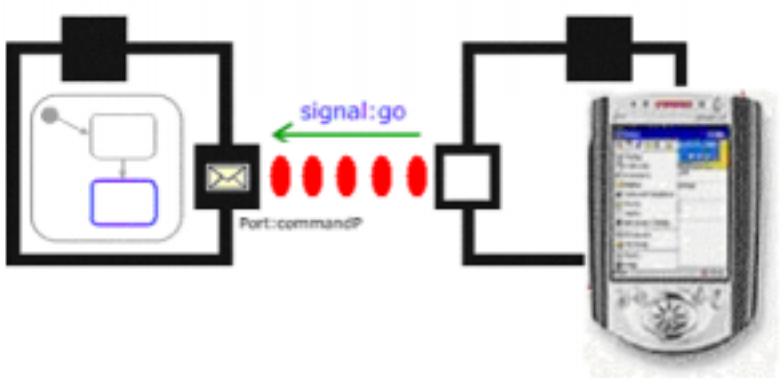
University of Technology, Sydney
Faculty of Information Technology

Masters of Science in Internetworking
32934 Project B2

Developing Infrared Distributed Applications on the Windows CE 3.0 Platform Using Rational Rose RealTime



Student Name: Paul McIntosh
Student Number: 9901 9248

Supervisor: Paul Kennedy

This project has been submitted as a requirement of the Masters of Science in Internetworking.

Acknowledgements

- Paul Kennedy, John Hughes, David Wilson and Ury Szewcow (University of Technology, Sydney) for assistance on this project from proposal to final submission.

- My Rational Software colleagues in Rose RealTime Development, Technical Support and the Field, that answered my questions and shared their Rose RealTime knowledge with me.

Preface

This thesis represents the final project in the Masters of Science in Internetworking at the University of Technology, Sydney, Australia. This degree was undertaken part-time and this project represents 6 months of preparation and 6 months execution, undertaken in lunch breaks, evenings and weekends.

This thesis demonstrates an integration exercise followed by applying the integration results to a practical application. This statement does not only refer to thesis subject (Rose RealTime) but also to the integration of the knowledge acquired over the entirety of the 3-year Masters course. In this project I was pleased to be able to combine all the knowledge gained from the course (list of subjects below) with my professional experience and apply it to a practical real-world problem. By integrating my knowledge of Rose RealTime, programming, networking and operating systems I was able to obtain the results contained in this report.

- Principles of Object Oriented Programming in C++
- Internet Programming
- Unix Administration
- Internetworking
- C for Systems Programmers
- Operating Systems for Internetworking
- Lans and Routing
- Distributed Software Programming
- Information Technology Research Methods

Abstract

Rational Rose RealTime (Rose RealTime) is a visual modelling software development application produced by Rational Software. Rose RealTime utilises Unified Modelling Language (UML) and real-time extensions to enable Software Engineers to design real-time embedded applications, generate code from the design, compile, debug and deploy, all from within the one tool.

The project undertaken was to extend Rose RealTime to enable development of distributed models which utilise infrared communications between Windows 2000 (PC) and Windows CE 3.0 (Pocket PC) targets. To achieve this, first Rose RealTime required configuration to target Windows CE 3.0 Pocket PC and then to “road test” that configuration in the development of an infrared application.

At the conclusion of this report it can be stated that Rose RealTime can successfully build an application for a Windows CE 3.0 ARM target. Rose RealTime can also provide all the development functionality given to other target processors, so the advantages Rose RealTime gives to those targets extends to the Windows CE 3.0 ARM. With the configuration described in this report a developer is able to design in UML Windows CE 3.0 real-time embedded applications, generate code from the design, compile, debug and deploy, all from within the one tool.

For infrared, Rose RealTime has been proven to allow development of infrared applications. Although, infrared applications can vary in physical needs (i.e. permanent connections or one off connections) component reuse is still possible with inheritance. Also, as different development projects are undertaken, a “library” of reusable infrared components will quickly form with components being reusable “as is”.

1 Introduction

Rational Rose RealTime (Rose RealTime) is a visual modelling software development application produced by Rational Software. Rose RealTime utilises Unified Modelling Language (UML) and real-time extensions to enable Software Engineers to design real-time embedded applications, generate code from the design, compile, debug and deploy, all from within the one tool.

Rose RealTime also has the ability, through the use of another Rational product called Connexis, to apply the same UML visual modelling development principles to distributed applications.

The project undertaken was to extend Rose RealTime to enable development of distributed models which utilise infrared communications between Windows 2000 (PC) and Windows CE 3.0 (Pocket PC) targets. To achieve this, first Rose RealTime required configuration to target Windows CE 3.0 Pocket PC and then to “road test” that configuration in the development of an infrared application.

In addition to the technical aspects of the project, the results had to be “user friendly” and up-to-date to allow them to be applied to “real world” software development projects. In other words, the ability to build a Pocket PC application using Rose RealTime is not the end goal. The end goal is that Rose RealTime can not only build an application, but there are advantages in doing so from a user’s point of view.

2 Technology Overview

To enable the development of a Windows CE Pocket PC application an integration exercise had to be undertaken. This section gives an overview of the components that required integration with Rose RealTime. The illustration below (Figure 1) gives the context of the components. The following sections give a brief overview of the components with references to extra information for further reading.

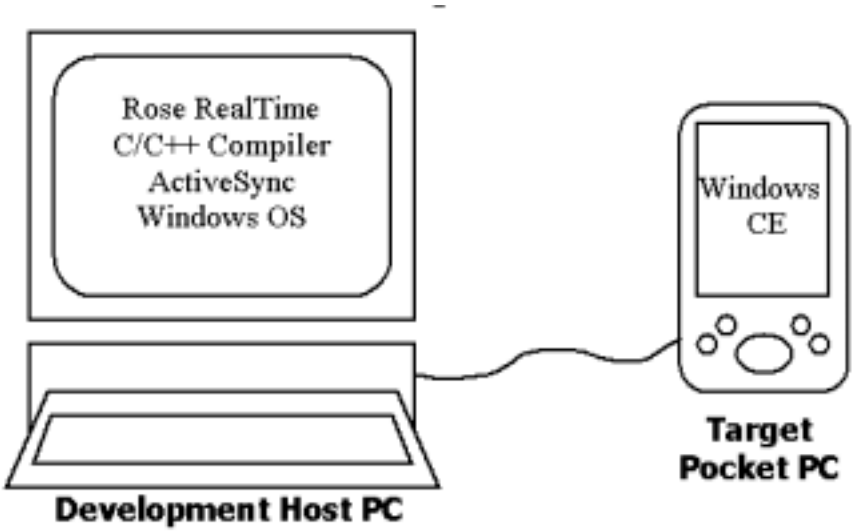


Figure 1 - Components of a Rose RealTime Windows CE development environment

2.1 *Rational Rose RealTime*

Rose RealTime (Figure 2) is a tool which allows you to build, debug, deploy a complete C/C++ application in UML. No coding is done outside the tool (as is the case with Rational Rose). To address the issues of real-time development and to enable complete code generation, real-time UML extensions are used to model an application. In Rose RealTime terms, the model “is” your application. “Model” and “Application” are used interchangeably throughout this report.

