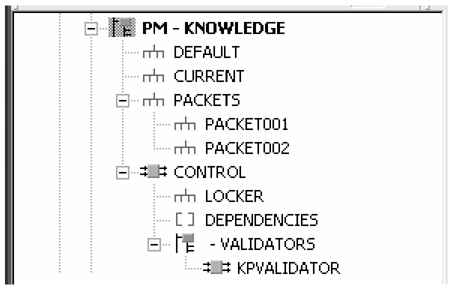


Fig. 4. The packet manager structure.

block is executed in. All OPC inputs are executed in one or more RTIN schedulers independently of the rest of the application. This improves the application stability when the time for reading is comparable to the sampling period. When there are more groups of input data with different sampling periods, one RTIN scheduler for each data group is created. All RTIN schedulers run periodically.

Data are delivered to the external process by the same OPC Client function block. Output data are located under one or more RTOUT schedulers. Again, OPC outputs are executed independently, not to influence the rest of the application. RTOUT schedulers run periodically or on-demand when all output data are available. One RTOUT scheduler typically consists of a data group that must be written in one pass.

5.5 Parameters organization

Each control application has a set of tuning parameters affecting its behaviour. It is very convenient for the application tuning, if this set can be switched to another one by means of a special mechanism.

The set of tuning parameters is called a packet. The mechanism designed to switch the packets is called a packet manager.

The packet manager consists of the following parts (see Fig. 4):

- default packet – default packet to create a clone
- current packet – the currently used packet of the tuning parameters
- list of packets – the list of all packets which can be selected as current packet
- packet manager control – the function block to support packet management functions

The packet manager control function block provides packet management functions including creation, selection, validation and removal of the packets. It has to guarantee access synchronization for multiple clients (e.g. human-machine-interface applications) and consistency of data in the current packet.

The content of the current packet is changed using the packet manager control select function. Packets are created as a clone of the default packet using the create function. Content of a packet has to be validated to ensure the packet data consistency. For the process of validation, a set of function blocks can be specified providing a concrete packet structure validation. The packet manager executes them on the validation function request.

6. APPLICATION EXECUTION

Scheduler blocks perform execution. A scheduler block typically heads a branch that contains the function blocks and data items for an application instance. This branch is referred to as a scheduling module. Each scheduler block exposes child data items that hold scheduling information such as desired execution interval, interval offset, demand trigger, execution priority, and active/inactive status.

6.1 Execution thread

A scheduler block creates an execution thread when the block is activated. This thread lives as long as the scheduler remains active (which is indefinitely long for typical continuous-control applications). Execution of all the components under the scheduler block occurs on this thread.

Use of a separate thread for each scheduling module, which typically means for each application instance, makes it possible for applications with widely different execution frequencies and processing requirements to coexist peacefully. Applications with short intervals and modest processing loads can be assigned higher priorities than applications with long intervals and heavy processing loads.

6.2 Commands

Commands are used to tell tree member components when they should carry out various kinds of tasks. A command is a method implemented by each tree member with children. The minimal implementation is to call the same method on each child. Beyond that minimal requirement, a component is free to do whatever it deems appropriate in response to the particular command. Once a command is executed on a particular node, the command will propagate to all nodes on the branch below the initiating node. Along the way, each component gets to do whatever is appropriate for the particular command.

Execution Commands. When a scheduling module is executed, either at a scheduled interval or because of a demand, the scheduling block sends a series of commands to its branch. An execution cycle consists of the following three commands carried out in sequence: PreExecute, Execute and PostExecute. Application function blocks typically perform their main functionality in Execute. OPC Client function block and data items with connections use PreExecute and PostExecute such that inputs are read just prior to use and outputs are sent just after they are generated.

Notification Commands. A number of other commands are issued by the scheduler and by other components, or by external clients. These include:

- InitNew – Issued on a component when it is added to a platform. A component can set its initial value.
- Save – Issued on the top node to persist a platform. Each node writes all its own properties, in an eXtensible Markup Language (XML) node that is a child of its parent's XML item.
- Load – Issued on the top node to load a platform from persistent storage. Each node is recreated and attached by its parent and then loads its own properties.
- OnPostBuild – Issued on any node to indicate that configuration activity has occurred on the branch below the node. Data items check that connection targets are still valid. Function blocks can check whether they need to respond to configuration changes.

6.3 Cross platform execution

The execution control based on commands is a fast and effective system for controlling execution inside one platform. Controlling execution of several applications running on several platforms, possibly on several machines, has different issues and priorities. The special function block 'CASCADE' covers requirements of controlling a cascade of several control applications running in *1:n* relationship, one master application can manage *n* slave applications in parallel.

In addition to the commands, 'CASCADE' function block uses data connections (links, URT connections or OPC connections) for controlling slave applications. Two data commands are issued by the master application's PreExec and Execute.

The PreExecute command is first passed to all slaves, than it waits while all slaves finish the PreExec phase, and finally the application executes its own PreExec procedure. It means the PreExec is actually processed from the bottom of the cascade to the top.

