View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Lund University Publications - Student Papers

ISSN 0280-5316
ISRN LUTFD2/TFRT--5824--SE

Visualization of task execution in ABB 800xA

Martin Furmanski

Department of Automatic Control
Lund University
September 2008

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
September 2008

Document Number
ISRN LUTFD2/TFRT---5824--SE

Author(s)
Martin Furmanski

Supervisor
Gert-Ola Carlsson and Peter L. Nilsson ABB Process Automation, Malmö
Karl-Erik Årzén Automatic Control (Examiner)

Sponsoring organization

Title and subtitle
Visualization of task execution in ABB 800xA (Visualisering av exkvering i ABB 800xA)

Abstract
The 1131-tasks that execute within the 800xA controller have been visualized by implementing three components extending the functionality of the 800xA control and automation platform. The components named TaskVisualizerCommServer (TSCS) and TaskVisualizerCommClient (TSCC) have been added to the controller and control builder respectively and are responsible for data collection and networking. The third component named Visualizer is a stand alone component handling graphical presentation and associated data handling.

Keyword
task visualization, task schedule, ABB system 800xAs

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
84

Recipient's notes

Security classification

http://www.control.lth.se/publications/

Preface

This thesis was carried out at ABB Process Automation in Malmö. The responsibilities of this company ranges over the full spectrum needed for their main product line, system 800xA, from sales-related/administrative to the more interesting parts of actual product implementation. The idea for the thesis originated both from company internal and external wishes of gaining further insight into perhaps the most fundamental and central entity of the system, the task execution.

This report details a very practically oriented master's thesis. Most of the work has been in understanding, analyzing and amending the works of others, being in the form of program code. Plenty of documentation has been sifted through in efforts of better understanding the system whose functionality has been extended, and although helpful in generalities, for specifics mostly the code itself has been analyzed with the much needed aid of my supervisors to at least point me in the right direction. A few words spoken by a helpful colleague by proximity describes the experienced situation and also a proposed way of approaching any difficulty faced; "the code is the truth".

Although not a great work of creativity, even what might seem as simple choices have all been made under scrutinized evaluation of possible alternatives. And with the project itself being founded in a very practical and world-bound wish from the inventors/initiators, all choices have been made under just this eye of real world application.

Allow me here to also express a big thanks to my supervisors and all the nice and helpful people who surrounded me at my workplace.

TABLE OF CONTENTS

1 INTRODUCTION **5**

2 BACKGROUND **6**

2.1 SYSTEM 800XA 6
2.1.1 The Process Module (PM) *6*
2.1.2 The Control Builder (CB) *7*
2.1.3 Components of the executive system *8*
2.1.4 Networking capabilities *10*
2.2 RELATED WORK 11

3 FUNCTIONS OF THE TASK VISUALIZATION **15**

3.1 TASK SCHEDULE 15
3.2 GRAPHICAL PRESENTATION FORMAT 15
3.3 USAGE SCENARIOS 17
3.4 DATA COLLECTION 17
3.5 STARTING A DATA COLLECTION 18
3.6 AUXILIARY FUNCTIONS OF THE GRAPHICAL VIEW 19

4 IMPLEMENTATION **21**

4.1 MAIN COMPONENTS AND MODIFICATIONS 22
4.1.1 TaskScheduleCommServer (TSCS) *23*
4.1.2 TaskScheduleCommClient (TSCC) *25*
4.1.3 Visualizer *26*
4.2 PROTOCOLS AND DATA PACKETS 28
4.2.1 TSCC to TSCS *28*
4.2.2 TSCS to TSCC *29*

5 EVALUATION AND FUTURE WORK **31**

6 APPENDIX **33**

6.1 SOURCE CODE TSCS 33
6.2 SOURCE CODE TSCC 47
6.3 SOURCE CODE VISUALIZER 60

7 REFERENCES **84**

1 Introduction

Simplifying the system 800xA to a simple scenario suitable for understanding the gist of this thesis, we have a number of tasks and these tasks execute in a not fully predictable way. The different tasks are developed by a user writing code and tuning task parameters for each one of them. This set of tasks can later be downloaded to a controller which starts executing the code using the timing information specified in the task parameters.

The data collection and graphical visualization presented in this paper can then be used to present a view of how the tasks executed, post-execution. This is done in three stages; data collection, data transferring and data visualization. Data collection is done in the controller and the data is then sent over the network to the visualization tool.

This functionality is implemented by amending the code base for both the controller and for the controller operating and task developing tool, the Control Builder, as well as by adding an additional component responsible for visualization.

The text that follows is organized as to firstly introduce the existing 800xA system in a suitable level of detail and relevance and secondly to describe the added visualization functionality from a user perspective. Finally the text delves into the most important implementation issues leaving less interesting details to be found in the source code appendix.

2 Background

In this section all the relevant parts of 800xA will be described. This is needed as a basis for the chapters that follow in the same manner as it was needed for the work during the production of this thesis.

2.1 System 800xA

Following a line of predecessors, the current control and automation platform goes under the collective name of Advant-800xA. Within this platform the parts of essence here are the process module and the control builder. These are the two main components used in development and deployment of control applications by system users.

2.1.1 The Process Module (PM)

At the core of executing the control and automation routines is the process module. A picture of a process module surrounded by a few other modules can be seen in Figure 2.1. The process module is a CPU-based device with firmware, RAM and I/O-capabilities. The firmware grants the services available to control routines, as programmed by a developer. A specific version of a firmware exports an interface which can not change unless the firmware version also changes.

Figure 2.1: A process module on a rail accompanying other modules.

The firmware contains the operating system on which the rest of the firmware and above layers execute. In other words, the bottom layer is a VxWorks OS and in this OS low-level services are implemented as for example scheduling and memory management. This

scheduling is not directly related to the scheduling of user-made control applications, as explained in the sequel.

Scheduling thread

The control routines are downloaded to the online process module and stored in its RAM. When executing these control routines their scheduling is handled by functionality present in a low-level(VxWorks) thread, simply named the scheduler thread, or even simpler the scheduler.

1131-tasks

The scheduler executes control routines which are most commonly referred to as 1131-tasks. This name comes from a specification for a general automation platform called IEC 61131-3. The tasks can be programmed in a multitude of different languages, defined in the same specification, with the main language being structured text (ST). This is a programming language similar to many others, where a list of statements are entered and when executed they execute in order. Also included are operators for common program control logic, e.g. for- and if-statements and their variants.

Soft controller

There is also a "pure software" version of the controller, in contrast to the hardware-based one. This controller runs on the Windows platform, and most parts of the firmware source code are identical between these versions. Although referred to as a "pure software" controller, it of course also runs on hardware in the end, but being a different platform on at least a couple of layers, there will be necessary differences in source code when dealing with these layers.

Nevertheless, despite the differences, during development and implementation, the use of this soft-controller is a very convenient way of working and testing new functionality. Any modified firmware thus need not be downloaded to a real hardware controller, and real-time debugging possibilities within the firmware code using standard Windows developmental tools is granted.

Any final results however have been tested on both the hardware and soft controller.

2.1.2 The Control Builder (CB)

Measuring this components code base against that of the process builder it is much larger. Its functionality, however, can be summarized very simply: it is an engineering tool for programming and interfacing with the process module. The code for the 1131-tasks are created in this tool and also uploaded to the process module using this tool. When using the tool one is usually working on a project which entails the control routines and their

connection to the process module. An open project can be seen in the Control Builders project view in Figure 2.2.

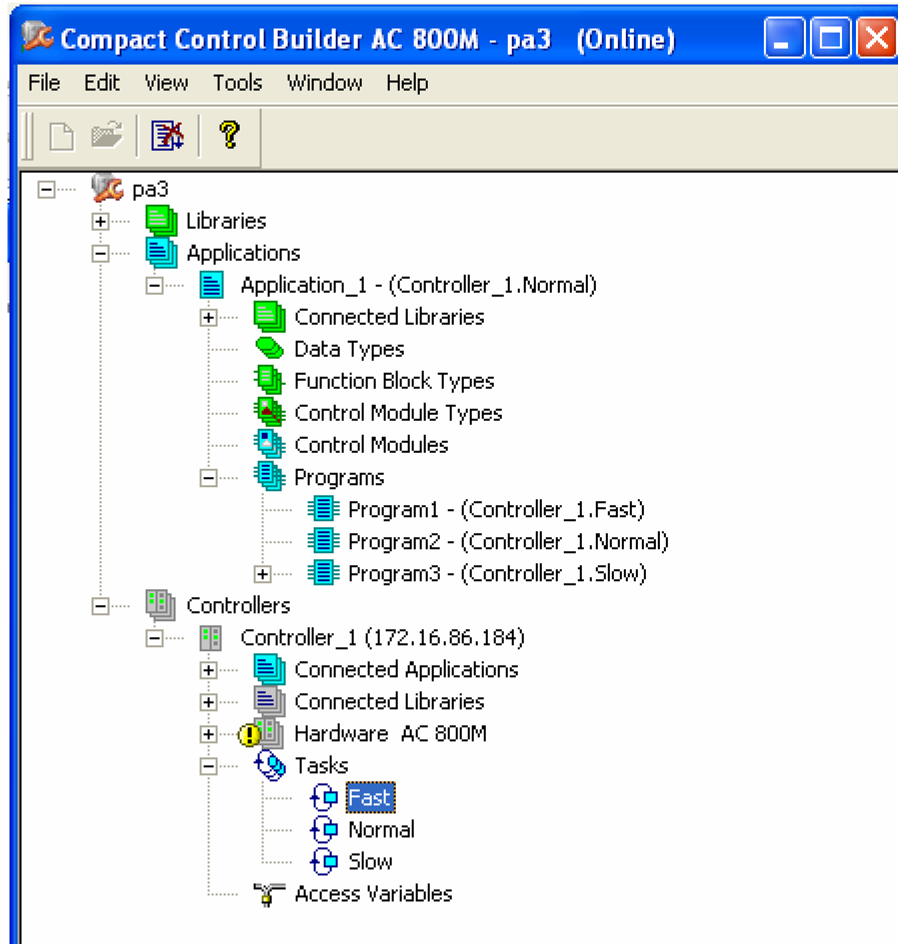

Figure 2.2: An open project in the Control Builder.

The important things to notice are how there are tasks defined under a controller and how the programs are assigned to the tasks using their names. The relations and restrictions will be made clearer in the next section. It can also be seen in the above figure that the controller is identified using an IP-address. Double-clicking on one of the programs would let one edit the source code for it. Double-clicking a task (e.g. Fast, Normal, Slow in the above figure) would let one view and alter task properties. From the menu-bar a command can be chosen to download the programs and task-settings to the controller. Just editing the source code does not automatically make it change whatever is executing inside the connected controller, stated just to make this point clear.

2.1.3 Components of the executive system

The different terms of this executive system and their interrelations will be defined for further insight.

Programs

In our context a program is defined as a sequence of statements to be executed sequentially. Within the set of possible program statements there exist the classical feature of jumps using conditional logic and repetition of statements, exemplified by if- and for-statements. Nevertheless any one execution of a program can still be unrolled to a sequential list of statements and hence our definition is sufficient for our purposes.

Other ways of defining a program are available within the Control Builder, notably some graphically aided languages with states and transition conditions. All are however in the end translated to the ST-format, as part of the compilation for downloading it to a controller.

Applications

Owing to the 61131 specification, an application is defined as an ordered collection of one or more programs. The ordering will be important for the task connection defined below. Any program is part of one and only one application.

Tasks

A task is defined as one or more programs to be executed by the scheduler, driven either by the tasks periodic timing properties or by events signaling the task to start. In the case of several programs within one task, they are executed in the order that they have in the application that they are a part of. For this to make sense one might realize that a task can only contain programs from a specific application, since the order was defined per application.

Every periodic task carries with it the time properties of period and offset, which together with the starting time of the whole scheduling process, forms a schedule of when each task should start preferably. A task should preferably start at these periodic instants, but owing to the interruption from other tasks this starting time could be delayed. The periodic task will never be allowed to execute earlier than the starting time, however.

The term task will be interchangeably used with that of 1131-tasks, unless otherwise stated as low-level tasks, signifying the lower level of the RTOS.

Scans

A single execution of a task's programs is called a scan. We will here not make any further distinction of this and simply assume that if two programs were coupled under the same task, it was just for the reason of coupling, and no distinctions will be made on a program basis. Hence the semantics of a start or stop of a scan will be interchangeably used with the start or stop of task, even though semantically a task will never stop (short

of a hardware reset), since it is a continuously present entity whose programs execute periodically.

In a real world application a scan usually implies reading and writing of I/O from sensors and to actuators. Since this has no direct relevance to the functionality developed within this thesis, we will not delve into the matter further.

Scheduling of tasks

As mentioned earlier there is a thread within the VxWorks RTOS which handles the scheduling of tasks, named the scheduler-thread. It does this with respect to their preferred starting times and priorities. Task properties consist of:

- Priority: Each task carries with it a user-chosen priority identified by an integer from 0 to 5. The level 0 signifies the highest priority and there can be at most one task with this priority assigned, named the "Time Critical" task. The levels 1 to 5 are of ascending priority. Priorities can not be changed online, i.e. during execution.

- Period: Every task gets a user-chosen period quantifying the time between two task starts, in the ideal case. Being a dynamic system the actual period will jitter and this is also a major indication for implementing this thesis's task visualization. Problems faced in online scheduling will be covered further down.

- Offset: To shift the start times of different tasks relative each other a period is chosen per task. This can be used to avoid colliding start times and optimizing a schedule.

Task switches

With the above task properties the continuous operation of the scheduler thread switches between 1131-tasks, whenever a higher priority task wants to execute it should be allowed to do so. This switching of tasks implies the pausing of one task and the continued execution of another one. The code of 1131-tasks is executed within the scheduler-thread, with exception of the Time Critical task which executes in its own RTOS-thread. Hence the Time Critical tasks time allocation is guaranteed by the RTOS while the other 1131-tasks are switched by functionality within the scheduler and the scheduler-thread. The switches can only occur at specific breaking points which get automatically placed throughout the 1131-code during compilation.

2.1.4 Networking capabilities

The connection between a controller and the Control Builder can have many different configurations and this field it is not investigated within this thesis. All communication has been set up as standard TCP/IP over Ethernet. Configuring the IP and type of the

controller within the Control Builder is basically enough to gain a connection. The capability of a connection is basically a presumed prerequisite for the functionality developed and is not used in any special way, but it is mentioned for the sake of clarity and completeness.

MMS

Owing to the generality and standards conformity of 800xA it supports the Manufacturing Message Specification. This is not central for the following description but some use of MMS-components can be seen in the source code in the appendix.

2.2 Related work

A few projects also offering within them task visualization functionality will be presented in this section.

RTSIM

In the paper "A tool for simulation and fast prototyping of embedded control systems" (Luigi Palopoli, Luca Abeni, Marco Di Natale, Paolo Ancilotti, Retis Lab, Scuola Superiore, S. Anna, Fabio Conticelli, 2001) a set of C++ libraries are presented that can be used for simulating continuous plants together with a real-time embedded control system. Based on a discrete event model, the simulated entities of the control system exhibit correct states at these event instants, as would have been observed had it not been a simulation, with the amount of correctness of course depending on how well the situation is modeled.

Simulated entities could be kernels implementing different scheduling policies and network technologies like CAN or Ethernet. Another relevant entity is the real-time task, characterized by four events: activation, schedule, suspension and end of the job. Within the RTLIB of RTSIM comes a Java-package capable of graphically presenting an execution trace of events and similarly as with the visualization in this paper, the events surrounding tasks are visible for all tasks by drawing blocks along a timeline, representing the current states. See the figure below, which comes from the above mentioned paper.