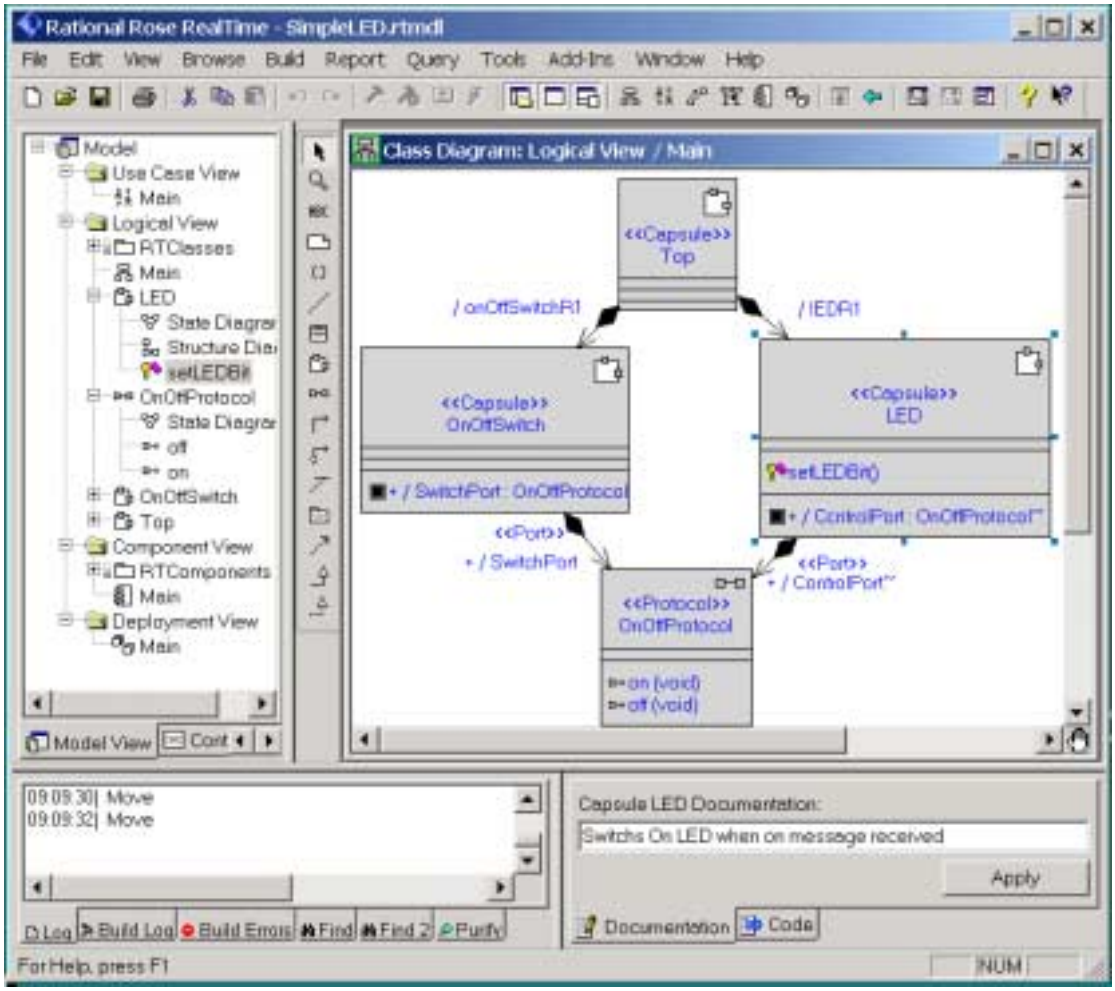


Figure 2 – Screen shot of Rational Rose RealTime v2001a

The basic UML building block of a Rose RealTime application is a “Capsule”, which communicates with other capsules via messages. A capsule is an Active Class, which has its own thread of control defined by a state machine. It also has an interface defined by a message port(s). Capsules can also have attributes and operations but these are not visible externally, all interaction externally is via the capsule port interface.

Capsule message communication is defined by ports (capsule interface) and protocols (in/out signals and message content). When a message is received at a port, the capsule state machine reacts to the message depending on how the state machine has been defined. Messages normally trigger a state machine to change state. The state change is called a transition and it is in these transitions that user code can be executed.

Using this approach a complete event driven application can be created as a system of connected communicating capsules. Figure 3 is an example of parts of a simple application which turns a LED (Light Emitting Diode) on and off with a switch.

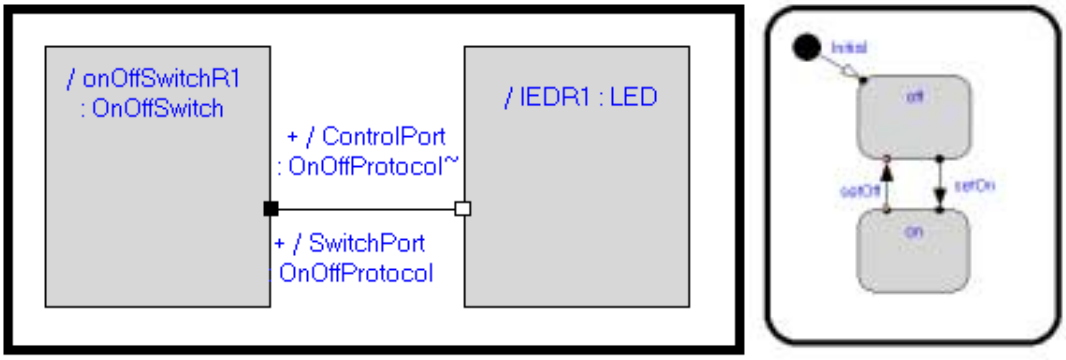


Figure 3 – Example of a "Structure Diagram" and a “State Diagram” that form part simple LED application modelled by Rose RealTime

The box outline on the left shows the “Structure Diagram” of the “Top” capsule (the main capsule in which the other capsules are contained). In this capsule we have an OnOffSwitch Capsule Role and a LED Capsule Role, which can communicate with each other via an OnOffProtocol through their port interfaces and the connector between them. A Capsule Role is an instance of a capsule, so the LED capsule can play different Roles depending on your system (here it’s named the default “IEDR1” but in a different system we might call it “warningLight” or “landingGearDown” depending on its use).

The “State Diagram” shown on the right is the behaviour defined for the LED capsule. This shows that an LED’s initial state is “off”, if it gets an “On” message, the state will change to “On”. The actual turning on of the LED could be done in the “setOn” transition (e.g. C++ code setting a hardware bit controlling the LED to 1).