The Execute command is processed from the top to the bottom of the cascade. All slave applications are started simultaneously as they can run on different processors. Slave synchronization is not provided.

7. APPLICATION BUILDING

Creation of a control application tree structure is a complex and time-consuming task. The structure depends on configuration parameters, i.e. the number of manipulated variables, controlled variables, etc. Therefore a maintenance function block is created for each type of control application. The function block has the following features and responsibilities:

- contains information about the tree structure
- creates or updates the tree according to the application configuration parameters
- creates link data items and connections to shared data between function blocks
- sets default values of selected data items

- updates variable ranges
- calls PostBuild() method on application root node to reconnect all data items

The maintenance function block can be considered an object, which stores parameterized information of the application structure with methods allowing application (re)building and managing.

8. REAL TIME RECONFIGURATION

The ability to create and modify a URT platform from the platform itself together with indirection (links) allows implementing advanced features like real time configuration switching, on-demand reconfiguration or off-line analyses.

The real time configuration switching is a typical requirement when some process value is changing its control role in real time, e.g. a manipulated value becomes a measured disturbance value and vice versa. In such a case, one controller for each scenario is configured and tuned in a URT platform. Input/output data and execution are then switched in real-time to the new controller by changing one link.

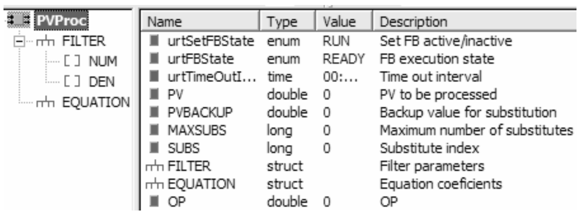
Example: Master controller output is configured as indirect link: MASTER.OP = /SELECTOR#/SP, where link SELECTOR can be one of {SLAVE1, SLAVE2}. The resulting connection for master output is /SLAVE1/SP or /SLAVE2/SP depending on actual SELECTOR value.

If there are many possible configurations and the selection involves a complex decision-making, the required configuration can be generated in real-time on a low priority thread. The execution is switched to the new configuration when the latter is available and ready to use.

Most of today's advanced control applications provide, in addition to their main control functionality, also some what-if analyses. These analyses require the control algorithm to run on a data set where the process data are partially replaced by data supplied by a client. A simple and elegant solution in the URT platform is cloning the corresponding part of a controller function and connecting the particular client to the cloned part. The client can modify inputs and see possible outputs without changing the current control strategy.

9. MESSAGES AND OPC EVENTS

All components, including function blocks, have use of a messaging system to issue messages for operator and engineer viewing and acknowledgement. The messaging system takes care of such things as filtering out redundant messages (e.g., when the same error condition is reported for a number of successive intervals) and clearing messages when a condition is no longer in effect.



Name	Type	Value	Description
urtSetFBState	enum	RUN	Set FB active/inactive
urtFBState	enum	READY	FB execution state
urtTimeOutI...	time	00:...	Time out interval
PV	double	0	PV to be processed
PVBACKUP	double	0	Backup value for substitution
MAXSUBS	long	0	Maximum number of substitutes
SUBS	long	0	Substitute index
FILTER	struct		Filter parameters
EQUATION	struct		Equation coeficients
OP	double	0	OP

Fig. 5. Function block for PV processing

The text of a message may be generated in any manner, but a convenient mechanism provided by the message system is the use of XML files to hold the raw message text. XML attributes may be used to specify message attributes such as severity, acknowledgement, category, and many others. To issue a message, a component selects the message by specifying a handle attribute, and provides non-static information in the form of strings to be inserted into the message text at marked points.

The message system is integrated with URT's OPC Alarm & Event Server. Issued messages are transformed into OPC A&E events. The events can be integrated with process system events for display to operators and for operator action if the process system has a suitable OPC client.

10. PV PROCESSING BLOCK EXAMPLE

An example of a function block, which processes PV (can be either a single or double value or an array of double values) and calculates OP (the same type and size as PV) using both PV and OP range, filter and equation parameters is shown in Figure 5.

11. CONCLUSION

Overall performance of the platform is limited by the underlying operating system (Windows 2000). The platform overhead resulting from the COM technology used is insignificant and comparable to DLL calls. Scheduling accuracy is around 10 ms.

The URT platform itself does not implement any control algorithm, but currently some Honeywell controllers, e.g. for the petrochemical and power markets, have been ported to URT platform. No significant performance degradation compared to previous Windows native applications was observed during testing.

REFERENCES

Grimes, R. (1997). *Professional DCOM Programming*. Wrox, Birmingham, Canada.

Passetti, A. (2002). *Software Frameworks and Embedded Control Systems*. Springer, Berlin.

Szyperski. C. (1999). *Component software – Beyond Object-Oriented Programming*. Addison-Wesley, Harlow.