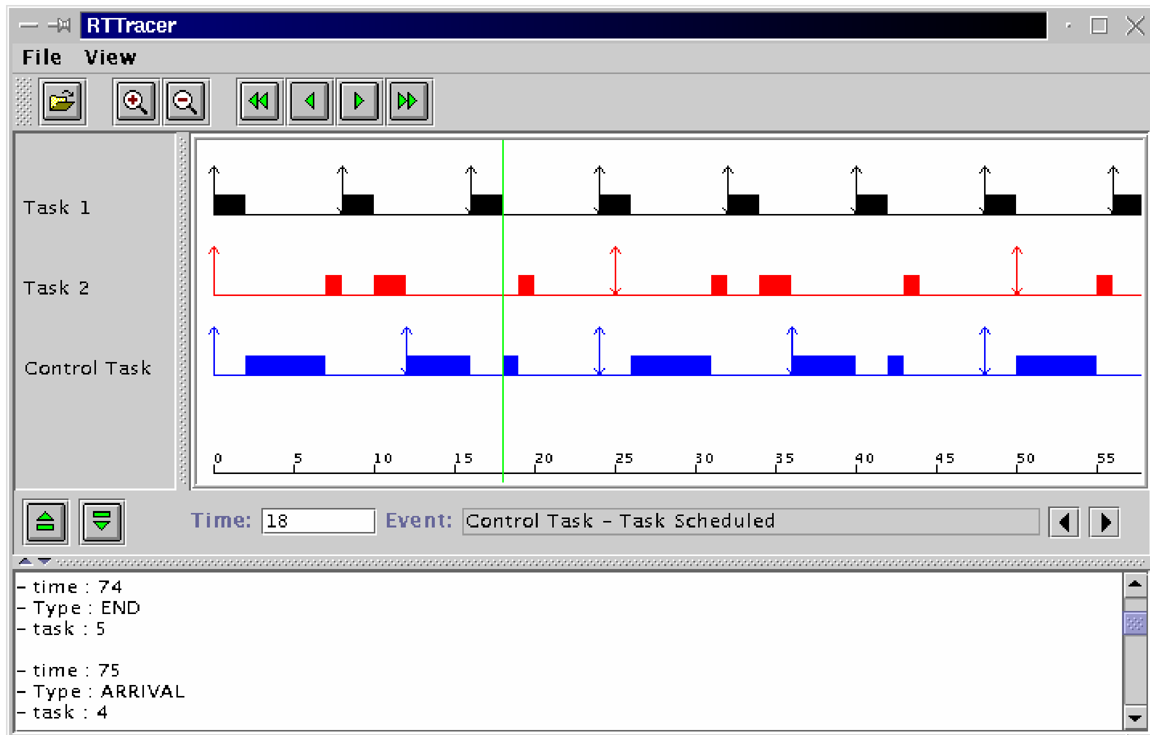

Figure 2.3: RTLIBs visualization tool, RTTracer.

TrueTime

TrueTime, presented in "How does control timing affect performance?" (Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, Karl-Erik Årzén, 2003), is a MATLAB/Simulink-based addition and concerns itself with real-time aspect of tasks in real-time kernels connected by networks and to continuous plants as modeled in Simulink. The Simulink-blocks representing kernels can be monitored in "real-time" using standard scope-blocks and the states of its internal tasks easily viewed. Building on the very flexible environment of Simulink, TrueTime makes it possible to mix simulated and non-simulated parts and leaves the system open to much modification, especially after being released under the GNU General Public License.
An example that comes with the TrueTime 1.5 package can be seen in Figure 2.4. The schedule view represents two tasks executing periodically. The blocking of the upper task is drawn by a halfway excited line (0.25 above base for that task) while a fully excited line (0.5 above base) represents active execution.

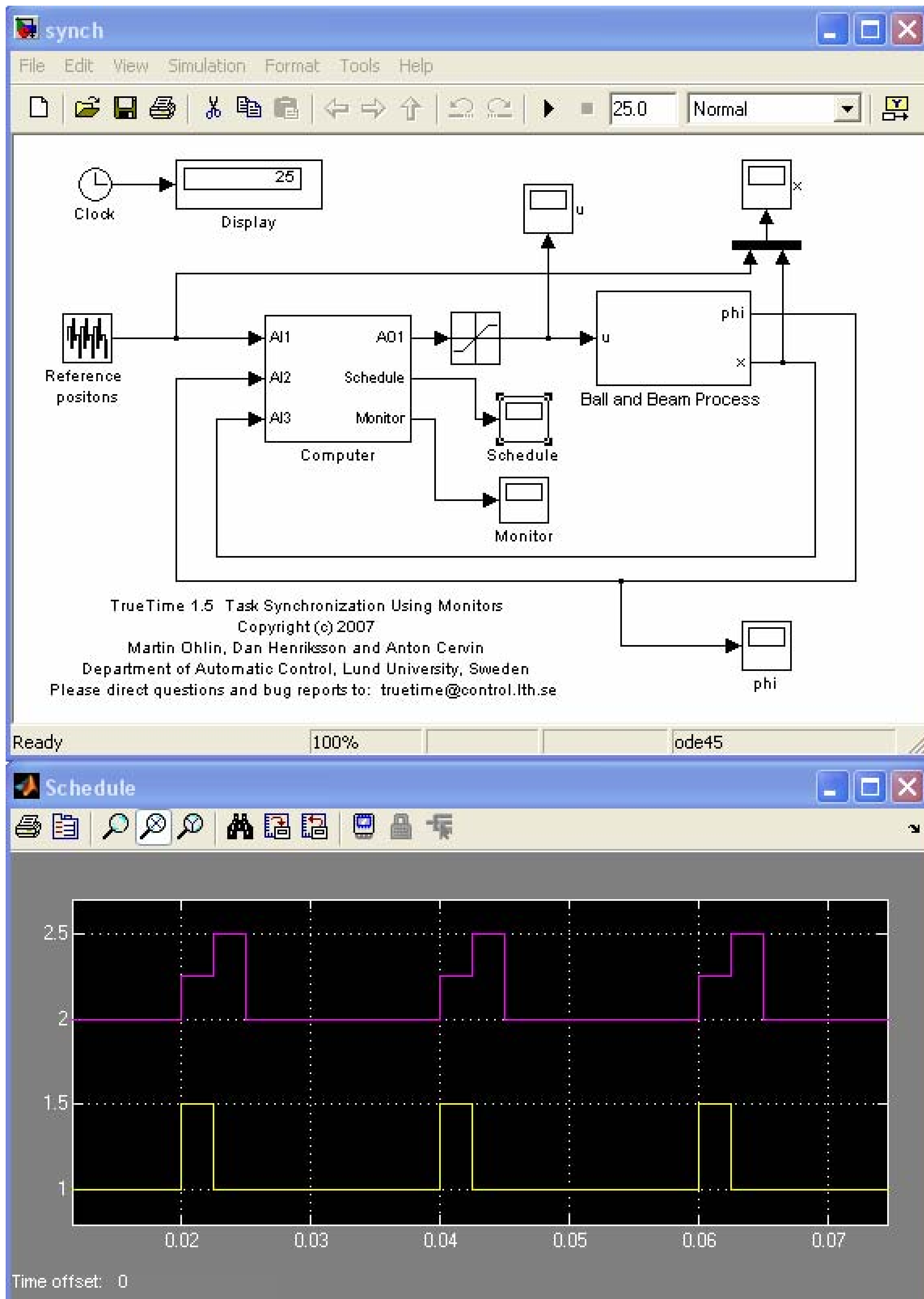

Figure 2.4: Synch example from TrueTime.

Wind River System Viewer

Wind River Workbench 3.0 (http://www.windriver.com/products/product-notes/Workbench-Tech-Note.pdf) is a collection of tools used to develop for the Wind River Linux and VxWorks platforms. A tool within this package is the System Viewer allowing real-time inspection of, among other things, tasks and their context switches. The dynamics surrounding the context switches can be monitored by inspecting other system events as for example semaphores, message queues, signals, timers and user events. The above paper suggest its use in dealing with problems like dead locks, starvation, race conditions, priority settings, resource contention and timing problems due to interrupt and task interaction.

The image below is from the paper above.

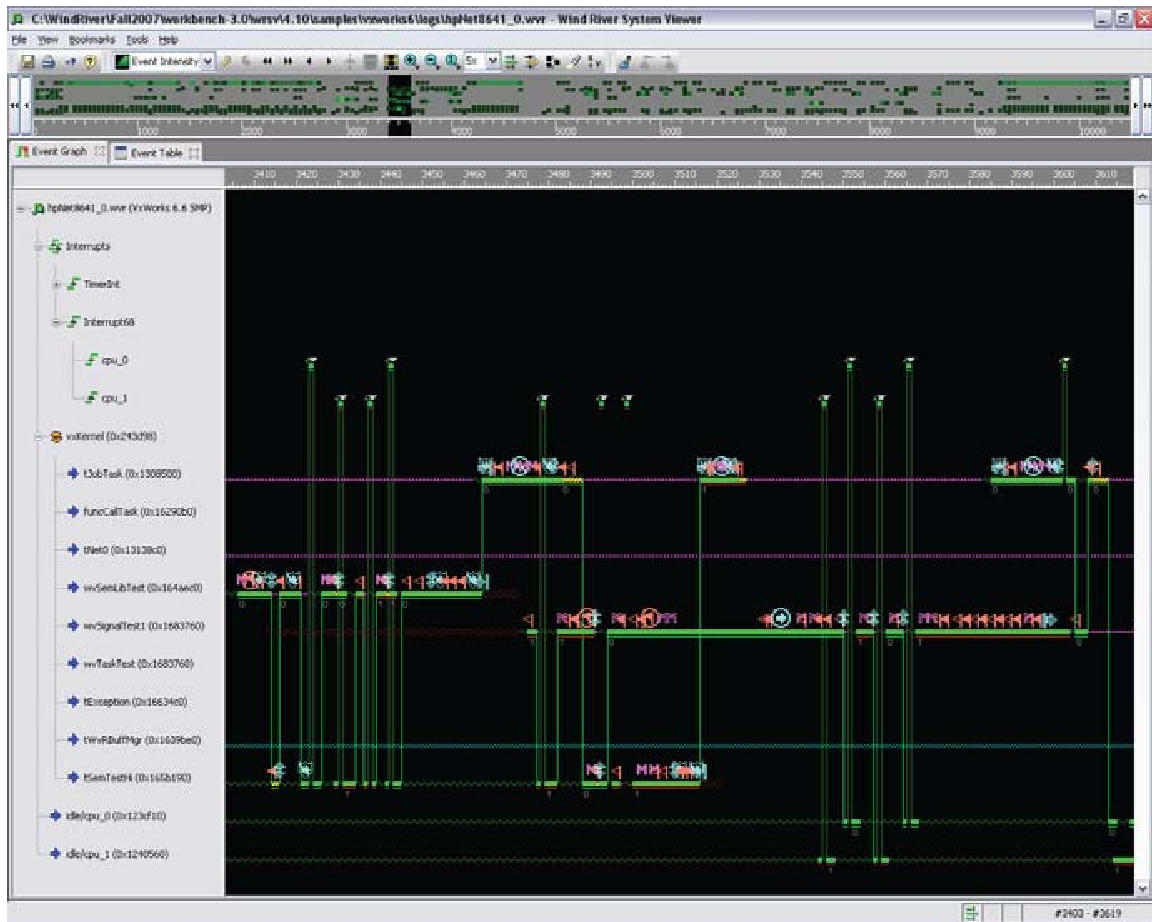

Figure 2.5: Wind River System Viewer with SMP support.

3 Functions of the task visualization

3.1 Task Schedule

Every task has a unique identifier, its name, chosen in the Control Builder. For every task there is also a time line of states assumed; active, blocked or idle. In the active state a task has been granted time by the scheduler and is actively executing. If a higher priority task wants to execute during some other tasks active state, the active task will eventually become blocked and the higher priority task will become the active one. When the higher priority task has finished its work, it becomes idle, and the blocked task once again becomes active, resuming its work until it also goes into idle. This is an example of what a task schedule can constitute.

It is important to note that the dynamics forming the task schedule is founded in the choice of a priority-based scheduling. Better timely guarantees for highly prioritized tasks are favored at the possible expense of those with lower. Further complexity to the dynamics is added by the usage of common resources.

3.2 Graphical presentation format

In Figure 3.1 a simplified but representative view of the graphical format chosen to represent the task schedule can be seen. The different features are colored but this might not be visible in all formats of this paper, keep this in mind when reading the explanation for each one:

- Timeline: At the bottom of the figure is a horizontal line representing time passed in milliseconds from the start of the data collection.

- Task-lines: For each task there is a corresponding line on which the state of the task is shown. The time on the task-lines are directly connected to the time on the timeline. When following a time-line the different states of that specific task can be seen as:

  - Green bars(darker): Representing the time periods when this task is actively executing.

  - Pink bars(lighter): Representing the time periods when this task has started but is not executing due to blockage from other tasks.

  - Initial grey flat line: Representing that the state of this task is unknown.

  - Black flat line: Represents an idle state of a task.

- Task name: To the left is the same name as was chosen within the Control Builder for this task. The tasks are sorted in order of priority.

- Custom event: The vertical dashed line with the number 7 below represents that a custom event with EventId := 7 occurred here.

- Time grid: The vertical lines that span over the whole height of the task schedule are part of a grid, consisting of these lines every 100 pixels. There is one vertical line for each number printed out below the time line.

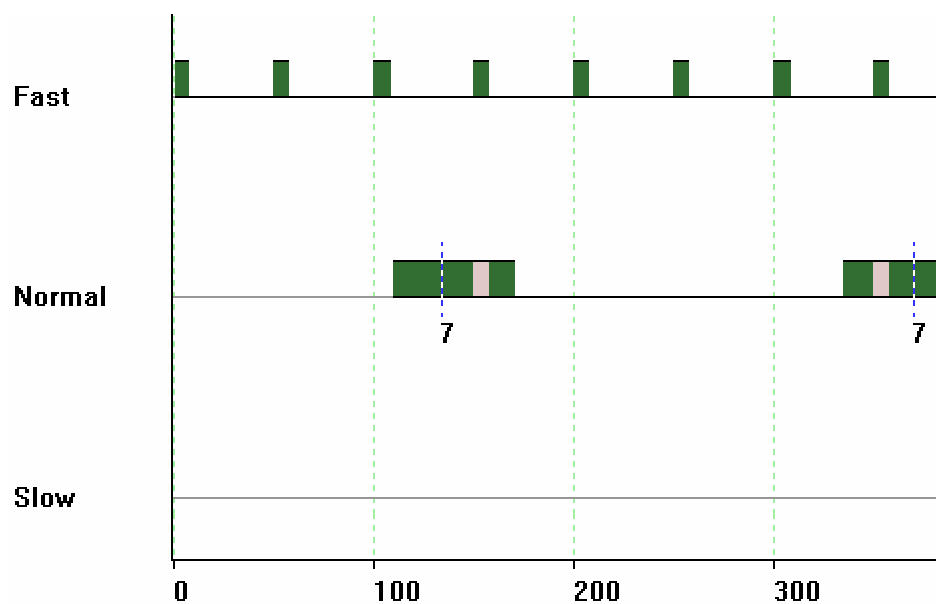

Figure 3.1: Example of a task schedule and the graphical representation.

In the above figure three tasks can be observed. The tasks are sorted by priority with a task of higher priority being above one of lesser.

This particular instance shows a task schedule where the task referred to as Fast gets to execute its routine many times and even interrupt the lower priority task named Normal at around 150 ms and 350 ms. It can be easily noted that Fast has a period of 50 ms. The task named Slow obviously has a long period and does not execute within this almost 400 ms long task schedule.

The number 7 visible under the task-line of Normal signifies the occurrence of a custom event, to be explained in detail later.

3.3 Usage scenarios

Something should be said about possible usage of the developed functionality. The value is rooted in its origins, springing from wishes of the tool being developed. Possible uses and ideas for it will be listed but certainly this will just be a list of suggestions. It is most useful in engineering endeavors. For example:

- Developmental phases like
  - Testing
  - Debugging
- Surveillance of online operation

A combination of markings using custom events can be used to see the chronology of events when debugging, which could have uses when events do not seem to happen in the order that they should. Another possibility is to get an overview of which tasks are present and why and how they get interrupted. To say the possibilities are endless would be an overstatement, it is rather of such a nature that the possibilities are not clearly defined and it could more generally be classified as an engineering tool to be used as fit, without having a specific purpose in and of itself. Thoroughly understanding the governing dynamics of the task schedule is of essence in analysis.