Rose RealTime can generate code from these diagrams, which a third party compiler can then build into an executable. To provide the Run Time Services (e.g. messaging, controlling state transitions, creating capsules etc.) Rose RealTime has a "targetRTS", which is a pre-built library for a target, linked into the application. Rose RealTime can then run the built application on the target and debug the model visually (called Target Observability).

Rose RealTime, as an application itself, can run on Windows and Unix Host Development environments. For this project a Windows 2000 host was used as the Host Development environment.

More information can be found out about Rational Rose RealTime from Rational Softwares on-line documentation [WEB1][RTDOC3][RTDOC4]

2.2 *Connexis*

Connexis is an extension to Rose RealTime. Connexis uses the UML message mechanism to allow you to carry on modelling distributed applications in the same way. So in the above LED example, the two capsules talking to each other no longer need to be contained in another capsule, they can instead be run as separate processes on separate machines (e.g. OnOffSwitch capsule running on a PC in Australia and LED capsule running on a Pocket PC on Sweden).

Connexis takes care of the complicated bit (encoding, getting the messages there and decoding) and the application designer carries on defining protocols and messages etc., as was the case previously. Connexis comes out-of-the-box with TCP and UDP support but can be ported to other transport systems.

More information can be found out about Connexis at the Rational Website [WEB1]

2.3 *Windows CE 3.0*

Windows CE is Microsoft's embedded real-time version of Windows operating system. Windows CE provides similar functionality to Windows on the PC but for devices with limited memory and operating characteristics (e.g. low power, quick start-up). Windows CE supports a number of embedded processors and can be used for a variety of systems (not just handheld devices).

For Pocket PC devices there is an application called ActiveSync, which allows data to be synchronised between a desktop PC and the Pocket PC. ActiveSync resides on the desktop PC and manages the communication between the PC and handheld device.

Windows CE applications for Pocket PC's can be developed using eMbedded Visual Tools. This provides the ability to build Visual C++ and Visual Basic programs. Visual eMbedded Tools supports a number of processors including both the ARM and SH3 processors mentioned in this report.

More information can be found out about the Windows CE components one the following websites Windows CE [WEB6], ActiveSynce [WEB9] and Visual eMbedded Tools [WEB10].

2.4 *Compaq iPAQ Pocket PC*

The target hardware chosen was the Compaq iPAQ Pocket PC H3630. This uses a 206 MHz Intel StrongARM 32 bit RISC (SA 1110) Processor and comes with 32M of memory. The iPAQ uses Windows CE and provides an exceptional colour interface.

The target was chosen because it was the latest technology available at the outset of this project (released July 2000) and becoming very popular. The ability to interact with Windows 2000 and the similar target OS specially lent itself to Rose RealTime. With the possibility to build the same capsule for a PC or Pocket PC, the amount of memory available and the communication capability, meant that very complex and adaptable real-time cross platform applications were possible.

More information can be found out about the Compaq iPAQ Pocket PC at the following Compaq product website [WEB7].

2.5 *Infrared Communications*

As the research was for an Internetworking Masters project, I decided to make the example application generated use infrared. This would provide added value to the results, as I could find no information on using Rose RealTime for an infrared application.

Windows CE uses the IrDA (Infrared Data Association) standard for communication. This is implemented within the Windows CE WinSock library and from a programmer's point of view, is similar to creating TCP/IP socket communication.

More information can be found out about the IrDA standard at the Infrared Data Association website [WEB3]

3 Problem Description

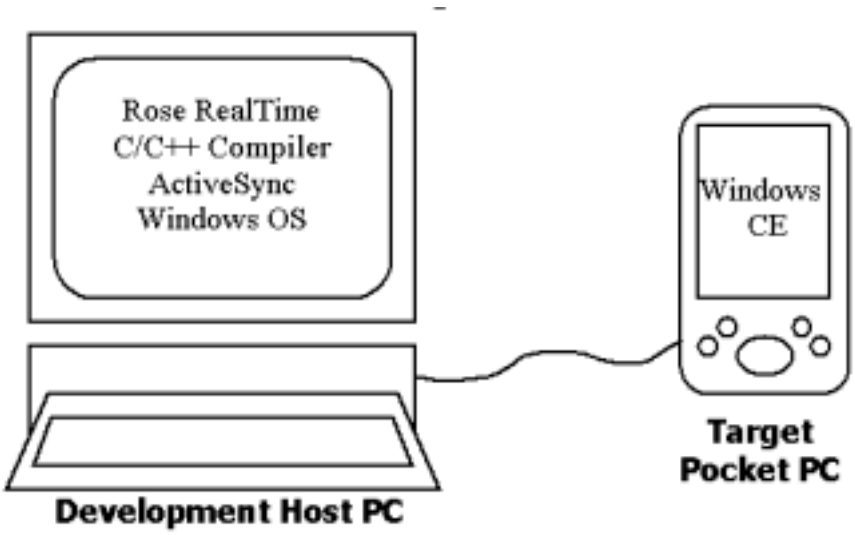


Figure 4 - Components of a Rose RealTime Windows CE development environment

As mentioned previously, the initial problem is one of integrating the various components. Referring to the overview diagram (Figure 4) and the information provided in the Technology Overview, this problem can now be expanded on.

The present development method is to write your application on the PC using the Visual eMbedded Tools IDE, build the code, load the target onto the Pocket PC and run the application with the use of ActiveSync services. ActiveSync, which resides on the host, does the communication to the target Pocket PC through a USB cable (in my initial configuration but it can be done by other means).

To integrate Rose RealTime into this environment the following needs to be achieved:

- Compiling the Run Time Services/Connexis library for Pocket PC – in Rose RealTime terms this is called “Porting” the TargetRTS and Connexis libraries.
- Enabling TCP/IP connections to the target – having this ability will enable Target Observability.
- Enabling control of the external Pocket PC application from within the Rose RealTime toolset – the Rose RealTime term for this is called “Target Control”.

As mentioned in the introduction, the ability to build a Pocket PC application using Rose RealTime is not the end goal. The end goal is that Rose RealTime can not only build an application, but there are advantages in doing so from a user’s point of view. With this in mind, once Rose RealTime has been configured to build Windows CE applications for the Pocket PC, the next step would be to put it through a development exercise.

The integration results need to be exercised by developing a simple application and then a similar more realistic complex application. In this report an Infrared application was chosen, as this had not been done before.

3.1 *Porting the TargetRTS and Connexis Libraries*

Rose RealTime has a library, which provides the Run Time Services. This provides things like communication services, debugging functionality, timing services, capsule control and coordination of state machines. Also provided with Rose RealTime is the Connexis library, this handles the communication services for distributed models.