3.4 Data collection

The first main step for the task visualization is collecting the data needed for drawing. It is initiated by the user through the Control Builder which serves the request as appropriate to the Controller. The data is collected by the controller and further communicated to the Control Builder and visualized to the user. The data collected can be of two different types, collectively referred to as data points. The sequence of data points is ordered in time with time stamps and additional information depending on the type of data point.

Types of data points

There are as mentioned only two different types of data points, signified by the event that they capture.

Task event

The task event occurs when a task starts or stops. It is thus essential for drawing a task schedule. In the data collection the following information is saved:

- Time
- Task Identifier
- Event type (start or stop)

Custom event

This type of event is collected when a user-placed event-trigging function is placed within the 1131-code. An example looks as follows:

```
Mark( EventId := 7 );
```

This is a function made available to 1131-tasks within the modified code and interface of the firmware. With the above code a data point will be collected at the time instant that this function is called. The following information is saved:

- Time
- Task Identifier
- Custom event identifier (`EventId` above)

Due to implementation choice the `EventId` has a minimum integer value that it must assume to be effective. During data collection the same field is used for storing either the information representing a task event or a custom event. This means that the lowest numbers will represents starts and stops and higher numbers will represent custom events with that same number as `EventId`. The `EventId` thus has to be chosen higher than or equal to 3. Choosing a lower number will not issue a false task event, it will simply be ignored.

3.5 Starting a data collection

The data collection is started by a user menu choice. Before this menu choice can be chosen an appropriate project defining the connection to a controller must have been configured within the Control Builder. To facilitate reaching the interesting data to be collected, there are three different ways of starting a data collection covered next. The data collection in itself does not vary between the methods.

Immediate

Starting a data collection immediately makes the controller start the collection as soon as the message reaches it. There are no special guarantees on when the data collection will start and basically this will give a snapshot of the current activity.

At task

Another possibility is to start the data collection when the start of a specific task is encountered. This triggers the data collection immediately and hence the triggering task start is also included within the set of data points.

At custom event

Similarly as the previous case, this immediately triggers the collection of data points and hence this specific occurrence of the custom event will be the first one in the collection.

3.6 Auxiliary functions of the graphical view

Zoom

Coming back to the time grid mentioned in section 3.2, a feature has been implemented to allow selection of the actual time that the distance between two of these grid lines represents. The selection is made by choosing a time-based grid length from the menu. As stated earlier the distance between two grid lines is always fixed at 100 pixels, and if a choice of 100 milliseconds is made for this distance, it would equate to 1 millisecond per pixel. This can be seen as a horizontal zooming, i.e. a zoom along the time axis, see Figure 3.2.

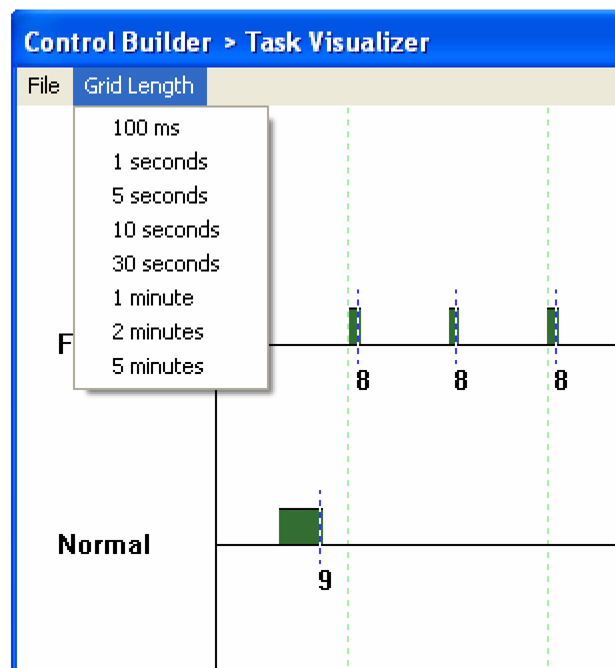

Figure 3.2: Choosing the time between two vertical grid lines.

Scrolling

Another feature is that of vertical scrolling. In the event of many executing tasks they might not all fit on the screen and hence this is needed.

Export/import

To further elaborate on the usefulness and tool-like aim of it all, a feature for exporting and importing collected data has been implemented, available through the file-menu of the graphical view. The data is exported to a file in a tabular format allowing easy access to it from other tools. It is also of course directly importable back into this tool, for future viewing.

Some possible uses for it are:

- Saving examples of problems encountered
- Exporting data to external tools for further processing, e.g. MATLAB or Excel
- Mailing a task schedule of interest to another colleague

The data format is best described by an example:

```
ID     PRIO   NAME
1      1      Fast
2      3      Normal

TIME   ID     EVENT
10     1      1      -
15     1      2      -
18     2      1      -
20     1      1      -
25     1      2      -
26     2      3      5
28     2      2      -
30     1      1      -
```

As can be seen, first there is a header describing all the tasks and connecting ids to names and priorities. It can also be seen in the above example that two tasks are executing and that the task Fast has a higher priority.

After the header follows a chronological list of events with time, id and event type for each. The column EVENT represents the event type with:

1 = Start
2 = End
3 = Custom event

The last column is only valid for custom events (EVENT=3), for which it specifies the custom event id.

4 Implementation

The code base for the 800xA system upon which this project builds is known as ATLAS. It is a large and structured code base and the modifications added by this project will only touch a few of the subparts relating to task execution, network communication and some simple user interface modifications. It consists mostly of C++ code and that is the language used also by the new additions to the project needed to support the developed task visualization functionality.

A subdivision of the modules within ATLAS that have been modified, even if to a small extent, is presented:

- **ApplicationLayer**
    - **AppExecution:** Firmware function and collection of custom events.
    - **ControlComm:** Collection of events relating to task starts and stops.
    - **ProjectBuilder:** Small helper-functions.
- **BasicServiceLayer**
    - **Execution**
        - **Conevent:** Background job functionality.
- **UILayer**
    - **ProjectUI**
        - **SysMenus:** User interface additions.

The main components presented in the following, TSCS and TSCC, have both been put as modules under ControlComm above.

4.1 Main components and modifications

The figure below illustrates the general relationship between the three main components of task visualization which will all be covered in the following text.

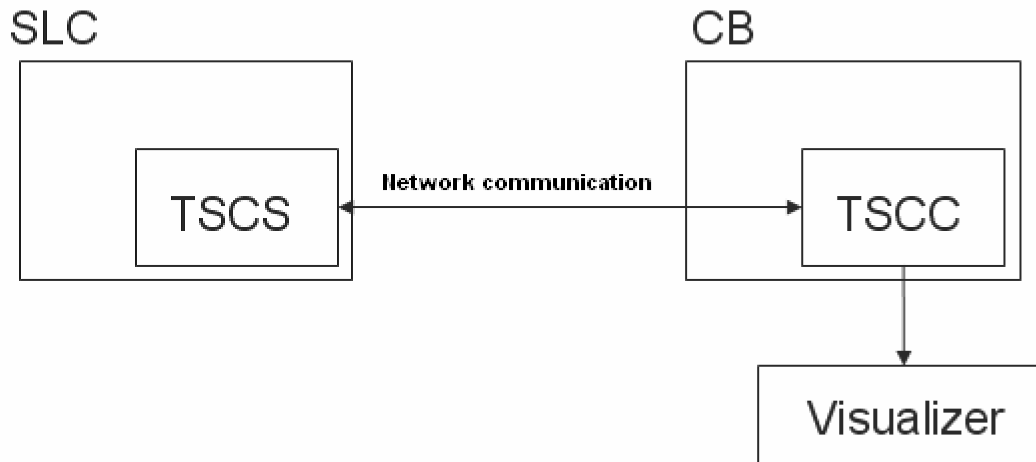

Figure 4.1: Main components of the task visualization functionality.

The left side of Figure 4.1 resides within the modified firmware of the controller and the right side within the code of the Control Builder. Network communication used by the components is the MMS layer mentioned in the background section. The connection however is set up using knowledge of IP-addresses.

The Visualizer component is technically a COM-object residing separately outside the code of the CB and accessible through COM-routines, but this is only a developmental choice and issue, for all other purposes it can be thought of as a separate module residing in the CB as is the TSCC.

The main components are the only new modules added to the code base but it interrelates with many other parts of it. Some of the parts that it interrelates with have also been modified, for example for data collection there have been modifications made within the parts relating to starting and stopping the execution of a task.

Many modifications have also been made altering the subsystem of MMS communication. This has consisted of many small changes throughout the code base, guided by thesis supervisors and examples of how they were modified for other functionalities. Explanations of why these changes were necessary will not be a part of this thesis, as the components underlying were not analyzed fully or even founded in the work by the author, not saying that it was not a lot of work to analyze them enough to actually make it coherent and functional.

Much the same can be said for many parts of the implementation where analyzing what was already implemented constituted more work than actually implementing what was strived for. This of course comes from the size and complexity that follows with a code base of this size. Looking from the perspective of the main components (Figure 4.1) the relations will be brightly seen, without going into petty detail.

4.1.1 TaskScheduleCommServer (TSCS)

The main software component within the process module supporting the implementation of the task visualization is called the TaskScheduleCommServer (TSCS). Its duties are comprised of collecting data requested by the control builder and sending that data back to the corresponding component within the Control Builder (TSCC).

The general functionality can be most easily visualized and discussed through a state diagram, as below.

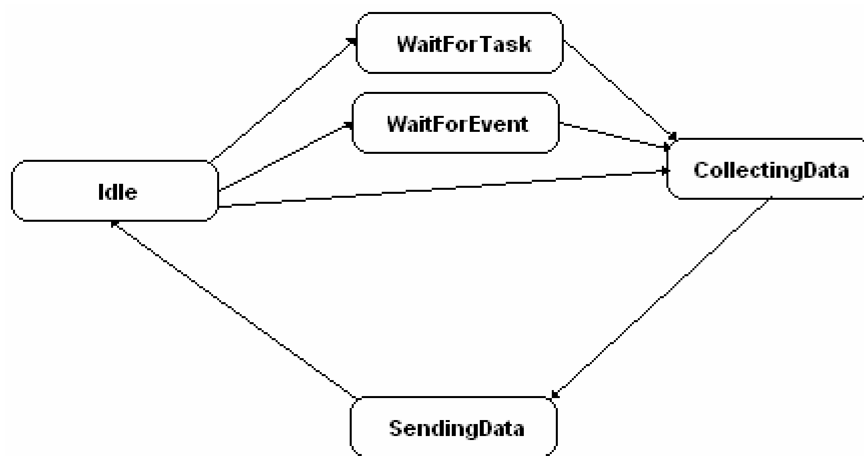

Figure 4.2: State diagram for TSCS.

One by one the states and the transitions between them will be described:

- **Idle** This is the initial state and it is left by an external network message coming from the TSCC of the Control Builder. Depending on this message it will pass to one of the wait states or go directly into collecting data.
- **Wait for task** This corresponds directly to the waiting for a task to start, as explained in the section describing task visualization functionality. This state will be left and transitioned into the collecting data state when the task specified in the network message starts.

- **Wait for custom event** Analogous to the former, this state waits but for a custom event and transitions when it is encountered.

- **Collecting data** This state corresponds to active collection of the events as a sequence of data points. It is left when the fixed number of data points, as mentioned in the functional description, have been collected.

- **Sending data** This state entails sending of the collected data back to the Control Builder, where it is further shown graphically. After the last packet of data has been sent, the TSCS once again enters the Idle state.

The event that triggered the transition from a wait state, is also included in the data collection.

On a language basis TSCS is a C++ class with member properties and functions. Its relations to other parts of the code can be simplified with the following easily overviewed figure which introduces the most central member functions.

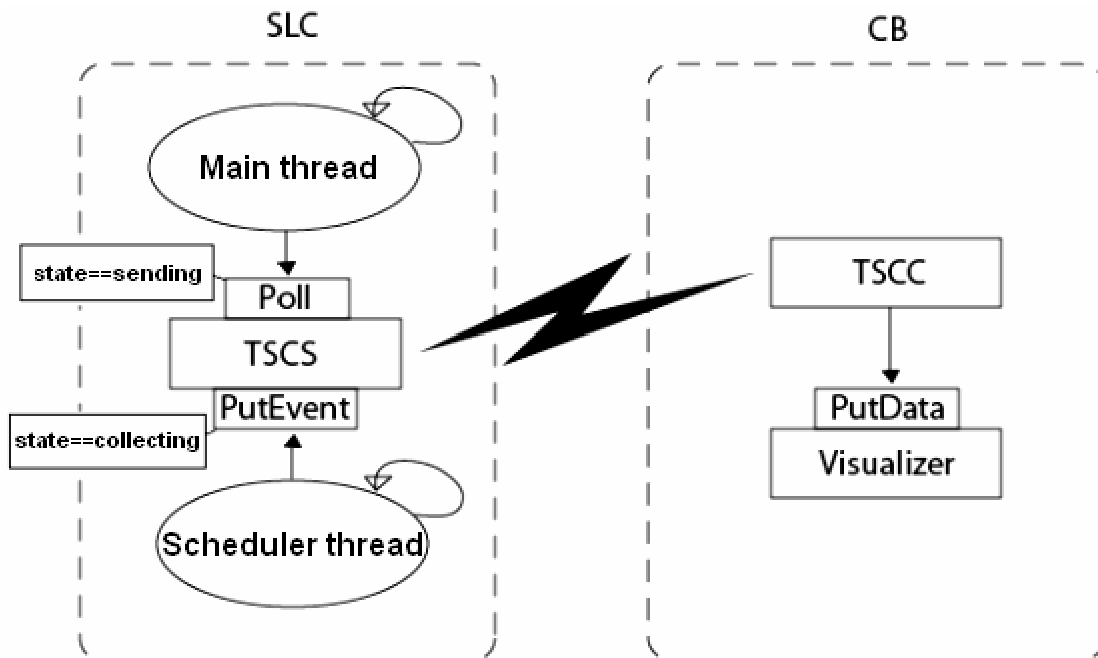

Figure 4.3: Implementation details of central functionality.

The notion of two threads accessing the exposed interface of TSCS can be seen. During collection the `PutEvent(..)` function is called from the scheduler-thread. This is implemented as a very light-weight function, only saving the information needed for the data point in a previously allocated array, i.e. without any dynamic memory allocation. When the `PutEvent`-function is called, it either puts a task start/stop event or a custom event in the array.

The `Poll()` function on the other hand is called when TSCS is in the SendingData state as explained previously. This function is called from another thread within the controller, named the main thread. Within this thread different functionalities are handled, e.g. network communication, as is relevant in this case. The main-thread gets time to do its

work when the scheduler-thread is out of work, i.e. when no 1131-tasks are active. One of the functionalities that reside within the processing time of the main-thread are what are known as background jobs. This is exactly the way that the networking of collected data to the TSCC is implemented within the TSCS. The main thread calls the static `Poll()` member function of TSCS in the same manner as it calls `Poll()`-equivalents of other background jobs, so there is another type of time-sharing here.

Within the background job of the TSCS there is a need for a separate state machine keeping track of the network sending progress, because of the fact that the time-sharing of background jobs in the main-thread is based on voluntary yielding of processing time. The states do one part at a time of assembling the packets containing header data and collected data points in the following manner:

- Initiating the header.
- Filling one item in the header
- Filling one sequence of data points up to the maximum size of the packet

The details are best left to be viewed in the actual source code.

On a further note it can be stated that the TSCS is implemented as a singleton and hence there is only one ever present within the object space. During initialization, i.e. the first time this class is accessed and the constructor is called, the allocation of the data buffer is made, as to avoid dynamic allocation during active data collection. Owing to the fact that a network packet will always be received from within the main-thread before any data collection, and that this packet is further directed to the TSCS from the same context, it will always have been created when the scheduler thread accesses it.

4.1.2 TaskScheduleCommClient (TSCC)

The Control Builder counterpart of the TSCS is the TSCC. The naming of these modules corresponds to their general roles as requester/responder. The TSCC has an intermediate role responding to user commands with network messages to and from the TSCS and also to and from the Visualizer component, covered later.

Understanding the TSCS one should have a fairly good picture of how the state diagram of TSCC looks like. As was the fact for the controller component, the TSCC is also a singleton C++ class.

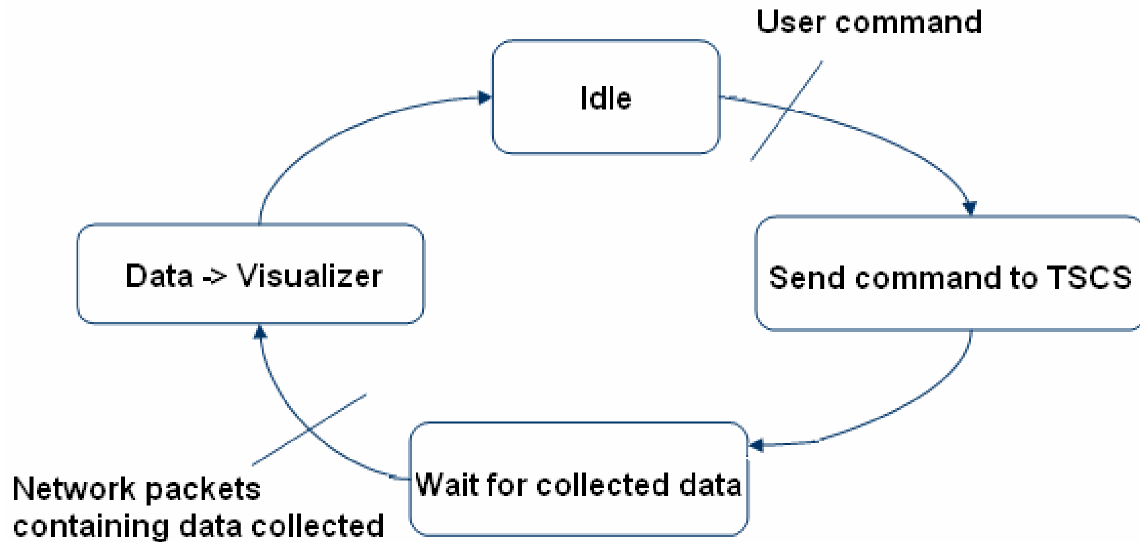

Figure 4.4: General outline of TSCC states.

It can be seen that both transitions depend on external events, in one case a user command and in the other a network response.

When the data collection has been retrieved, it is translated into a format suitable for the interface of the Visualizer. This entails concatenating the information in the packets and other small transformations.

4.1.3 Visualizer

The visualizer is concerned with transforming the time series of events, i.e. the data collection, into a graphical task schedule as explained in the section on the graphical presentation format. It does this mainly using two different algorithms explained in the subsections.

This component is self-contained and exposes just a simple COM-interface to the TSCS. The only relevant part of this interface, disregarding COM-specific necessities, is a function called `PutData(..)` which could be seen in Figure 4.3. This function can be called many times with new data and the graphical view will update accordingly.

The data sent to `PutData` contains both header-data, containing task names, task identifiers and task priorities, as well as a time series of the events. This time series will be shifted relative one another by the visualizer so that the time stamps start at 0.

Scheduler simulation algorithm

There can be noted a gap of information between the discrete time series of data points and the continuous states of a task, i.e. what happens in between data points. This problem is further amended by the fact that only starts and stops of tasks are collected as data points, hence the blocking and activation of the different tasks within a specific task set, will have to be inferred using the scheduler simulation algorithm. The states that any one task can assume within this structure coincide with those that can be represented in the graphical presentation format:

- Unknown
- Active
- Blocked
- Idle

The scheduler simulation algorithm is linear and successively updates the task states as the data points are iterated through. Initially the state of each task is assumed to be simply unknown and thus graphically represented as such. As more and more starts are encountered within the time series, more and more tasks go into the state of active/blocked/idle etc., and when a task is started, the current task with state marked as active will go into the blocked state, and only the just started one will have its state set as active. As task stops are encountered, the corresponding task is noted as being in idle state. The previously active task once again is assumed to be active. The list of previously active tasks is kept internally as a stack of tasks, pushing and popping them as they start and stop. This can be implemented as such due to the fixed priority-based scheduling used in the controller.

Task visualization algorithm

The task visualization algorithm is tightly knit to the scheduler simulation algorithm. The latter runs inside the former in parallel and stepwise while each data point in the data collection is iterated over. Using the following figure the full functionality will be easily explained.

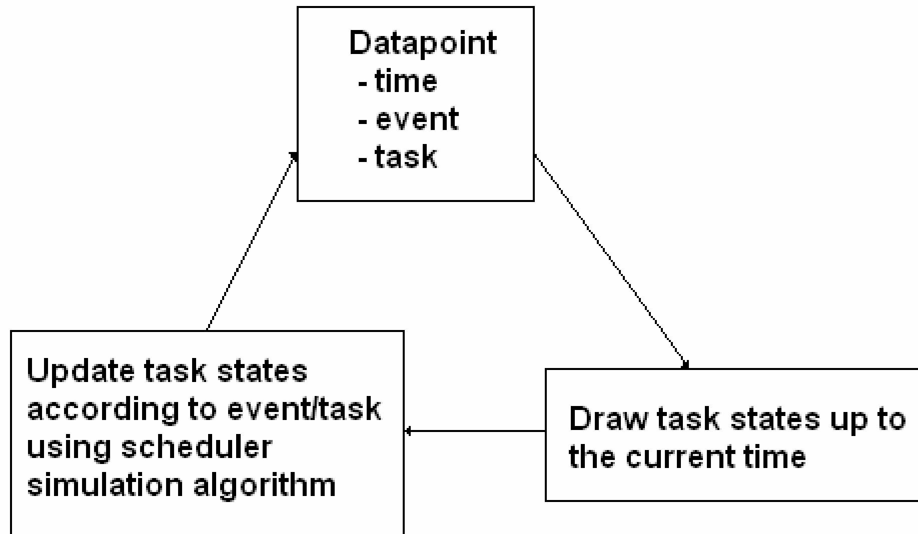

Figure 4.5: Transforming data points into graphics.

As stated earlier the scheduler simulator assumes each state to be unknown initially. Hence when the first data point is encountered it will in the case of a task start, change the state of that task into active. Since the first time stamp is at time zero, nothing can be drawn, since the interval over which information is known is basically of zero length. The next time a data point is encountered however, the state has not changed *before* the time of this new data point, and hence up to this time point, all the tasks states can be drawn to the extent that they are known.

4.2 Protocols and data packets

The protocols behind these data packets are trivially understood consulting the sections of how TSCS and TSCC work, where they have been mentioned. For completion their build up will be shown and discussed briefly.

4.2.1 TSCC to TSCS

Three very similar types of packets can be sent from the TSCC to the TSCS, messaging that the data collection should be started and how. The when-field is used to distinguish if the extra field is present and how it should be interpreted. In the case of starting the data collection at the start of a specific task, the name of this task is communicated. In the case of starting at a specific custom event, the numerical id of this custom event is communicated.

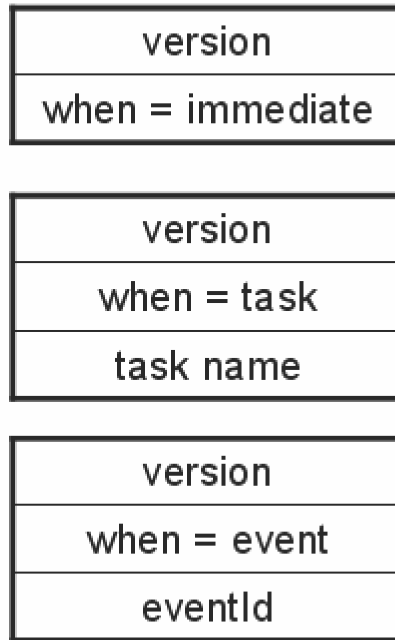

Figure 4.6: Data packets concerning starting of data collection.

4.2.2 TSCS to TSCC

In Figure 4.7 the format of the packets traveling back to the TSCC due to a request can be seen. There are two different types of packets because of the header data which is sent first. Unless the header data fills a whole packet or more by itself, there will be appended a part or the whole of the collected data points. If there is not enough room for all the collected data within this first packet, they will be sent in subsequent packets.

The stars to the left of the data packets signify multiplicity of specific parts. For example, in a packet there can be sent many concatenated parts looking like the part signified by HEADER in the figure, each representing the *tscsId*, *priority* and *task name* of a separate task.

The *tscsId* is a unique id created within the controller that binds each task to a specific integer, reason being compactness and not having to send the *task name* for every data point.

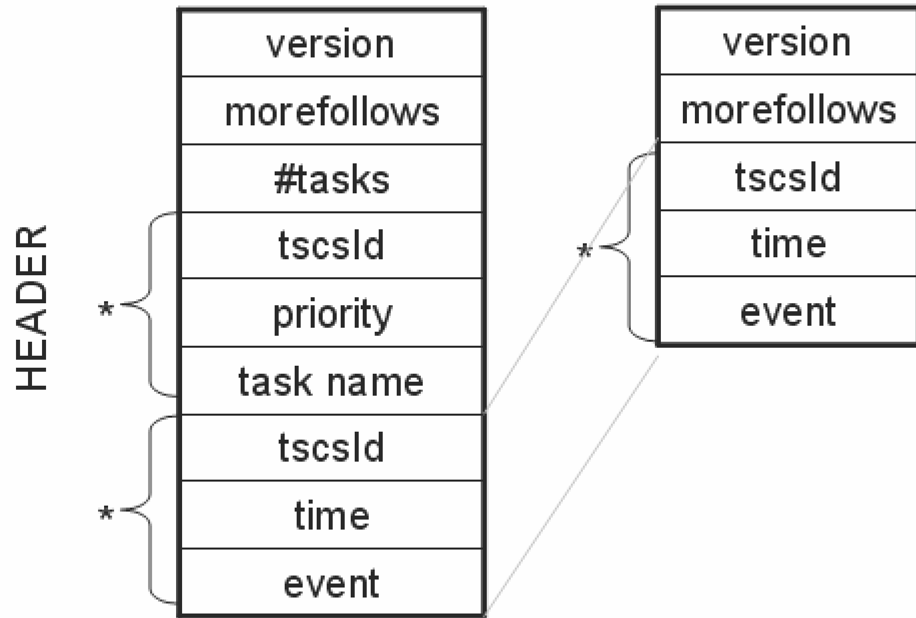

Figure 4.7: Data packets containing header and collection of data points.

5 Evaluation and future work

The core goal of graphically illustrating the execution of tasks has been implemented and tested to work. However, there are a few drawbacks or issues noted here which suggest possible future work. These points will be discussed below.

Blocked starts

Tasks that move into the ready queue of the scheduler get blocked from becoming active if there is currently a task of higher priority executing. This blockage will not be visible in the visualization since there is no data being collected for this specific type of event. Where and how this data point could be collected has not been investigated, but in the framework of TSCS, TSCC and Visualizer, support for this addition would be easily added.

Time gaps

In situations when the scheduler is making a task switch from task A to task B, and there is a simultaneously blocked task C of lower priority, the gap in time between the block/stop of A and the start of B will be interpreted by the current visualization algorithm as if task C executed within that small gap. Using the data points collected, the following two situations are indistinguishable.

a) a time gap allowing C to actually execute
b) a time gap caused by scheduler internal task switching time

The problem is tightly coupled with the needed use of a stack in the scheduler simulation algorithm. A solution to both this problem and the previous one would have to entail two more data points being collected i.e. blocked start of a task and activation of a blocked task. Collecting these would imply modifying the scheduler on a deeper level than what has been done here, since the changes in the scheduler’s ready queue would have to be monitored.

Low-level time addition

There are some low-level tasks with higher priority than the scheduler thread and their execution time would be seen as 1131-task execution time within the visualization. This is not necessarily an unwanted situation. It can also be noted that considering an even spread of this lower level execution time would equate to an overall slower processing speed. The effects have not been measured.

Data field sharing of custom events and task events

Relating back to Section 3.4 on data collection a usability problem can be noted regarding the demand for a minimum value of the `EventId`. The non-intuitive nature of

this part of the custom event functionality should for example be redesigned or, tool-assisted or clearly documented. A simple solution could be to internally shift the stored value relative the user entered value to normalize the latter to start from zero. This would have the effect of barring high numbers instead.

6 Appendix

6.1 Source code TSCS

TaskScheduleCommServer.h

```
/***********************************************************************
*******
*
* Filename:      TaskScheduleCommServer.h
*
* Author:        Martin Furmanski
* Responsible dept: ATPA/XA/ACN
* Creation date: 2008-01-31
*
* Description:
*
* Products:         slc
* SIL level:        nonSIL
*
* Definitions of classes:
*
************************************************************************
*******/

#ifndef _TaskScheduleCommServer_H
#define _TaskScheduleCommServer_H

#include <heap/heap_gtc.h>
#include <varuser/varuser_gtc.h>
#include <transac/transac_gtc.h>
#include <asn1types/asn1types_gtc.h>
#include <asn1/asn1_gtc.h>
#include <strings/stringtype_ftc.h>
#include <appexecution/IAppRT_ProgControl_gtc.h>
#include <values/values_gfv.h>
#include <hwinterface/hwinterface_gtc.h>
#include "conevent/PollController.h"
#include <progcontrol/programinstance_ftc.h>
#include <progcontrol/scangroup_ftc.h>

#include "C:\root\Martin\TEST2\TEST2\scan_structs.h"

class TaskScheduleCommServer : public HeapObj
{
public:
  int Connect(Element eAssoc);
  int WriteInd(pAssoc eAssoc, int Pos, int LastPos, UnsignedByte*
UnconstrainedAddr);

  static TaskScheduleCommServer* Instance();// Create single instance.
```

```
  void PutTaskEvent(int tscsId, int time, unsigned char eTaskEvent); //
Put data point in pre-allocated array
  void PackValue(rCodedListOfData* pResMsg, tValue Value);
  void PackString(rCodedListOfData* pMsg, const char* Str);
  int ValueSize(tValue Value) const;
  static void Poll();
  void Poll2(); // Polled network sending of data collection
  bool CollectActive();
  void SetupTaskIdentifiers();
  int GetTSCSId(char* task);
  bool FindAndResetAssocDesc(pAssocDesc eAssocDesc);

  enum state_t { eIdle, eWaitForTask, eWaitForEvent, eCollectingData,
eSendingData };
  enum sendState_t { eInitHeader, eFillHeader, eSendTaskData, eSendIdle
};

  TaskScheduleCommServer();
  ~TaskScheduleCommServer();
private:

  static TaskScheduleCommServer* m_theTSCS;
  pAssocDesc m_pAssocDesc;
  bool m_Connected;

  state_t m_State;
  int m_AwaitedTask;
  int m_AwaitedEvent;
  sendState_t m_sendState;
  int m_sendStatePos;
  int m_SendBufStartPos;
  unsigned int nTasks;
  static const unsigned char cTSVersion = 1;
  static const int cMoreFollowsPos = 1;

  const int m_cEventsToCollect;
  rCodedListOfData* m_sendBuf;

  int m_nEventsCollected;
  unsigned char* m_pEventsBuf;
  WirthsDevice32 Help32;
  PollController m_pollController;

  rProgramInstance* eProgramInstance;
  rScanGroupSystem* puTaskSystem;
  rScanGroupData* puTaskData;

  char chbuf[256];
};


#endif // _TaskScheduleCommServer_H
```