The first step in the integration is to build these libraries with the Windows CE eMbedded C++ ARM compiler. The source code for these libraries is available in the Professional and DevelopmentStudio versions of Rose RealTime. These libraries are configurable and instructions on how to port the targetRTS and Connexis libraries to a new platform are given in the on-line help (Porting Guide [RTDOC1]).

Fortunately, Rose RealTime (v2001.04.0 and v2002.05.0) already supports Windows CE for the SH3 processor. This means that the integration exercise is to take the SH3 configuration and adapt it so that a new targetRTS can be compiled. With this a Rose RealTime model can be built into an application that runs on the Pocket PC.

3.2 *Enabling Target Observability*

Target Observability is the ability for Rose RealTime, running on the host, to control a running debug application on the target and observe the behaviour visually. This is achieved by Rose RealTime connecting to the running application via a TCP/IP socket connection. The Rose RealTime generated application is started on the target with an “obslisten” parameter, this sets a socket listening to a port ready for Rose RealTime to connect to it (e.g. myRoseRTApp.exe –obslisten=22222 sets a socket port of 22222).

Once connected Rose RealTime can start and stop the model, watch state changes, capture message traces between capsules, watch and change attributes, inject messages and other useful development activities.

The integration problem here is that a standard Pocket PC communication set-up does not allow direct TCP/IP connections. Communication is done via ActiveSync and no external TCP/IP connections are allowed. One solution is to buy an expansion pack with an Ethernet card. This gives the Pocket PC an Ethernet address and allows TCP/IP communications but at the expense of money and losing an expansion slot (which might be required for development of a application that uses that slot).

So the next integration exercise here was to see if Target Observability could be done without the need for extra hardware.

3.3 *Target Control*

Target Control is different to Target Observability. Target Control allows the developer to load the application on to the target from within Rose RealTime, start the application with the appropriate options and kill the application when they are done. This is done from within Rose RealTime using component instance menu items (i.e. the model element which represents the executable), which invoke Perl Target Control scripts.

So the final integration exercise is to create Target Control scripts, which would allow a developer to get a freshly created executable onto the Pocket PC, run the executable with the parameters (such as –obslisten) so it waits for Target Observability. Then when the developer has finished, tidy everything up by killing the application and going back to a state where everything is ready for the next version of the application to be tested.

3.4 *Keeping it Sensible*

The final project report should be something that is useable in a current real development environment ("current" being the time the final report is published not a baseline chosen when the research was started). The rate of change of the components (on a monthly basis) requires that the latest technology be absorbed as it is released during the life of the research project.

To ensure that the results were useable, the problem is addressed in two stages. The first stage is configuring Rose RealTime for Pocket PC development and the second is putting it through a development "road test".

For this project Action Research, normally associated with Social Sciences, was used. This allowed me to place myself as an active participant in the research as an end user and make subjective decisions as to the value of what was uncovered. The process meant that I could concentrate on usability of the tool in a real environment (upgrades, patches, conflicting installations, viruses and all) and adapt the research direction to fit the user, based my experience as a Software Engineer. I have attached experiences about using this methodology in Appendix A.

4 Configuring Rose RealTime for Pocket PC Development

Information on porting a target can be found in the Rose RealTime on-line documentation “Porting Guide”[RTDOC1]. The information provided describes porting in general and does not address specific unsupported targets. This report outlines the steps for porting specific to the ARM processor and Windows CE 3.0, and is intended to be useful for developers planning a similar target port.

4.1 *Testing the Tool Chain*

Before any target port is undertaken, the elements of the tool chain should be verified by testing the capability without Rose RealTime. In this case the first step is to compile a simple Pocket PC "Hello World" using the eMbedded Visual C++ 3.0 IDE. As the final application was to use infrared, an "Infrared Hello World" was compiled also.

The initial tool chain consisted of either a PC or Notebook (two development hosts were used for cross checking behaviour, convenience and infrared capability). The initial set-up had the Pocket PC in a USB cradle, ActiveSync running the default installation and eMbedded Tools the default installation.

This test was done by simply compiling and testing the examples give in “Windows CE 3.0 Application Programming” [CEDOC1]. This provided proof that the initial development environment was functional and also provided a broad range of code examples, which were proven to work (and were accompanied by good explanations of how they worked).

4.2 *Porting the TargetRTS and Connexis*

Knowing that normal code compilation was possible, the next step of compiling of the targetRTS was started. Before the step was completed though, Stephen Rooks (Rational Software [COM1]) announced successfully porting the libraries to the Pocket PC. With his configuration available, concentration turned to creating a repeatable process for doing this and documenting it for use by others. Later this was also done for the Connexis libraries.

As a result of this investigation I published two Rational documents. The documented processes give step-by-step instructions on how configure existing Rose RealTime targets to enable the Pocket PC to be targeted.

- Porting RoseRT C++ TargetRTS to Windows CE 3.0 - ARM processor [TN18596]
- Porting Connexis Libraries to Windows CE 3.0 - ARM processor [TN21927]

The documents are attached (Appendix B and C) and have been made freely available to Rational customer's as Technical Notes on the Rational web site (http://www.rational.com/support/technotes/).

The porting was a fairly simple process due to having the Windows CE SH3 port to base things on. However, there were a few points discovered on the way to the documented process that should be considered when undertaking a similar port.

- **Windows and Windows CE path conflicts**. This is important to watch out for. As Windows and Windows CE are similar, it is very easy to confuse a compiler/linker. The porting scripts set up the environment for you with the correct include and lib paths, but when it comes to building in the real world Rose RealTime is relying on the system set paths (i.e. the environment variables "include" and "lib") or what you explicitly set in the Rose RealTime component specification (a component represents the final build artefact). It is very easy to pick up the wrong file (e.g. winbase.h exists for both platforms) or the wrong library (e.g. there is a commctrl.LIB for each machine type).

- **eMbedded Visual Tools is an IDE for different compilers**. Some cross compilers are implemented using a "wrapper" program that then calls the specific compiler based on a target option. For Windows CE the individual compilers are explicitly called and these also have different options. So when doing a port from one Windows CE target to another, the engineer should be aware of the idiosyncrasies of each compiler.

- **Beware of options that aren't required**. Since the easiest way to port a target is to base it on another, obsolete options can be introduced. As mentioned above, compiler options will vary between targets, my port initially tripped up on a "/MD" compiler option. When investigated more closely, it was found that the "/MD" option was not required in the SH3 port and was obviously a leftover from a previous port, there because it didn't have adverse effect.

After working through the above issues, a “Hello World” model could be built and run on the target Pocket PC. At this stage the HiCapsule.exe had to be copied manually to the Pocket PC and started using the RTS debug command line via keyboard entry (Figure 5).