TaskScheduleCommServer.cpp

```
#include "TaskScheduleCommServer.h"

TaskScheduleCommServer* TaskScheduleCommServer::m_theTSCS = NULL;

TaskScheduleCommServer::TaskScheduleCommServer() :
  m_Connected(false), m_State(eIdle), m_cEventsToCollect(150),
m_sendBuf(0), m_pollController(2, 7)
{
  m_pAssocDesc = 0;
  m_pEventsBuf = new unsigned char[m_cEventsToCollect*9]; // 4byte
tscsId, 4 byte time, 1 byte event = 9 bytes
}

TaskScheduleCommServer::~TaskScheduleCommServer()
{
}

/* Creates the single instance of TSCS. */
TaskScheduleCommServer* TaskScheduleCommServer::Instance()
{
  if(m_theTSCS == NULL)
    m_theTSCS = new TaskScheduleCommServer();

  return m_theTSCS;
}

void TaskScheduleCommServer::Poll() {
  TaskScheduleCommServer::Instance()->Poll2();
}

/* Used for sending collected data, polled by main thread's background
job functionality. */
void TaskScheduleCommServer::Poll2() {
  if(!m_pollController.enter()) {
    return;
  }

  if(m_State != eSendingData)
    return;

  while(!m_pollController.leave()) {
    switch(m_sendState) {
      case eInitHeader:
        printf("TSCS sendState->eInitHeader\n");
        eProgramInstance =
(rProgramInstance*)First(GetProgramInvocationList());
        puTaskSystem = (rScanGroupSystem*)First(eProgramInstance-
>SystemList);
        puTaskData = (rScanGroupData*)First(puTaskSystem-
>ScanGroupList);

        CreateCodedListOfData((pCodedListOfData*) &m_sendBuf,
NormalCodedListOfData);
        tIdentifier bogus; bogus.SetContents("");
```

```
        PackPICHeader(bogus, TSKind, m_sendBuf->OctetString,
&m_sendBuf->pos);
        m_SendBufStartPos = m_sendBuf->pos;
        ((rOctetString*) m_sendBuf->OctetString)->buf[m_sendBuf->pos++]
= cTSVersion; // Pack version number for TaskSchedule data
        ((rOctetString*) m_sendBuf->OctetString)->buf[m_sendBuf->pos++]
= true; // Morefollows after this packet? Default: True (can be changed
later)

        tValue val;
        val.ValType = DIntType;
        val.UU.DIntVal = nTasks;
        PackValue(m_sendBuf, val); // Pack number of tasks

        m_sendStatePos = 0;
        m_sendState = eFillHeader;
        break;
      case eFillHeader:
        printf("TSCS sendState->eFillHeader\n");
        if(!(eProgramInstance == NULL)) {
          if(!(puTaskSystem == NULL)) {
            if(!(puTaskData == NULL)) {

              tValue tscsId; tscsId.ValType = DIntType;
              tscsId.UU.DIntVal = puTaskData->tscsId;

              tValue prio; prio.ValType = DIntType;
              prio.UU.DIntVal = puTaskData->Priority;


              int strsize = ((rString*)puTaskData->Name_)->CurrLength
                            + 1 // valtype
                            + 1; // len
              int j;
              for(j=0;j<((rString*)puTaskData->Name_)->CurrLength;j++)
{
                chbuf[j] = ((char*)&((rString*)puTaskData->Name_)-
>UU.FirstByte)[j];
              }
              chbuf[j] = 0;

              if(m_sendBuf->pos + 1 + ValueSize(tscsId) +
ValueSize(prio) + strsize > MaxWriteRequestDataSize) {
                PutTSReqAction((rAssocDescription*) m_pAssocDesc,
m_sendBuf);
                CreateCodedListOfData((pCodedListOfData*) &m_sendBuf,
NormalCodedListOfData);
                tIdentifier bogus; bogus.SetContents("");
                PackPICHeader(bogus, TSKind, m_sendBuf->OctetString,
&m_sendBuf->pos);
                m_SendBufStartPos = m_sendBuf->pos;
                ((rOctetString*) m_sendBuf->OctetString)-
>buf[m_sendBuf->pos++] = cTSVersion; // Pack version number for
TaskSchedule data
                ((rOctetString*) m_sendBuf->OctetString)-
>buf[m_sendBuf->pos++] = true; // Morefollows after this packet?
Default: True (can be changed later)
```

```
              }

              PackValue(m_sendBuf, tscsId);
              PackValue(m_sendBuf, prio);
              ((rOctetString*) m_sendBuf->OctetString)->buf[m_sendBuf-
>pos++] = StringType;
              PackString(m_sendBuf, chbuf);

              puTaskData =
(rScanGroupData*)SuccElem((Element)puTaskData);
            } else {
              puTaskSystem =
(rScanGroupSystem*)SuccElem((Element)puTaskSystem); // next sys
              if(puTaskSystem != NULL)
                puTaskData = (rScanGroupData*)First(puTaskSystem-
>ScanGroupList);
            }
          } else {
            eProgramInstance =
(rProgramInstance*)SuccElem((Element)eProgramInstance); // next prog
            if(eProgramInstance != NULL) {
              puTaskSystem = (rScanGroupSystem*)First(eProgramInstance-
>SystemList);
              if(puTaskSystem != NULL)
                puTaskData = (rScanGroupData*)First(puTaskSystem-
>ScanGroupList); }
          }

        } else {
          m_sendStatePos = 0;
          m_sendState = eSendTaskData;
        }
        break;
      case eSendTaskData:
        printf("TSCS sendState->eSendTaskData\n");
        if(m_sendBuf->pos + 1 + (m_cEventsToCollect*9-m_sendStatePos) >
MaxWriteRequestDataSize) {
          while(m_sendBuf->pos < MaxWriteRequestDataSize) {
            ((rOctetString*) m_sendBuf->OctetString)->buf[m_sendBuf-
>pos++] = m_pEventsBuf[m_sendStatePos++];
          }

          PutTSReqAction((rAssocDescription*) m_pAssocDesc, m_sendBuf);

          CreateCodedListOfData((pCodedListOfData*) &m_sendBuf,
NormalCodedListOfData);
          tIdentifier bogus; bogus.SetContents("");
          PackPICHeader(bogus, TSKind, m_sendBuf->OctetString,
&m_sendBuf->pos);
          m_SendBufStartPos = m_sendBuf->pos;
          ((rOctetString*) m_sendBuf->OctetString)->buf[m_sendBuf-
>pos++] = cTSVersion; // Pack version number for TaskSchedule data
          ((rOctetString*) m_sendBuf->OctetString)->buf[m_sendBuf-
>pos++] = true; // Morefollows after this packet? Default: True (might
be changed later by code)
        } else {
```

```
          ((rOctetString*) m_sendBuf->OctetString)-
>buf[m_SendBufStartPos+cMoreFollowsPos] = false;

          while(m_sendStatePos < m_cEventsToCollect*9)
            ((rOctetString*) m_sendBuf->OctetString)->buf[m_sendBuf-
>pos++] = m_pEventsBuf[m_sendStatePos++];

          PutTSReqAction((rAssocDescription*) m_pAssocDesc, m_sendBuf);
          m_sendBuf = 0;

          m_sendState = eSendIdle;
        }
        break;
      case eSendIdle:
        printf("TSCS sendState->eSendIdle\n");
        m_pollController.jobDone();
        m_State = eIdle;
        return;
        break;

    }
  }
}

/*
We create a special identification number that is unique for every
Task.
*/
void TaskScheduleCommServer::SetupTaskIdentifiers()
{
  nTasks = 0;

  rProgramInstance* eProgramInstance =
(rProgramInstance*)First(GetProgramInvocationList());
  while (eProgramInstance != NULL)
  {
    rScanGroupSystem* puTaskSystem =
(rScanGroupSystem*)First(eProgramInstance->SystemList);
    while (puTaskSystem != NULL)
    {
      rScanGroupData* puTaskData = (rScanGroupData*)First(puTaskSystem-
>ScanGroupList);
      while (puTaskData != NULL)
      {
        puTaskData->tscsId = nTasks;
        nTasks++;

        puTaskData = (rScanGroupData*)SuccElem((Element)puTaskData);
      }
      puTaskSystem =
(rScanGroupSystem*)SuccElem((Element)puTaskSystem);
    }
    eProgramInstance =
(rProgramInstance*)SuccElem((Element)eProgramInstance);
  }
}
```

```
int TaskScheduleCommServer::GetTSCSId(char* task)
{
  rProgramInstance* eProgramInstance =
(rProgramInstance*)First(GetProgramInvocationList());
  while (eProgramInstance != NULL)
  {
    rScanGroupSystem* puTaskSystem =
(rScanGroupSystem*)First(eProgramInstance->SystemList);
    while (puTaskSystem != NULL)
    {
      rScanGroupData* puTaskData = (rScanGroupData*)First(puTaskSystem-
>ScanGroupList);
      while (puTaskData != NULL)
      {
        unsigned int len = ((rString*)puTaskData->Name_)->CurrLength;
        if(len == strlen(task)) {
          bool match = true;

          for(unsigned int i=0;i<len && match;i++)
            if(!      (((char*)&((rString*)puTaskData->Name_)-
>UU.FirstByte)[i]==task[i])  )
              match = false;

          if(match)
            return puTaskData->tscsId;
        }

        puTaskData = (rScanGroupData*)SuccElem((Element)puTaskData);
      }
      puTaskSystem =
(rScanGroupSystem*)SuccElem((Element)puTaskSystem);
    }
    eProgramInstance =
(rProgramInstance*)SuccElem((Element)eProgramInstance);
  }

  return -1;
}

bool TaskScheduleCommServer::FindAndResetAssocDesc(pAssocDesc
eAssocDesc)
{
  bool Equal = false;

  if (m_pAssocDesc == eAssocDesc)
  {
    m_pAssocDesc = NULL;
    m_Connected = false;
    Equal = true;
  }

  return Equal;
}

int TaskScheduleCommServer::WriteInd(pAssoc eAssoc, int Pos, int
LastPos, UnsignedByte* ReqMsg)
{
```

```
  int Status = cSuccess;

  if(!(m_State == eIdle))
    return cErrOperationNotAllowed;

  if (m_pAssocDesc == NULL) {
      Status = Connect(eAssoc);
  }
  else if (GetMMSUserRef(eAssoc) != m_pAssocDesc)
    Status = cErrOperationNotAllowed; // Only one active client is
allowed, return error

  if (Status == cSuccess) {
    Pos+=1; // Skip null progname.

    int when = ReqMsg[Pos++];

    SetupTaskIdentifiers();

    switch(when) {
      case 1:
        m_nEventsCollected = 0;
        m_State = eCollectingData;
        break;
      case 2:
        {
          int len = ReqMsg[Pos++];
          char *task = new char[len+1];

          for(int i=0;i<len;i++)
            task[i] = ReqMsg[Pos++];
          task[len] = 0;

          m_AwaitedTask = GetTSCSId(task);

          if(m_AwaitedTask>=0) {
            m_nEventsCollected = 0;
            m_State = eWaitForTask;
          }
        }
        break;
      case 3:
        m_AwaitedEvent = ReqMsg[Pos++];
        m_nEventsCollected = 0;
        m_State = eWaitForEvent;
        break;
    }
  } else {
    printf("Error connecting for task data.\n");
  }

  return Status;
}

int TaskScheduleCommServer::Connect(Element eAssoc)
{
  bool Ok;
```

```
  int Status = cSuccess;

  if (m_pAssocDesc != NULL)
  {
    // Abort assoc to client with no active subscription.
    AbortReq(m_pAssocDesc->dAssocDescription.eAssoc);
  }

  m_pAssocDesc = GetMMSUserRef(eAssoc);
  if (m_pAssocDesc == NULL)
  {
    CreateServerAssocDesc(eAssoc, &m_pAssocDesc, &Ok);
    if (Ok)
    {
      SetMMSUserRef(eAssoc, m_pAssocDesc);
    }
    else
    {
      Status = cVarLocalHeapFull;
    }
  }

  if(Status == cSuccess)
    m_Connected = true;

  return Status;
}

bool TaskScheduleCommServer::CollectActive()
{
  return (m_State == eCollectingData || m_State == eWaitForTask ||
m_State == eWaitForEvent);
}

/* Called from the scheduler thread when a data point is collected.*/
void TaskScheduleCommServer::PutTaskEvent(int tscsId, int time,
unsigned char eTaskEvent)
{
  if (m_pAssocDesc == NULL || !m_Connected || !CollectActive())
  {
    return;
  }

  if(m_State == eWaitForTask) {
    if(tscsId == m_AwaitedTask && eTaskEvent == SCAN_START_NORMAL)
      m_State = eCollectingData;
    else
      return;
  } else if(m_State == eWaitForEvent) {
    if(eTaskEvent == m_AwaitedEvent)
      m_State = eCollectingData;
    else
      return;
  }

  unsigned int offset = m_nEventsCollected++*9;
```

```
  Help32.L = tscsId;
  m_pEventsBuf[offset++] = Help32.U1.lsb;
  m_pEventsBuf[offset++] = Help32.U1.nlsb;
  m_pEventsBuf[offset++] = Help32.U1.nmsb;
  m_pEventsBuf[offset++] = Help32.U1.msb;

  Help32.L = time;
  m_pEventsBuf[offset++] = Help32.U1.lsb;
  m_pEventsBuf[offset++] = Help32.U1.nlsb;
  m_pEventsBuf[offset++] = Help32.U1.nmsb;
  m_pEventsBuf[offset++] = Help32.U1.msb;

  m_pEventsBuf[offset] = eTaskEvent;


  if(m_nEventsCollected == m_cEventsToCollect) {
    m_State = eSendingData;
    m_sendState = eInitHeader;
  }
}

void TaskScheduleCommServer::PackValue(rCodedListOfData* pResMsg,
tValue Value)
{
  tWirthFloat HelpFloat;
  WirthsDevice32 Help32;
  WirthsDevice16 Help16;

  ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Value.ValType;

  switch (Value.ValType)
  {
  case BoolType:
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Value.UU.BoolVal;
    break;

  case RealType:
    HelpFloat.FloatVal = Value.UU.RealVal;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
HelpFloat.U0.msb;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
HelpFloat.U0.nmsb;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
HelpFloat.U0.nlsb;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
HelpFloat.U0.lsb;
    break;

  case DIntType:
  case DWordType:
    Help32.L = Value.UU.DIntVal;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Help32.U1.msb;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Help32.U1.nmsb;
```

```
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Help32.U1.nlsb;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Help32.U1.lsb;
    break;

  case IntType:
    Help16.SW = Value.UU.IntVal;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Help16.U1.msb;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Help16.U1.lsb;
    break;

  case UIntType:
  case WordType:
    Help16.W = Value.UU.IntVal;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Help16.U1.msb;
    ((rOctetString*) pResMsg->OctetString)->buf[pResMsg->pos++] =
Help16.U1.lsb;
    break;

  case StringType:
    PackString(pResMsg, GetCharArrayRef(Value.UU.StringVal));
    break;

  default:
    break;
  }
}

void TaskScheduleCommServer::PackString(rCodedListOfData* pMsg, const
char* Str)
{
  int len = 0;

  if (Str != NULL)
  {
    len = strlen(Str);
  }
  assert(len <= 255);

  ((rOctetString*) pMsg->OctetString)->buf[pMsg->pos++] = len;
  for (int i = 0; i < len; i++)
  {
    ((rOctetString*) pMsg->OctetString)->buf[pMsg->pos++] = Str[i];
  }
}

int TaskScheduleCommServer::ValueSize(tValue Value) const
{
  int Size = 1;
  switch (Value.ValType)
  {
    case BoolType:
      Size += 1;
```

```
      break;
    case RealType:
    case DIntType:
    case DWordType:
      Size += 4;
      break;
    case IntType:
    case UIntType:
    case WordType:
      Size += 2;
      break;
    case StringType:
      Size += strlen(GetCharArrayRef(Value.UU.StringVal)) + 1;
      break;
    default:
      break;
  }
  return Size;
}
```

6.2 Source code TSCC

TaskScheduleCommClient.h

```
/*************************************************************************
*******
*
* Filename:      TaskScheduleCommClient.h
*
* Author:        Martin Furmanski
* Responsible dept: ATPA/XA/ACN
* Creation date: 2008-01-30
*
* Products:         CB
* SIL level:        nonSIL
*
* Definitions of classes:
*
*************************************************************************
*******/
#ifndef _TaskScheduleCommClient_H
#define _TaskScheduleCommClient_H

#include <heap/heap_gtc.h>
#include <XMLUTILITIES/XMLSTREAM_GCL.H>
#include
<ApplicationLayer/EvaluationReport/EvaluationReport/DifferencesCollecto
r.h>
#include <varuser/varuser_gtc.h>
#include <transac/transac_gtc.h>
#include <asn1types/asn1types_gtc.h>
#include <asn1/asn1_gtc.h>
#include <strings/stringtype_ftc.h>
#include <appexecution/IAppRT_ProgControl_gtc.h>
#include <mmsuser/transactioninstance_ftc.h>
#include <varaccesssource/VAComliLEGTables.h>

#include <list>

#include "C:\root\Martin\TEST2\TEST2\TEST2.h"
#include "C:\root\Martin\TEST2\TEST2\scan_structs.h"

#define STARTIMMEDIATE 1
#define STARTTASK 2
#define STARTEVENT 3

class TaskScheduleCommClient : public MMSTransacHandler {
public:
  static void AssocDescConnected(pAssocDesc eAssocDesc);
  static void AssocDescAborted(pAssocDesc eAssocDesc);

  /* No other functions may be called before this has been called. Not
thread safe.
   * May/should only be called once, subsequent calls does nothing.
```

```
   * Sets up the link between CB and PM for communication of task
execution data.
  */
  static TaskScheduleCommClient* Instance(); // Create single instance
  void Connect();

  void WriteInd(pAssoc eAssoc, int BufPos, int LastPos, UnsignedByte*
UnconstrainedAddr); // Incomming MMS packet
  void SendEnableTSMessage(int when, char* task, int evt); // Sends
network message starting the data collection
  void UnPackValue(UnsignedByte* Data, int& Pos, tValue& Value);
  pString CreateAndUnpackString(UnsignedByte* Data, int& Pos);
  void SetRemSys(pString str);

  void Confirmation(tOperationStatus Status);
  void Confirmation(pItem ListOfAccResult);
  void ProgInvAttrConfirmation(tProgramInstanceState State,
                               pItem ListOfDomainNames,
                               bool MMSDeletable,
                               bool Reusable,
                               bool Monitor,
                               pString ExecutionArgument);

  void Start(int when, char* task, int evt); // Start a data collection
  void StartAtTask(HWND hWnd, HINSTANCE hInst); // Start at task dialog
  void StartAtEvent(HWND hWnd, HINSTANCE hInst); // Start at event
dialog
private:
  TaskScheduleCommClient();
  ~TaskScheduleCommClient();

  CComPtr<ITaskVisualizer>* tv;
  pAssocDesc m_eAssocDesc;
  static TaskScheduleCommClient* m_theTSCC;
  static const unsigned char cTSVersion = 1;
  bool m_Connected;
  bool m_Connecting;
  pString m_pRemSys;
  bool m_WriteOnce;
  std::list<BYTE*> data;
  std::list<int> datalen;

  int nTasks;
  int readTasks;
  bool m_sendEnableAtConnect;
  bool m_Collecting;
  bool gotFirstResponse;
  bool gotAllOfHeader;
};

#endif // _TaskScheduleCommClient_H
```

TaskScheduleCommClient.cpp

```
#include "TaskScheduleCommClient.h"

#include <heap/heap_gtc.h>
#include <varuser/varuser_gtc.h>
#include <transac/transac_gtc.h>
#include <asn1types/asn1types_gtc.h>
#include <asn1/asn1_gtc.h>
#include <strings/stringtype_ftc.h>
#include <appexecution/IAppRT_ProgControl_gtc.h>
#include <values/values_gfv.h>
#include <hwinterface/hwinterface_gtc.h>
#include <progcontrol/programinstance_ftc.h>
#include <progcontrol/scangroup_ftc.h>

TaskScheduleCommClient* TaskScheduleCommClient::m_theTSCC = NULL;
INT_PTR CALLBACK  ChooseTaskDlgProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK  ChooseEventDlgProc(HWND, UINT, WPARAM, LPARAM);

TaskScheduleCommClient* TaskScheduleCommClient::Instance()
{
  if(m_theTSCC == NULL) {
    m_theTSCC = new TaskScheduleCommClient();
  }

  return m_theTSCC;
}

void TaskScheduleCommClient::SetRemSys(pString str)
{
  m_pRemSys = str;
}
/* Connect to the controller. */
void TaskScheduleCommClient::Connect()
{
  if(m_Connected || m_pRemSys == NULL)
    return;

  tIdentifier DummyChannel;
  unsigned int InternalConnectionRef; // Not used
  int Status;

  InitMMSIdentifier(&DummyChannel);
  IntegerToMMSIdentifier(1, DummyChannel);


  InternalDefineConnection(cMMSProtocol, DummyChannel,
                           m_pRemSys, &m_eAssocDesc,
                           &InternalConnectionRef, &Status, false);


  if(Status != cSuccess)
    printf("TSCC::Connection failed\n");
```

```
  if (m_eAssocDesc != NULL && GetAssocDescState(m_eAssocDesc) ==
Connected)
  {
    m_Connected = true; // The connection was already established
    m_Connecting = false;
  }
}


TaskScheduleCommClient::TaskScheduleCommClient() :
  m_Connected(false),
  m_Connecting(true),
  m_WriteOnce(false),
  m_sendEnableAtConnect(false),
  m_Collecting(false),
  tv(0),
  m_pRemSys(0)
{

}
/* Dialog used for chosing the task for triggering data collection.*/
void TaskScheduleCommClient::StartAtTask(HWND hWnd, HINSTANCE hInst)
{
  HWND dlg = CreateDialog(GetModuleHandle("resources_enu.dll"),
MAKEINTRESOURCE(IDD_CHOOSETASK), hWnd, ChooseTaskDlgProc);
  ShowWindow(dlg, SW_SHOW);
}

/* Dialog used for chosing the custom event number for triggering data
collection.*/
void TaskScheduleCommClient::StartAtEvent(HWND hWnd, HINSTANCE hInst)
{
  HWND dlg = CreateDialog(GetModuleHandle("resources_enu.dll"),
MAKEINTRESOURCE(IDD_CHOOSEEVENT), hWnd, ChooseEventDlgProc);
  ShowWindow(dlg, SW_SHOW);
}

/* Start data collection immediately. */
void TaskScheduleCommClient::Start(int when, char* task, int evt)
{
  if(tv == 0) {
    tv = new CComPtr<ITaskVisualizer>;
    ASSERT(tv);
    HRESULT hr = tv->CoCreateInstance(OLESTR("TEST2.TaskVisualizer"));
    ASSERT(SUCCEEDED(hr));
  }

  if(!m_Connected && !m_Collecting) {
    m_Collecting = true;
    gotFirstResponse = false;

    Connect();
    if(m_Connected)
      SendEnableTSMessage(when, task, evt);
    else
      m_sendEnableAtConnect = true;
  }
```

```
  if(m_Connected && !m_Collecting) {
    m_Collecting = true;
    gotFirstResponse = false;

    SendEnableTSMessage(when, task, evt);
  }
}


TaskScheduleCommClient::~TaskScheduleCommClient()
{
}

void TaskScheduleCommClient::Confirmation(tOperationStatus Status)
{
}


void TaskScheduleCommClient::Confirmation(pItem ListOfAccResult)
{
}

void
TaskScheduleCommClient::ProgInvAttrConfirmation(tProgramInstanceState
State,
                                                pItem ListOfDomainNames,
                                                bool MMSDeletable,
                                                bool Reusable,
                                                bool Monitor,
                                                pString ExecutionArgument)
{
}

/* Connection established. */
void TaskScheduleCommClient::AssocDescConnected(pAssocDesc eAssocDesc)
{
  if (Instance()->m_eAssocDesc == eAssocDesc)
  {
      Instance()->m_Connected = true;
      Instance()->m_Connecting = false;
  }
}

void TaskScheduleCommClient::AssocDescAborted(pAssocDesc eAssocDesc)
{
  if (Instance()->m_eAssocDesc == eAssocDesc)
  {
    Instance()->m_Connected = false;
  }
}

/* Send network message requesting start of data collection.*/
void TaskScheduleCommClient::SendEnableTSMessage(int when, char* task,
int evt)
{
  rCodedListOfData* pMsg = NULL;
```

```
  CreateCodedListOfData((pCodedListOfData*) &pMsg,
NormalCodedListOfData);
  Assert(pMsg != NULL);

  tIdentifier ProgName;
  ProgName.SetContents("");

  PackPICHeader(ProgName, TSKind, pMsg->OctetString, &pMsg->pos);

  ((rOctetString*) pMsg->OctetString)->buf[pMsg->pos++] = when;
  switch(when) {
    case STARTIMMEDIATE:
      break;
    case STARTTASK:
      {
        int len = strlen(task);
        ((rOctetString*) pMsg->OctetString)->buf[pMsg->pos++] = len;

        while(len-->0)
          ((rOctetString*) pMsg->OctetString)->buf[pMsg->pos++] =
*(task++);
      }
      break;
    case STARTEVENT:
      ((rOctetString*) pMsg->OctetString)->buf[pMsg->pos++] = evt;
      break;
    default:
      assert(0);
      break;
  }

  PutTSReqAction((rAssocDescription*) m_eAssocDesc, pMsg);
}

/* Incoming MMS packet.*/
void TaskScheduleCommClient::WriteInd(pAssoc eAssoc, int FirstPos, int
LastPos, UnsignedByte* Data)
{
  int Pos = FirstPos;
  Pos++; // skip null from progname

  if(!m_Collecting) {
    printf("Got response without requesting it, error!\n");
  }


  unsigned char DiffDataVersion = Data[Pos++];
  if (DiffDataVersion != cTSVersion)
  {
    if (!m_WriteOnce)
    {
      WriteInformation();
      printf("Incompatible TS data format. Got: %d  Expected: %d\n",
DiffDataVersion, cTSVersion);
      m_WriteOnce = true;
    }
```

```
    return;
  }


  bool morefollows = Data[Pos++];
  unsigned char* buf;

  if(!gotFirstResponse) {
    gotAllOfHeader = false;
    gotFirstResponse = true;
    data.clear();
    datalen.clear();

    tValue val;
    UnPackValue(Data, Pos, val);
    nTasks = val.UU.DIntVal; // DIntVal = 4 byte

    buf = new BYTE[4];
    memcpy(buf, &val.UU.DIntVal, 4);
    data.push_back(buf);
    datalen.push_back(4);

    readTasks = 0;
  }

  if(!gotAllOfHeader) {
    tValue tscsId, prio, name;

    while(Pos < LastPos && readTasks < nTasks) {
      UnPackValue(Data, Pos, tscsId); // DIntVal 4 byte
      UnPackValue(Data, Pos, prio); // DIntVal 4 byte
      UnPackValue(Data, Pos, name); // StringVal = pString
      readTasks++;

      assert(((rString*)name.UU.StringVal)->CurrLength <= 255);

      WirthsDevice32 help32;
      help32.L = ((rString*)name.UU.StringVal)->CurrLength;

      buf = new BYTE[4 + 4 + 1 + help32.U1.lsb];
      memcpy(buf, &tscsId.UU.DIntVal, 4);
      memcpy(buf+4, &prio.UU.DIntVal, 4);
      memcpy(buf+8, &help32.U1.lsb, 1);
      memcpy(buf+9,&((rString*)name.UU.StringVal)->UU.Characters[0]
,help32.U1.lsb);

      data.push_back(buf);
      datalen.push_back(4 + 4 + 1 + help32.U1.lsb);
    }


    if(readTasks == nTasks)
      gotAllOfHeader = true;
  }

  if(gotAllOfHeader && LastPos>Pos) {
    buf = new BYTE[LastPos-Pos];
```

```
    memcpy(buf, &Data[Pos], LastPos-Pos);

    data.push_back(buf);
    datalen.push_back(LastPos-Pos);
  }

  if(!morefollows) {
    m_Collecting = false;
    int totalBytes = 0;

    std::list<int>::iterator itr_len = datalen.begin();

    while(itr_len != datalen.end()) {
      totalBytes += *itr_len;
      itr_len++;
    }

    BYTE* allData = new BYTE[totalBytes];

    itr_len = datalen.begin();
    std::list<BYTE*>::iterator itr_data = data.begin();


    int allPos = 0;
    while(itr_data != data.end()) {
      memcpy(&allData[allPos], *itr_data, *itr_len);
      delete *itr_data;

      allPos += *itr_len;

      itr_data++;
      itr_len++;
    }

    data.clear();
    datalen.clear();

    (*tv)->PutData(totalBytes, allData);
  }
}

void TaskScheduleCommClient::UnPackValue(UnsignedByte* Data, int& Pos,
tValue& Value)
{
  tWirthFloat HelpFloat;
  WirthsDevice32 Help32;
  WirthsDevice16 Help16;

  Value.ValType = (tValType) Data[Pos++];
  switch (Value.ValType)
  {
  case BoolType:
    Value.UU.BoolVal = Data[Pos++];
    break;

  case RealType:
    HelpFloat.U0.msb = Data[Pos++];
```

```
    HelpFloat.U0.nmsb = Data[Pos++];
    HelpFloat.U0.nlsb = Data[Pos++];
    HelpFloat.U0.lsb = Data[Pos++];
    Value.UU.RealVal = HelpFloat.FloatVal;
    break;

  case DIntType:
  case DWordType:
    Help32.U1.msb = Data[Pos++];
    Help32.U1.nmsb = Data[Pos++];
    Help32.U1.nlsb = Data[Pos++];
    Help32.U1.lsb = Data[Pos++];
    Value.UU.DIntVal = Help32.L;
    break;

  case IntType:
    Help16.U1.msb = Data[Pos++];
    Help16.U1.lsb = Data[Pos++];
    Value.UU.IntVal = Help16.SW;
    break;

  case UIntType:
  case WordType:
    Help16.U1.msb = Data[Pos++];
    Help16.U1.lsb = Data[Pos++];
    Value.UU.IntVal = Help16.W;
    break;

  case StringType:
    Value.UU.StringVal = CreateAndUnpackString(Data, Pos);
    break;

  case UndefType:
    // No value exists, just init the variable
    Value.UU.DIntVal = 0;
    break;

  default:
    break;
  }
}

pString TaskScheduleCommClient::CreateAndUnpackString(UnsignedByte*
Data, int& Pos)
{
  pString Str = NULL;

  int Size = Data[Pos++];
  if (Size > 0)
  {
    Str = NewString(Size);
    CopyChBuffToString(&Data[Pos], Size, Str);
    Pos += Size;
  }
  return Str;
}
```

```
INT_PTR CALLBACK ChooseTaskDlgProc(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
{
      UNREFERENCED_PARAMETER(lParam);
      switch (message)
      {
      case WM_INITDIALOG:
            {
      unsigned int nTasks = 0;
                  // Populate combo box
                  HWND combo = GetDlgItem(hDlg, IDC_CHOOSETASK_COMBO);

      rProgramInstance* eProgramInstance =
(rProgramInstance*)First(GetProgramInvocationList());
      while (eProgramInstance != NULL)
      {
        rScanGroupSystem* puTaskSystem =
(rScanGroupSystem*)First(eProgramInstance->SystemList);
        while (puTaskSystem != NULL)
        {
          rScanGroupData* puTaskData =
(rScanGroupData*)First(puTaskSystem->ScanGroupList);
          while (puTaskData != NULL)
          {
            nTasks++;

            if(puTaskData->ScanGroupType == ScanGroupControlType) {
              rString* name = (rString*)puTaskData->Name_;
              char* c = &name->UU.Characters[0];
              char *buf = new char[name->CurrLength+1];
              memcpy(buf, c, name->CurrLength);
              buf[name->CurrLength] = 0;

              SendMessage(combo, CB_ADDSTRING, 0, (LPARAM)buf);
            }

            puTaskData = SuccElem(puTaskData);
          }
          puTaskSystem = SuccElem(puTaskSystem);
        }
        eProgramInstance = SuccElem(eProgramInstance);
      }
      if(nTasks>=1)
                    SendMessage(combo, CB_SETCURSEL, 0, 0);
      else
        DestroyWindow(hDlg);
            }
            return (INT_PTR)TRUE;
    break;

      case WM_COMMAND:
            if (LOWORD(wParam) == IDCANCEL)
            {
                  DestroyWindow(hDlg);
                  return (INT_PTR)TRUE;
    } else if(LOWORD(wParam) == IDOK) {
      // Get selection, call TSCS method
```

```
        char* task;

        HWND combo = GetDlgItem(hDlg, IDC_CHOOSETASK_COMBO);
        LRESULT index = SendMessage(combo,CB_GETCURSEL,0,0);
        task = new char[SendMessage(combo,CB_GETLBTEXTLEN,index, 0)+1];
        SendMessage(combo, CB_GETLBTEXT, index, (LPARAM)task);

        DestroyWindow(hDlg);
        TaskScheduleCommClient::Instance()->Start(STARTTASK, task, 0);
        delete [] task;

                        return (INT_PTR)TRUE;
    }
                break;
        } // end switch(message)


        return (INT_PTR)FALSE;
}

INT_PTR CALLBACK ChooseEventDlgProc(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
{
        UNREFERENCED_PARAMETER(lParam);
        switch (message)
        {
        case WM_INITDIALOG:
                {
                }
                return (INT_PTR)TRUE;
    break;

        case WM_COMMAND:
                if (LOWORD(wParam) == IDCANCEL)
                {
                        DestroyWindow(hDlg);
                        return (INT_PTR)TRUE;
    } else if(LOWORD(wParam) == IDOK) {
        // Get selection, call TSCS method
        HWND edit = GetDlgItem(hDlg, IDC_CHOOSEEVENT_EDIT);
        int len = SendMessage(edit, WM_GETTEXTLENGTH, 0, 0)+1;
        char* sevt = new char[len];
        SendMessage(edit,WM_GETTEXT, len, (LPARAM)sevt);

        int evt = atoi(sevt);

        DestroyWindow(hDlg);
        TaskScheduleCommClient::Instance()->Start(STARTEVENT, 0, evt);

                        return (INT_PTR)TRUE;
    }
                break;
        } // end switch(message)


        return (INT_PTR)FALSE;
}
```