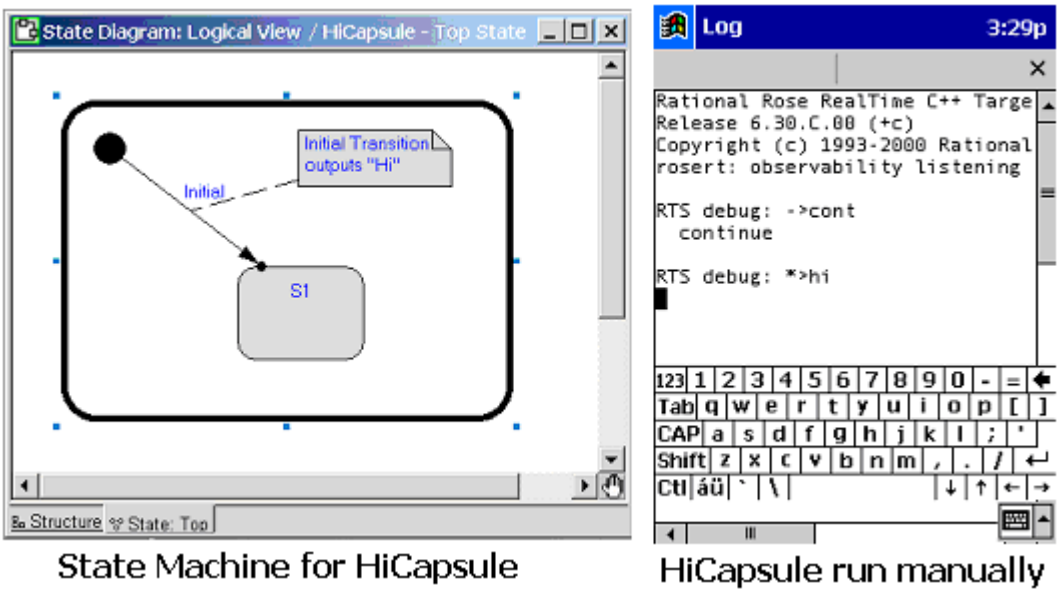


Figure 5 - HiCapsule state machine and screen capture of application running on the Pocket PC

4.3 *Enabling Target Observability*

Now that a Pocket PC application could successfully be developed from Rose RealTime, the next step was to be able to visually debug it and this meant getting a TCP/IP connection. Others [COM2] had been successful in this by using Ethernet cards but it was decided to see if it could be done without extra hardware.

References to TCP/IP connections in Grattan and Brain [CEDOC1] stated “... sockets cannot be used to communicate between a Windows CE device connected to a desktop PC” then later it indicated a workaround “You can configure ActiveSync to use RAS and hence provide TCP/IP support”.

Grattan and Brain [CEDOC1] indicated where to find information on RAS (Remote Access Services) but the information found was not usable. Enquires on microsoft.public.pocketpc.developer newsgroup confirmed with anecdotal evidence that TCP/IP was indeed possible over USB and had been done before on the iPAQ using RAS.

After trial and error I managed to get a TCP/IP connection working over USB. However as my environment changed (ActiveSync upgrade and Windows 2000 patches applied) I lost this ability and could not get it back because I had failed to understand how the USB driver had registered a COM port (the COM port being required for defining the RAS connection).

With the aid of “The Windows CE Technology Tutorial” [CEDOC4] a repeatable process was achieved that would enable TCP/IP connection via a serial cable. This worked with Rose RealTime Target Observability and initial operation showed that it was good enough for development purposes.

The way it works, is RAS enables the Host PC to provide an IP address to the Client in a similar fashion to dialling into an ISP network. This means that the Pocket PC connects to the Host via a serial cable and has an IP address assigned to it. ActiveSync can be configured to use the RAS connection instead of a direct serial connection, so ActiveSync services remain intact.

In the course of this integration exercise, a need to start applications with all the necessary command line options was found. Doing this from the Pocket PC interface was difficult but a solution for this was found using “links”, which are similar to the Windows desktop shortcuts. With this and the RAS information I created the following process documents (Appendix D and E):

- Target Observability to a Windows CE device using Serial Cable [TN20458]
- Using links to start Windows CE applications with Rose RT command options [TN19967]

The documents describe step-by-step how to achieve target observability and how to easily launch applications with the correct parameters. The points to note in addition to information contained in those documents are related to research process:

- **Investigate things that work, as you would things that don’t work**. When a problem is hit during an integration exercise, lots of knowledge is gained because things can’t progress until the situation is understood and fixed. However, if something works then the need for in-depth investigation is not as vital and can easily be overlooked. What I experienced with the USB, was that it worked, I didn’t understand how a COM port appeared for a USB port but I wasn’t blocked so I ignored it. When it broke, I couldn’t fix it because I didn’t understand it fully and valuable information was lost because I did not gain the knowledge when the opportunity was there. This is nothing new to research, but with Windows CE, it is particularly important because documentation can be non-existent (in this case the details of the Host USB driver WCEUSBSH.SYS).

- **Old information is still useful**. Even though this area of computing is changing at a rapid rate, important information for a current OS release can be found in literature written for the previous OS release. In this case a Windows CE 2.0 book [CEDOC4] provided the solution for a Windows CE 3.0 problem that could not be found in the 3.0 literature [CEDOC1].

4.4 *Target Control*

The next step in the exercise was to enable a developer to control everything from within Rose RealTime. The advantage of using the RAS solution for Target Observability is that all the ActiveSync services are still intact. So these services can be used for controlling the executable on the Pocket PC.

Before completing this section, I discovered that Windows CE Target Control was being introduced into the upcoming Rose RealTime release [COM3]. A pre-release version of the Target Control scripts and executable code was obtained. Although I had most of the functionality complete, I opted for the Rational developed code because this is what would eventually be used in a real development environment. Providing feedback and additional testing being more important than duplicating effort with my own solution.

Although, I did not use my solution for the final development test, this area of Rose RealTime has no public documentation. So documenting my experience in the report will provide valuable information for anyone needing to create his or her own control scripts.

Note: Target Control scripts are located in the $ROSERT_HOME\bin\tc\win32 directory. This directory needs to be selected in the Processor Specification (Figure 6 - the model element which represents the target processor). They are invoked by selecting the item by right-clicking on the Component Instance (the model element which represents the executable on the target).