6.3 Source code Visualizer

Only the most central and interesting files have been included in this appendix.

scan_structs.h

```
#ifndef __SCAN_STRUCTS__
#define __SCAN_STRUCTS__

enum { SCAN_START_NORMAL, SCAN_START_DELAYED, SCAN_END,
USER_EVENT_FIRST, USER_EVENT_LAST=255 }; // States for raw data-array

#endif
```

TaskSim.h

```
#ifndef _TASK_SIM_H_
#define _TASK_SIM_H_

#include <set>
#include <string>
#include <functional>
#include <list>

using namespace std;

#include "scan_structs.h"

enum states { STATE_UNKNOWN, STATE_ACTIVE, STATE_BLOCKED, STATE_IDLE};
enum events { EVENT_BLOCK, EVENT_ACTIVE, EVENT_END, CUSTOM_EVENT};

class scanevent {
public:
	events evt;
	unsigned int id;
	unsigned int time;
	unsigned int extra;

	scanevent(events _evt = EVENT_ACTIVE, int _id = 0, int _time = 0,
int _extra = 0) : evt(_evt), id(_id), time(_time), extra(_extra) {};
};

class Task {
public:
	unsigned int id;
	unsigned int prio;
    char* name;
    unsigned int state;

public:
	Task(unsigned int _id, unsigned int _prio=0, char* _name=0,
unsigned int _state=STATE_UNKNOWN) : id(_id), prio(_prio), name(_name),
state(_state)  {}
    ~Task() { if(name) delete [] name; }
    class byprio { public: bool operator()(const Task* a, const Task*
b) const { return a->prio < b->prio; } };
    class byid { public: bool operator()(const Task* a, const Task* b)
const { return a->id < b->id; } };
};

class TaskSim
{
public:
  typedef set<Task*, Task::byprio> tasks_byprio_t;
  typedef set<Task*, Task::byid> tasks_byid_t;

  tasks_byprio_t tasks_byprio;
  tasks_byid_t tasks_byid;

  typedef list<scanevent> scaneventset_t;
```

```
  scaneventset_t ses;
  scaneventset_t::iterator currentEvent;

  std::list<Task*> active;

  string error_str;

  TaskSim() : currentEvent(ses.begin()) {};
  ~TaskSim();

  void InsertTask(unsigned int id, int prio, char* name);
  void InsertEvent(unsigned int id, int time, int evt);
  void InsertEventInternalFormat(unsigned int id, int time, int evt,
int extra);
  bool ExistsTask(unsigned int id);
  Task* TaskSim::FindTask(unsigned int id);

  bool InitForDraw();
  bool NoMoreEvents();
  unsigned int GetFirstTime();
  unsigned int GetLastTime();

  void fireEvent(unsigned int id, events evt);
  void fireNext();

  void error(char* error) { error_str.append(error); }
  void clear_error() { error_str.clear(); };
  void printTaskState();

  void clear();
};

#endif // _TASK_SIM_H_
```

TaskSim.cpp

```
#include "StdAfx.h"
#include "TaskSim.h"

void TaskSim::clear()
{
  tasks_byprio.clear();
  tasks_byid.clear();

  ses.clear();
  active.clear();
}

Task* TaskSim::FindTask(unsigned int id)
{
  Task temp(id);
  tasks_byid_t::iterator itr = tasks_byid.find(&temp);

  if(itr == tasks_byid.end() || (*itr)->id != id)
    return 0;
  else
    return *itr;
}

bool TaskSim::ExistsTask(unsigned int id)
{
  Task temp(id);
  tasks_byid_t::iterator itr = tasks_byid.find(&temp);

  if(itr == tasks_byid.end() || (*itr)->id != id)
    return false;
  else
    return true;
}

void TaskSim::InsertTask(unsigned int id, int prio, char* name)
{
  if(!ExistsTask(id)) {
    Task* task = new Task(id, prio, name);

    tasks_byid.insert(task);
    tasks_byprio.insert(task);
  }

  ATLASSERT(ExistsTask(id));
}

void TaskSim::InsertEvent(unsigned int id, int time, int evt)
{
  ATLASSERT(ExistsTask(id));
  if(!ExistsTask(id))
    return;

  if(evt == SCAN_START_NORMAL) {
    ses.push_back(scanevent(EVENT_ACTIVE, id, time));
```

```
  } else if(evt == SCAN_END) {
    ses.push_back(scanevent(EVENT_END, id, time));
  } else if(evt >= USER_EVENT_FIRST) {
    ses.push_back(scanevent(CUSTOM_EVENT, id, time, evt));
  }
}

void TaskSim::InsertEventInternalFormat(unsigned int id, int time, int
evt, int extra)
{
  ATLASSERT(ExistsTask(id));
  if(!ExistsTask(id))
    return;
  if(evt == 1) {
    ses.push_back(scanevent(EVENT_ACTIVE, id, time));
  } else if(evt == 2) {
    ses.push_back(scanevent(EVENT_END, id, time));
  } else if(evt == 3) {
    ses.push_back(scanevent(CUSTOM_EVENT, id, time, extra));
  }
}

TaskSim::~TaskSim(void)
{
  clear();
}

void TaskSim::printTaskState()
{
  TaskSim::tasks_byprio_t::const_iterator titr = tasks_byprio.begin();

  printf("\nTASK STATE\n");
  while(titr!=tasks_byprio.end()) {
    printf("id: %d, prio: %d state: %d\n", (*titr)->id, (*titr)->prio,
(*titr)->state);
    titr++;
  }
  printf("\n");
}

bool TaskSim::InitForDraw()
{
  if(ses.size() < 2)
    return false;

  else {
    TaskSim::tasks_byprio_t::iterator titr = tasks_byprio.begin();

    while(titr!=tasks_byprio.end()) {
      (*titr)->state = STATE_UNKNOWN;
      titr++;
    }

    currentEvent = ses.begin();
    active.clear();

    return true;
```

```
  }
}

unsigned int TaskSim::GetFirstTime()
{
  return ses.front().time;
}

unsigned int TaskSim::GetLastTime()
{
  return ses.back().time;
}

bool TaskSim::NoMoreEvents()
{
  return (currentEvent == ses.end());
}

void TaskSim::fireNext()
{
  if(currentEvent != ses.end()) {
    if(currentEvent->evt != CUSTOM_EVENT) {
        fireEvent(currentEvent->id, currentEvent->evt);
    }

    currentEvent++;
  }
}

// Update task states according to event
void TaskSim::fireEvent(unsigned int id, events evt)
{
  Task* task = FindTask(id);

      if(task == NULL) {
            error("task id not found;");
            printf("taskid not found: %d\n", id);
            return;
      }

      switch(evt) {
            case EVENT_ACTIVE:
                  if(!active.empty())
                        active.front()->state = STATE_BLOCKED;

                  task->state = STATE_ACTIVE;
                  active.push_front(task);
                  break;
            case EVENT_BLOCK:
                  // NOT IMPLEMENTED
                  break;
            case EVENT_END:
      if(!active.empty()) {
                    active.pop_front();

        if(!active.empty())
```

```
                        active.front()->state = STATE_ACTIVE;

        }

        task->state = STATE_IDLE;
                break;
        }
}
```

TaskVisualizer.h (COM class)

```
// TaskVisualizer.h : Declaration of the CTaskVisualizer

#pragma once
#include "resource.h"       // main symbols

#include "TEST2.h"
#include "Visualizer.h"

#if defined(_WIN32_WCE) && !defined(_CE_DCOM) &&
!defined(_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA)
#error "Single-threaded COM objects are not properly supported on
Windows CE platform, such as the Windows Mobile platforms that do not
include full DCOM support. Define
_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA to force ATL to support
creating single-thread COM object's and allow use of it's single-
threaded COM object implementations. The threading model in your rgs
file was set to 'Free' as that is the only threading model supported in
non DCOM Windows CE platforms."
#endif



// CTaskVisualizer

class ATL_NO_VTABLE CTaskVisualizer :
	public CComObjectRootEx<CComSingleThreadModel>,
	public CComCoClass<CTaskVisualizer, &CLSID_TaskVisualizer>,
	public IDispatchImpl<ITaskVisualizer, &IID_ITaskVisualizer,
&LIBID_TEST2Lib, /*wMajor =*/ 1, /*wMinor =*/ 0>
{
public:
	CTaskVisualizer()
	{
	}

DECLARE_REGISTRY_RESOURCEID(IDR_TASKVISUALIZER)


BEGIN_COM_MAP(CTaskVisualizer)
	COM_INTERFACE_ENTRY(ITaskVisualizer)
	COM_INTERFACE_ENTRY(IDispatch)
END_COM_MAP()



	DECLARE_PROTECT_FINAL_CONSTRUCT()

	HRESULT FinalConstruct()
	{
		HANDLE msgQEvent = CreateEvent(0, true, false, 0);
		CreateThread(0, 0, UserInterfaceThread, msgQEvent, 0,
&threadId);
		WaitForSingleObject(msgQEvent, INFINITE);
		return S_OK;
```

```
        }

        void FinalRelease()
        {
        }

private:
        DWORD threadId;

public:

public:
        STDMETHOD(PutData)(unsigned int n, BYTE* data);
};

OBJECT_ENTRY_AUTO(__uuidof(TaskVisualizer), CTaskVisualizer)
```