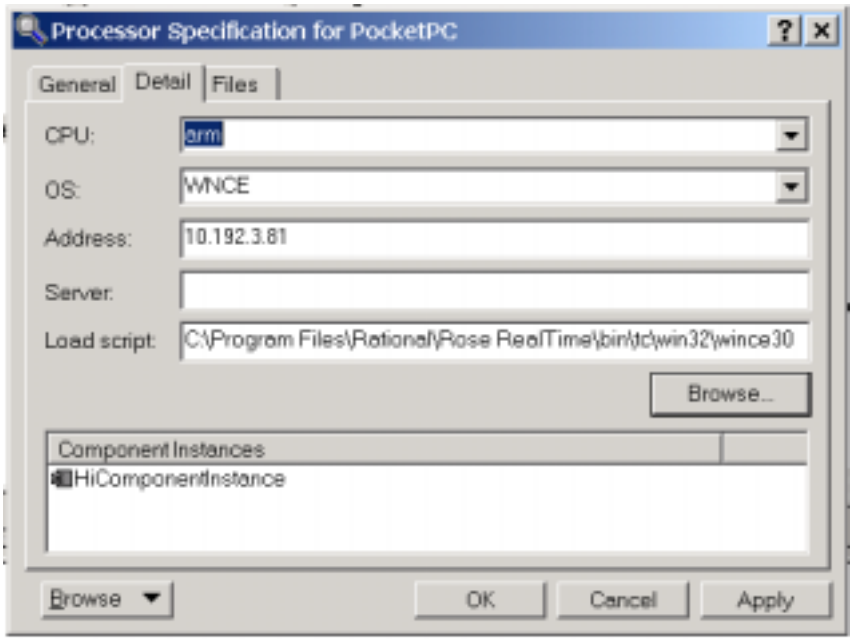


Figure 6 - Processor Specification showing settings for a Pocket PC target. "Load script" is set to the new wince30 Target Control script location.

To implement the target control, a new directory was created called wince30 and, placed in that directory, empty *.pl files which were to become the new perl control scripts.

load.pl – move the newly built application to it's location on the target
reset.pl – reset the target
terminate.pl – kill the application on the target
execute.pl – run the application on the target
unload.pl – remove built application from the target

I had access to some internal Rational documentation, which was very possibly outdated. Assuming the worst, the following method was found that could be applied to gain knowledge of the internal script/Rose RealTime operation without the need for any design documentation.

- Make the script output the values that it was invoked with. Having load.pl simply print the contents of the argument list, will mean the list will be printed in an error box (because Rose RealTime needs a "::Ok::" response to go ahead to the next step). Using this, it was easy to see what arguments were available to work with e.g. "Script (load.pl) -exe F:\shared\HiComponent\build\HiCapsule.EXE -port 30819 -ip 10.192.3.81 -os WNCE -cpu arm"
- To find out what Rose RealTime expected in response, the opposite method was used. Using Target Control scripts of an already implemented platform and starting them outside of Rose RealTime with the correct arguments to observe the response. It was found that "::Ok::" was printed when successful, and that Rose RealTime should recognise this as the success return value.

By using this method, the behaviour of each control script could be uncovered and then programming to achieve each step (i.e. load, execute, etc) could be done, based on the parameters and return codes expected.

I managed to "load" the executable file from the Rose RealTime build area to the target Pocket PC, using a simple dos copy command in the perl script. This simply copied the executable file to an ActiveSync directory, which was then automatically synchronised with the Pocket PC.

Execute.pl was the next step. This could not be done via a pure perl script and a Target Control application had to be created. To achieve this, the "prun" (run a process remotely from the PC) example code from the Windows CE Tools directory [CEDOC2] was adapted it run an *.exe with command line options and print "::Ok::" when it finished.

At this point I trying to work out how to terminate the application while running, when I found out about the "official" Target Control" scripts. The only solution that I could think of, and it was the one the Development came up with also, was to load a "kill" program. There are services to run an application remotely but none to stop one remotely. Ideally the method used should be the same as used from the Pocket PC, but there is no documentation on how to invoke QUtilities remotely.

To summarise the findings in this section, Target Control scripts can be created by:

- Capturing the parameters that Rose RealTime passes to a script.
- Use existing scripts and known parameters to find out what Rose RealTime expects in response to a successful (or unsuccessful) action.
- Implementing the actions may require a variable number of things to happen. It could be simply done entirely in a perl script or may require a couple of specialised applications to achieve the end goal.

4.5 *Configuration Conclusion*

At the end of the integration exercise, Rose RealTime has been successfully configured for developing an application for the Pocket PC. The final development environment consists of a mixture of old and new technology:

- TargetRTS and Connexis ported by a documented process
- Target Observability using RAS to communicate over a 115K2 serial link.
- Pre-release Target Control scripts created by Rational Software.

These all worked on a simple model and in theory should be suitable for developing more complex applications. The test model so far consisted of a single capsule transitioned between two states every second. In the transitions it logged to the screen " State1 <----- State2 " or " State1 -----> State2 " depending on the transition (Figure 7).

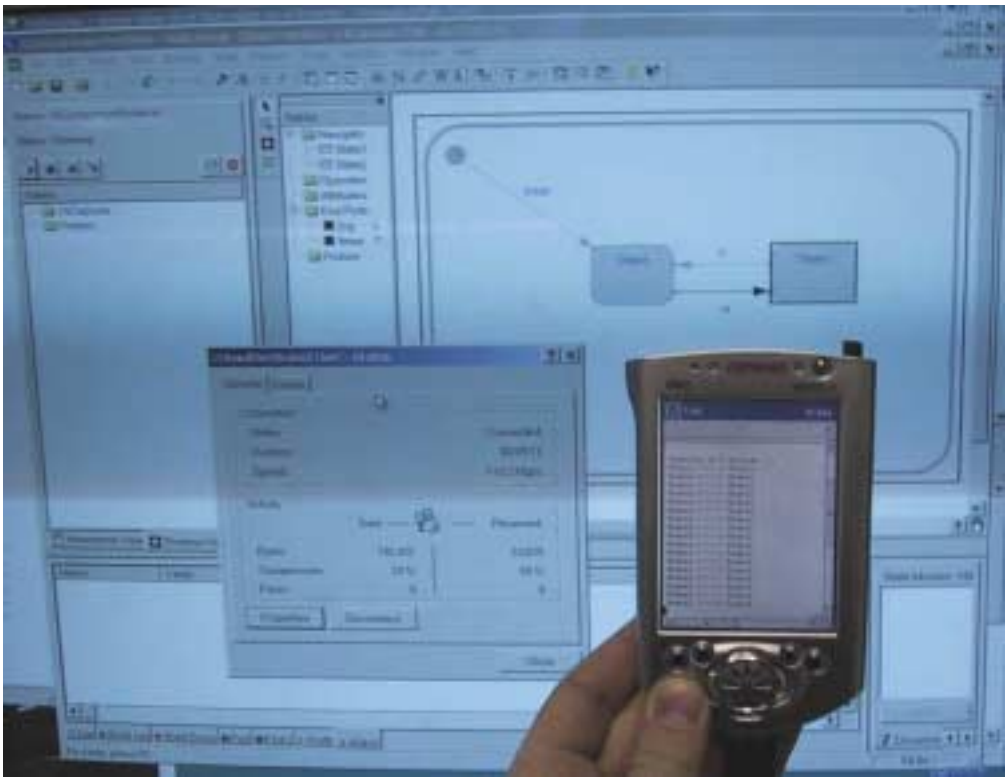


Figure 7 - The final working development environment. This shows the application running on the Pocket PC with Rose RealTime displaying the behaviour of the state machine. The RAS connection box is displayed to show the connection speed of 115.2 Kbps (this connection is normally indicated only as an icon on the Windows Task Bar).

The model used was trivial, the next stage was to test out the environment on a more complex infrared applications.

5 Development Road Test

Now that development was possible, the next phase was to test development on more complex models. The plan here was to investigate a few things that would provide even more functionality to what was there already. Then assess what was available and run through a “Development Road Test” of a simple HelloWorld distributed infrared application. After the road test the next phase would be to apply what was learnt to a more realistic application.

Note: Time was one of the constraints on this project so complete investigation into all areas was not possible.

5.1 *Connexis*

The first thing investigated was Connexis and the possibility of using it for infrared. If Connexis could be ported to use infrared, then that would make the modelling a lot easier.

The Connexis libraries were ported first, tested successfully and the process documented [TN21927] (Appendix C). This enabled development of distributed models that make use of TCP/IP or UDP/IP. This was tested using the “PingPong” and “BasicTest” example models provided with the Rose RealTime install [RTDOC1].

Investigation into porting infrared to Connexis showed it was a lot harder than was initially thought. I imagined that the Connexis library would make its own TCP/IP calls and to port it, since IrDA sockets are similar in structure, it would just be a matter of changing the parameters to suit IrDA sockets. However, it was discovered that Connexis uses the targetRTS TCP/IP calls (i.e. wrapped socket calls) and changing those would impact all TCP/IP services (e.g. Target Observability) not just Connexis.

Conclusion: At the end of this investigation TCP or UDP communication was possible through Connexis but an infrared Connexis version would require more time than was available. Infrared would have to be implemented at capsule level.

Note: At the infrared development stage, I found myself spending most of my effort encode/decoding code messages and implementing buffer protocols for sending messages across the infrared link. This is all standard for socket communication but in hindsight, a Connexis infrared port may have negated the need for this and warranted more investigation here.

5.2 *MFC*

The next issue to address was the possibility of a GUI (Graphic User Interface). Windows 2000 and Windows CE can both use MFC classes. The actual classes are different but they are similar enough to be easy to adapt from one platform to another.

Rational Rose (a more general UML tool) comes with a complete UML MFC model. Rose RealTime can import Rose models, so the idea here was to import the Rose MFC model to see if it could be used to generate a complete Windows application within Rose RealTime.

Although, not having experience with MFC, I achieved some limited success in generating a Windows application. However, simply generating an application is only half the battle. Both Windows and Rose RealTime's targetRTS want to be in control of what happens in an application (Windows for processing windows events and the targetRTS imposing predictable run-time behaviour). Getting these to work together is not impossible but again required more time than what was available.

The model shown (Figure 8) displays a Window if linked with the option /subsystem:windows, as "WinMain" is the entry point of the application. The capsule behaviour works if linked with /subsystem:console, as "main" is the entry point (the entry point for the targetRTS).

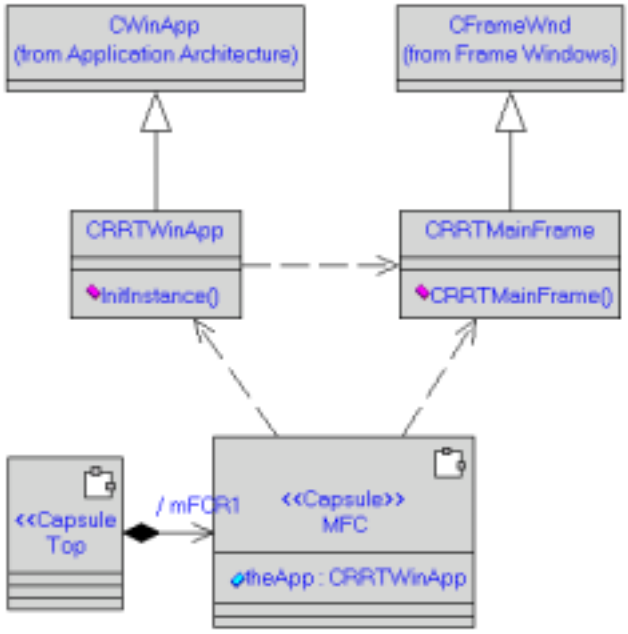


Figure 8 - UML Class Diagram of a simple MFC Hello World application.

It is possible to start the targetRTS from a thread and therefore have both a Windows and the targetRTS running together. However, this was not pursued mainly because my lack of MFC understanding.

To enable Rose RealTime to create the MFC model above here are the steps required:

- Import the Rose MFC Package (Rose 2001 is required).
- For each MFC Class Specification change the Language from VC++ to C++ (VC++ indicates Visual C++ and this isn't valid for code generation) and change the C++ TargetRTS GenerateDescriptor to "False" (this will stop an attempt to generate code to describe the MFC classes to the targetRTS).
- In the Component View, create an external library component which points to the MFC library so that a dependency to this can be created (C:\Program Files\Microsoft Visual Studio\VC98\MFC\Lib\mfcs42.lib)
- Rose RealTime will now be able to generated code, however it will create the derived class with a header file of the base. For example the generated "CRRTWinApp.h" will #include "CWinApp.h" but the definitions for the CWinApp class are actually defined in "afxwin.h". The workaround for this is to create CWinApp.h with one line, "#include <afxwin.h>", and place it in the MFC include path.

To then model an MFC application:

- Create a Class Diagram and place the MFC classes on it.
- Derive the MFC Classes and draw the dependencies as shown in the diagram.
- Set the GenerateDescriptor to "False" for the classes.
- The MFC capsule will #include afx.h and have an attribute of your derived CWinApp Class ("theApp" shown in the Figure 8).
- The CRRTWinApp::InitInstance() operation contains the following code:

```
m_pMainWnd = new CRRTMainFrame();
m_pMainWnd->ShowWindow(m_nCmdShow);
m_pMainWnd->UpdateWindow();
return TRUE;
```

- The CRRTMainFrame() operation contains the following code:

```
Create(NULL,_T("Hello World!"),
    WS_OVERLAPPEDWINDOW,rectDefault);
```

To finally build the MFC application:

- For an application that starts with WinMain(), your Component Specification should have LinkArguments set to "/subsystem:windows" and the CompilationArguments set to "/D "_AFXDLL"".
- For an application that starts with main(), our Component Specification should have LinkArguments set to "/subsystem:windows /entry:mainCRTStartup" and the CompilationArguments set to "/D "_AFXDLL"".
- Draw a dependency to the MFC external library component from your application component.

Conclusion: The result of this investigation showed that Rose RealTime could be used to generate complete MFC applications. However, only the surface was scratched and there needs to be much more investigation done in this area before it could be put to practical use. Also whether or not it would be "user friendly" is also an issue, Rose RealTime is not a GUI development tool. This, along with the issues of application control (Windows event handler Vs TargetRTS) mean that practically, GUI development might best be done external to Rose RealTime and integrated as an external library (as is the present case).