Visualizer.h (graphical visualization)

```
#pragma once
#include <set>
#include <map>
#include "scan_structs.h"
#include "TaskSim.h"
#include "resource.h"

using namespace std;

extern class Visualizer* visualizer;

DWORD WINAPI UserInterfaceThread(LPVOID lpParam);

class Visualizer : public CFrameWindowImpl<Visualizer>
{
public:
  DECLARE_FRAME_WND_CLASS(_T("Visualizer Window"), IDR_TASKVISUALIZER);

  BEGIN_MSG_MAP_EX(Visualizer)
    MSG_WM_CREATE(OnCreate)
    MSG_WM_DESTROY(OnDestroy)
    MSG_WM_SIZE(OnSize)
    MSG_WM_COMMAND(OnCommand)
    MSG_WM_PAINT(Draw)
    MSG_WM_HSCROLL(OnHScroll)
    MSG_WM_VSCROLL(OnVScroll)
    CHAIN_MSG_MAP(CFrameWindowImpl<Visualizer>)
    END_MSG_MAP()

    void OnSize(UINT nType, CSize size)
  {
    UNREFERENCED_PARAMETER(nType);
    UNREFERENCED_PARAMETER(size);

    Invalidate(false);
  }

  void OnHScroll(UINT nSBCode, UINT nPos, HWND pScrollBar)
  {
    UNREFERENCED_PARAMETER(pScrollBar);

    if(nSBCode == SB_THUMBPOSITION || nSBCode ==  SB_THUMBTRACK) {
      int min, max;

      GetScrollRange(SB_HORZ, &min, &max);

      scrollH = double(nPos-min)/(max-min);
      SetScrollPos(SB_HORZ, nPos,1);
      Invalidate(false);
    } else if(nSBCode == SB_ENDSCROLL) {
      SetScrollPos(SB_HORZ, GetScrollPos(SB_HORZ), 1);
      Invalidate(false);
    }
  }
```

```
void OnVScroll(UINT nSBCode, UINT nPos, HWND pScrollBar)
{
  UNREFERENCED_PARAMETER(pScrollBar);

  if(nSBCode == SB_THUMBPOSITION || nSBCode ==  SB_THUMBTRACK) {
    int min, max;

    GetScrollRange(SB_VERT, &min, &max);

    scrollV = double(nPos-min)/(max-min);
    SetScrollPos(SB_VERT, nPos,1);
    Invalidate(false);
  } else if(nSBCode == SB_ENDSCROLL) {
    SetScrollPos(SB_VERT, GetScrollPos(SB_VERT), 1);
    Invalidate(false);
  }
}

void OnCommand(UINT uNotifyCode, int nID, CWindow wndCtl)
{
  UNREFERENCED_PARAMETER(wndCtl);
  UNREFERENCED_PARAMETER(uNotifyCode);

  switch(nID) {
    case ID_GRIDLENGTH_100MS:
      GridLength = 100;
      break;
    case ID_GRIDLENGTH_1SECOND:
      GridLength = 1000;
      break;
    case ID_GRIDLENGTH_5SECONDS:
      GridLength = 5000;
      break;
    case ID_GRIDLENGTH_10SECONDS:
      GridLength = 10000;
      break;
    case ID_GRIDLENGTH_30SECONDS:
      GridLength = 30000;
      break;
    case ID_GRIDLENGTH_1MINUTE:
      GridLength = 60000;
      break;
    case ID_GRIDLENGTH_2MINUTES:
      GridLength = 120000;
      break;
    case ID_GRIDLENGTH_5MINUTES:
      GridLength = 300000;
      break;
    case ID_FILE_IMPORT:
      Import();
      break;
    case ID_FILE_EXPORT:
      Export();
      break;
  }
  Invalidate(false);
```

```
  SetMsgHandled(false);
}

bool ReadLine(HANDLE hf, char* buf, DWORD maxLen, DWORD* bytesRead)
{
  DWORD i=0;
  bool done = false;

  while(i<maxLen && !done) {
    if(ReadFile(hf, &buf[i], 1, bytesRead, 0)) {
      if(buf[i] == '\n' || *bytesRead == 0) {
        buf[i] = 0;
        done = true;
      } else
        i++;

    } else {
      return false;
    }
  }

  *bytesRead = i;
  return true;
}

char* Part(char *src, char *dst) {
  while(*src != '\t' && *src != 0) {
    *dst = *src;
    dst++; src++;
  }

  *dst = 0;

  return ++src;
}

// Import saved data collection
void Import()
{
  OPENFILENAME ofn; // common dialog box structure
  WCHAR szFile[260]; // buffer for file name
  HANDLE hf; // file handle

  // Initialize OPENFILENAME
  ZeroMemory(&ofn, sizeof(ofn));
  ofn.lStructSize = sizeof(ofn);
  ofn.hwndOwner = m_hWnd;
  ofn.lpstrFile = szFile;
  //
  // Set lpstrFile[0] to '\0' so that GetOpenFileName does not
  // use the contents of szFile to initialize itself.
  //
  ofn.lpstrFile[0] = '\0';
  ofn.nMaxFile = sizeof(szFile);
  ofn.lpstrFilter = L"Tabulated data (*.txt)\0*.txt\0";
  ofn.nFilterIndex = 1;
  ofn.lpstrFileTitle = NULL;
```

```
ofn.nMaxFileTitle = 0;
ofn.lpstrInitialDir = NULL;
ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;

// Display the Open dialog box.

if (GetSaveFileName(&ofn)==TRUE) {
  CComCritSecLock<CComAutoCriticalSection> Lock (cs);
  ts.clear();

  hf = CreateFile(ofn.lpstrFile, GENERIC_READ,
    FILE_SHARE_READ, (LPSECURITY_ATTRIBUTES) NULL,
    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
    (HANDLE) NULL);

  if(hf != INVALID_HANDLE_VALUE) {
    char buf[1000], part[1000];
    DWORD nBytes;

    bool error = false;
    bool done;

    if(ReadLine(hf, buf, sizeof(buf), &nBytes)) {
      if(strcmp(buf, "ID\tPRIO\tNAME") == 0) {
        done = false; // if done reading header

        while(!done) {
          if(ReadLine(hf, buf, sizeof(buf), &nBytes)) {
            if(nBytes == 0) {
              done = true;
            } else {
              int id, prio;
              char *pos = &buf[0];

              pos = Part(pos, part);
              id = atoi(part);

              pos = Part(pos, part);
              prio = atoi(part);

              pos = Part(pos, part);
              size_t len = strlen(part)+1;
              char* name = new char[len];
              if(name)
                strcpy_s(name, len, part);

              ts.InsertTask(id, prio, name);
            }
          } else {
            done = true;
            error = true;
          }
        } // header

        // DONE READING HEADER

        if(!error) {
```

```
            if(ReadLine(hf, buf, sizeof(buf), &nBytes)) {
              if(strcmp(buf, "TIME\tID\tEVENT") == 0) {
                done = false; // if done reading timeseries

                while(!done) {
                  if(ReadLine(hf, buf, sizeof(buf), &nBytes)) {
                    if(nBytes == 0) {
                      done = true;
                    } else {
                      int time, id, evt, extra=0;
                      char *pos = &buf[0];

                      pos = Part(pos, part);
                      time = atoi(part);

                      pos = Part(pos, part);
                      id = atoi(part);

                      pos = Part(pos, part);
                      evt = atoi(part);

                      if(evt == CUSTOM_EVENT) {
                        pos = Part(pos, part);
                        extra = atoi(part);
                      }

                      ts.InsertEventInternalFormat(id, time, evt,
extra);
                    }
                  } else {
                    done = true;
                    error = true;
                  }
                } // timeseries


              } else {
                error = true; // wrong title
              }
            } else {
              error = true; // read error
            }
          }
        } else {
          error = true;
        }
      }

      if(!error) {

      }

      CloseHandle(hf);
    }

  }
```

```
  }

  // Save data collection
  void Export()
  {
    OPENFILENAME ofn;       // common dialog box structure
    WCHAR szFile[260];       // buffer for file name
    HANDLE hf;              // file handle

    // Initialize OPENFILENAME
    ZeroMemory(&ofn, sizeof(ofn));
    ofn.lStructSize = sizeof(ofn);
    ofn.hwndOwner = m_hWnd;
    ofn.lpstrFile = szFile;
    //
    // Set lpstrFile[0] to '\0' so that GetOpenFileName does not
    // use the contents of szFile to initialize itself.
    //
    ofn.lpstrFile[0] = '\0';
    ofn.nMaxFile = sizeof(szFile);
    ofn.lpstrFilter = L"Tabulated data (*.txt)\0*.txt\0";
    ofn.nFilterIndex = 1;
    ofn.lpstrFileTitle = NULL;
    ofn.nMaxFileTitle = 0;
    ofn.lpstrInitialDir = NULL;
    ofn.Flags = OFN_PATHMUSTEXIST;

    // Display the Open dialog box.

    if (GetSaveFileName(&ofn)==TRUE) {
      hf = CreateFile(ofn.lpstrFile, GENERIC_WRITE,
        0, (LPSECURITY_ATTRIBUTES) NULL,
        CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL,
        (HANDLE) NULL);

      if(hf != INVALID_HANDLE_VALUE) {
        char buf[1000], buf2[100];
        DWORD lpNumberOfBytesWritten;
        TaskSim::tasks_byprio_t::const_iterator titr =
ts.tasks_byprio.begin();
        strcpy_s(buf, "ID\tPRIO\tNAME\n");
        WriteFile(hf, buf, DWORD(strlen(buf)), &lpNumberOfBytesWritten,
0);

        while(titr != ts.tasks_byprio.end())
        {
          sprintf_s(buf,"%d\t%d\t%s\n", (*titr)->id, (*titr)->prio,
(*titr)->name);
          titr++;
          WriteFile(hf, buf, DWORD(strlen(buf)),
&lpNumberOfBytesWritten, 0);
        }

        WriteFile(hf, "\n", 1, &lpNumberOfBytesWritten, 0);

        strcpy_s(buf, "TIME\tID\tEVENT\n");
```

```
        WriteFile(hf, buf, DWORD(strlen(buf)), &lpNumberOfBytesWritten,
0);
        TaskSim::scaneventset_t::iterator evt = ts.ses.begin();
        while(evt != ts.ses.end()) {
          if(evt->evt == CUSTOM_EVENT)
            sprintf_s(buf2, "%d", evt->extra);
          else
            sprintf_s(buf2, "-");

          sprintf_s(buf,"%d\t%d\t%d\t%s\n", evt->time, evt->id, evt-
>evt, buf2);

          evt++;
          WriteFile(hf, buf, DWORD(strlen(buf)),
&lpNumberOfBytesWritten, 0);
        }

        CloseHandle(hf);
      }

    }

  }

  LRESULT OnCreate(LPCREATESTRUCT lpcs)
  {
    UNREFERENCED_PARAMETER(lpcs);

    CreateSimpleToolBar();
    SetMsgHandled(false);
    return 0;
  }

  void OnDestroy()
  {
    SetMsgHandled(false);
  }

  /* Visualization algorithm, uses TaskSim */
  void Draw(HDC /*hdc*/)
  {
    CComCritSecLock<CComAutoCriticalSection> Lock (cs);
    PAINTSTRUCT ps;

    CRect rc;
    GetClientRect(&rc);

    if(cs.m_sec.LockCount > 0)
      ATLASSERT(false);

    if(!ts.InitForDraw() || rc.right <= 100) {
      BeginPaint(&ps);
      EndPaint(&ps);
      return;
    }

    BeginPaint(&ps);
```

```
    CDCHandle dc_real(GetDC());
    CDCHandle dc;

    HBRUSH activeBrush = CreateSolidBrush(RGB(50,110,50));
    HPEN activePen = CreatePen(PS_SOLID, 1, RGB(50,110,50));

    HBRUSH blockedBrush = CreateSolidBrush(RGB(225,200,200));
    HPEN blockedPen = CreatePen(PS_SOLID, 1, RGB(225,200,200));

    HPEN idlePen = CreatePen(PS_SOLID, 1, RGB(0,0,0));
    HPEN& blackPen = idlePen;

    HPEN unknownPen = CreatePen(PS_SOLID, 1, RGB(150,150,150));
    HPEN markPen = CreatePen(PS_DOT, 1, RGB(50,50,255));
    HPEN gridPen = CreatePen(PS_DOT, 1, RGB(160,240,160));

    dc.CreateCompatibleDC(dc_real.m_hDC);

    timewidth = (rc.right-100)*GridLength/100; //(ms)
    double ppt = (double)100/(double)GridLength;  //(pixel/ms)

    unsigned int firsteventtime = ts.GetFirstTime(); // should always
be zero
    unsigned int lasteventtime = ts.GetLastTime();

    if((lasteventtime - firsteventtime) < timewidth) {
      leftT = firsteventtime;
      rightT = leftT + timewidth;
      // disable scroller?
    } else {
      leftT = (int)(firsteventtime + (lasteventtime-timewidth-
firsteventtime)*scrollH);
      rightT = leftT+timewidth;

    }

    HBITMAP bmp;

    bmp = CreateCompatibleBitmap(dc_real.m_hDC, rc.Width(),
rc.Height());
    dc.SelectBitmap(bmp);

    dc.FillSolidRect(rc, RGB(255,255,255));

    unsigned int last_time = 0;

    { // vertical grid lines
      dc.SelectPen(gridPen);

      for(unsigned int xoffset = firsteventtime;xoffset < rightT;
xoffset+=GridLength) {
        if(xoffset>=leftT) {
          int x = (int)((xoffset-leftT)*ppt+0.5)+100;
          dc.MoveTo(x, 0);
          dc.LineTo(x, rc.bottom-40);
        }
```

```
      }
    }

    int x1 = 100;
    int x2;

    int htot = (int(ts.tasks_byprio.size())+1)*100; // height needed
for drawing vertically
    int ystart;

    if(htot<rc.bottom)
      ystart = 100;
    else
      ystart = 100-int((htot-rc.bottom)*scrollV+0.5);

    bool onlyOneMore = true;
    while(!ts.NoMoreEvents() && onlyOneMore) {
      if(!(last_time < rightT))
        onlyOneMore = false;

      x2 = int(((ts.currentEvent->time-leftT))*ppt+0.5)+100;

      if(x2 >= 100) {
        TaskSim::tasks_byprio_t::const_iterator titr =
ts.tasks_byprio.begin();

        int y = ystart;

        if(x1>=x2) x2=x1+1; // rightshift stacking events due to
temporal congestion (as to not overdraw previous)

        while(titr != ts.tasks_byprio.end()) {

          switch((*titr)->state) {
            case STATE_ACTIVE:
              dc.SelectPen(activePen);
              dc.SelectBrush(activeBrush);
              dc.Rectangle(x1, y, x2, y+18);

              dc.SelectPen(blackPen);
              dc.MoveTo(x1, y);
              dc.LineTo(x2, y);
              dc.MoveTo(x1, y+18);
              dc.LineTo(x2, y+18);
              break;
            case STATE_BLOCKED:
              dc.SelectPen(blockedPen);
              dc.SelectBrush(blockedBrush);
              dc.Rectangle(x1, y, x2, y+18);

              dc.SelectPen(blackPen);
              dc.MoveTo(x1, y);
              dc.LineTo(x2, y);
              dc.MoveTo(x1, y+18);
              dc.LineTo(x2, y+18);
              break;
```

```
            case STATE_IDLE:
              dc.SelectPen(idlePen);
              dc.MoveTo(x1, y+18);
              dc.LineTo(x2, y+18);
              break;
            case STATE_UNKNOWN:
              dc.SelectPen(unknownPen);
              dc.MoveTo(x1, y+18);
              dc.LineTo(x2, y+18);
              break;
          }

          if(ts.currentEvent->evt == CUSTOM_EVENT && ts.currentEvent-
>id == (*titr)->id) {
            dc.SelectPen(markPen);
            dc.MoveTo(x2-1, y+18+10);
            dc.LineTo(x2-1, y   -10);

            WCHAR buf[50];
            wsprintf(buf, L"%d", ts.currentEvent->extra);
            dc.TextOut(x2-2, y+18+10, buf);
          }


          y+= 100;
          titr++;
        }

        x1 = x2;
      }

      last_time = ts.currentEvent->time;
      ts.fireNext();
    }

    TaskSim::tasks_byprio_t::const_iterator titr =
ts.tasks_byprio.begin();

    int y = ystart+10;

    WCHAR buf[1024];
    while(titr != ts.tasks_byprio.end()) {
      Task* task = ts.FindTask((*titr)->id);
      wsprintf(buf, L"%hs", task->name);
      dc.TextOut(20, y, buf);

      y+= 100;
      titr++;
    }
    RECT low;
    low.left = 0;
    low.right = rc.right;
    low.top = rc.bottom-40;
    low.bottom = rc.bottom;

    dc.FillSolidRect(&low, RGB(255,255,255));
```

```
    { // x-axis time-text
      WCHAR buf[1024];

      for(unsigned int xoffset = firsteventtime;xoffset < rightT;
xoffset+=GridLength) {
        if(xoffset>=leftT) {
          int x = (int)((xoffset-leftT)*ppt+0.5)+100;

          dc.SelectPen(blackPen);
          dc.MoveTo(x, rc.bottom-40);
          dc.LineTo(x, rc.bottom-35);

          wsprintf(buf, L"%d", xoffset);

          dc.TextOut(x, rc.bottom-32, buf);
        }
      }
    }


    dc.SelectPen(blackPen);
    dc.MoveTo(99, 0);
    dc.LineTo(99, rc.bottom-40);
    dc.LineTo(rc.right, rc.bottom-40);

    dc_real.BitBlt(0,0,rc.right-rc.left, rc.bottom-rc.top, dc.m_hDC,
0,0, SRCCOPY);
    DeleteObject(bmp);

    DeleteObject(activeBrush);
    DeleteObject(blockedBrush);
    DeleteObject(activePen);
    DeleteObject(blockedPen);
    DeleteObject(idlePen);
    DeleteObject(unknownPen);
    DeleteObject(markPen);
    DeleteObject(gridPen);

    dc.DeleteDC();
    dc_real.DeleteDC();

    EndPaint(&ps);
  }

  void PutData(unsigned int n, BYTE* data)
  {
    CComCritSecLock<CComAutoCriticalSection> Lock (cs);

    if(!::IsWindow(m_hWnd))
      return;

    if(cs.m_sec.LockCount > 0)
      ATLASSERT(false);

    ts.clear();

    unsigned int i=0;
```

```
    int nTasks;

    nTasks = *((int*)(&data[i]));
    i+=4;

    // Header
    int iTask=0;
    while(iTask++<nTasks) {
      int tscsId, prio, nchar;
      char* name;

      tscsId = *((int*)(&data[i]));
      i+=4;

      prio = *((int*)(&data[i]));
      i+=4;

      nchar = data[i];
      i++;

      name = new char[nchar+1];
      memcpy(name, &data[i], nchar);
      name[nchar] = 0;
      i+= nchar;

      ts.InsertTask(tscsId, prio, name);
    }

    unsigned int time0 = *((int*)(&data[i+4]));

    while(i<n) {
      int tscsId, time, evt;

      tscsId = *((int*)(&data[i]));
      i+=4;

      time = *((int*)(&data[i]))-time0;

      i+=4;

      evt = data[i];
      i++;

      ts.InsertEvent(tscsId, time, evt);
    }

    InvalidateRect(NULL, false);
  }

  Visualizer(void);
  ~Visualizer(void);

private:
  TaskSim ts;

  unsigned long lastput;
  CComAutoCriticalSection cs;
```

```
  int GridLength; // 100px = GridLength ms

  int first;
  int last;
  unsigned int leftT, rightT;
  unsigned int timewidth;
  double scrollH, scrollV;
};
```

Visualizer.cpp (graphical visualization thread)

```
#include "StdAfx.h"
#include "Visualizer.h"

Visualizer* visualizer;

DWORD WINAPI UserInterfaceThread(LPVOID lpParam)
{
	BOOL bRet;
	MSG msg;

	visualizer = new Visualizer;
	visualizer-
>CreateEx(0,0,WS_VSCROLL|WS_HSCROLL|WS_MINIMIZEBOX|WS_SIZEBOX );
	visualizer->ShowWindow(SW_SHOW);
	visualizer->UpdateWindow();

	PeekMessage(&msg, NULL, WM_USER, WM_USER, PM_NOREMOVE);
	SetEvent((HANDLE)lpParam); // Signal that an event-queue and the
visualizer window has been created

	while( (bRet = GetMessage( &msg, 0, 0, 0 )) != 0)
	{
		if (bRet == -1)
		{
		}
		else
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return 0;
}

Visualizer::Visualizer(void) :
	GridLength(1000),
  timewidth(0),
  scrollH(0),
  scrollV(0)
{

}

Visualizer::~Visualizer(void)
{
}
```

7 References

C. Jacobsson, (2001-12-17) ABB Doc. No. 3BSE024658 *Functional Description: Program Execution*

Peter L. Nilsson, (2007-03-23) ABB Doc. No. 3BSE025583 *DoF: Controller Execution Model*

Peter L. Nilsson (2007-08-14) ABB Doc. No. 3BSE047273 *DoF: Execute Application in PM*

Luigi Palopoli, Luca Abeni, Marco Di Natale, Paolo Ancilotti, Retis Lab, Scuola Superiore, S. Anna, Fabio Conticelli (2001) *A tool for simulation and fast prototyping of embedded control systems*

Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, Karl-Erik Årzén, (2003) *How does control timing affect performance?*

Wind River Systems, Inc. (2007) *Wind River Workbench 3.0*, http://www.windriver.com/products/product-notes/Workbench-Tech-Note.pdf (available 9th of October 2008)