5.3 *Infrared*

At this point it was time to undertake a test run a developing a simple distributed infrared application. The need for the word "distributed" is necessary because it is possible to build an infrared application which has only one component. Grattan and Brain [CEDOC1] give an example that simply looks for infrared ports and enumerates them. For a distributed application, there needs to be a number of dislocated processes collaborating.

The simple distributed example used, sent data back and forth between capsules and manipulated the data in the process. The infrared code was based on the Visual eMbedded examples ("Infrared Sockets Client" and "Infrared Sockets Server" [CEDOC2]) An Iterative development process was used, the predicted outcomes of each development iteration were:

1. **PC executable** (Top capsule containing a PC capsule and a Pocket PC capsule, exchanging data)
2. **Separate PC executable and Pocket PC executable** (PC capsule and Pocket PC capsule built for their specific platform, not exchanging data)
3. **Infrared Client/Server Capsules with irServer Capsule functional** (the irServer capsule could be coded and tested against the client example code compiled outside of Rose RealTime)
4. **Functional irClient Capsule** (the irClient capsule could be coded and tested against the server example code compiled outside of Rose RealTime)
5. **Adapt functional IR Capsules to work correctly with targetRTS** (Anything can be coded in the initial transition but, to implement properly, transitions should be run to completion or the model designed to handle blocking of threads)
6. **Separate PC executable and Pocket PC executable communicating via Infrared** (By combining the irClient and irServer capsules with the original PC and Pocket PC capsules, to create a complete distributed application, running as it did in iteration 1 but across IR).

The initial "Road Test" was successful, however, it did not follow the predicted iterations (as expected, hence the need for an iterative process). The solutions to the problems found will be used in the Revised Development section but can be summarised below.

- **Blocking Sockets** – To handle the blocking of sockets the IR Capsules were created (incarnated) on their own thread, this allowed the IR Capsule threads to block without impacting the rest of the model.

- **Undocumented features** – Undocumented features (not bugs) have been added to the implementation Infrared Socket programming. The following options _WIN32_WINNT | _WIN32_WINDOWS | _WIN32_WCE must be defined so the correct settings a selected in the included infrared header file (af_irda.h). These are not documented anywhere but the compiler error is informative enough to solve the problem. The solution is to place the applicable definition in the HeaderPreface of the capsules that create the infrared socket connections.

- **Encode/Decode** – To communicate across IR, the data needs to be encoded and decoded at either end. To aid in coding this within a capsule, the capsules should use a protocol that can be easily encoded/decoded straight from a message signal. My original protocol for between PC and Pocket PC capsules used an integer signal. I found that if I redesigned everything around a char buffer signal, then the messages could easily be converted for IR transmission.

- **IR Physical Layer idiosyncrasies** – IR is not the same as TCP/IP although the socket creation is similar. The Rose RealTime documentation [RTDOC1] gives a “SocketInterface” example of how to use an RTCustomerController for handling blocking sockets, however this was not usable for IR. The IR physical layer, in the case of a handheld device, is only available in response to a user need and is physically put in place (by a user’s action) for the duration of that need. The SocketInterface example was designed with the idea that the physical layer was always in place and the connections made at the need of the application. Thus IR Capsule had to be designed to account for lack of physical layer first as a “likely” rather than error state to recover from.

- **Application awareness of data transfer logic** – Further to the absence of a permanent physical layer the communicating capsules need awareness of the state of the communication. The IR Capsules have to be executing the correct code to communicate (i.e. if the IR Client needs to send something, the IR Server needs to be trying to receive). The solution to this was to have the capsules follow a send/receive sequence. The capsule connected to the IR Server starts by setting the IR Server capsule ready to receive and the capsule connected to the IR Client sets it to send. So the actual talking capsules maintain a state of send or receive, which is the opposite state to the other capsule (Figure 9).

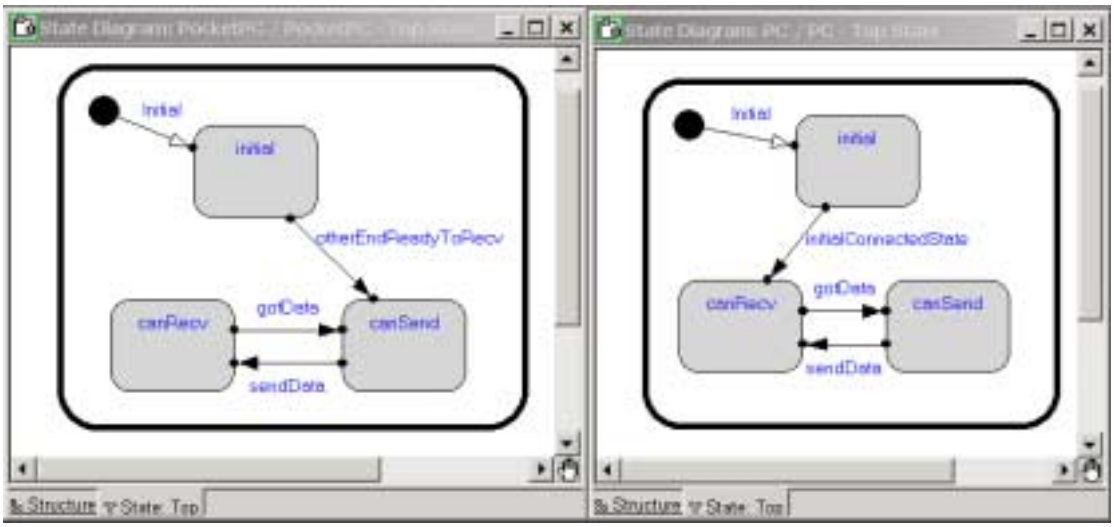


Figure 9 - Left is the Pocket PC Capsule state machine and right is the PC state machine. These state machines define a logical sequence for data communication. The PC (right) starts listening for a connection, when it detects a connection it expects to receive data (canRecv state) and when it gets data it expects to be able to the send data next (canSend state). The Pocket PC (left) does the opposite, it starts expecting to send data once a connection is detected. With this mechanism each capsule can share the communication media and only attempt to send data when the other end is ready for it (Note: This is only one solution and other solutions may be better).

The following is an overview of the final example model created. UML (Unified Modelling Language) makes it possible to describe vast amounts information about a software system using visual notation. Since this application was designed and built visually using UML, the design is given best described in a series of figures below.

Since Windows and Windows CE are so similar it was possible to model the entire application under Windows on the PC and once complete capsules could be rebuilt for the target platforms with minor modifications. Using the revised char buffer protocol, the non-infrared application modelled as a single process “Top” with a Pocket PC and PC capsule communication looked like the following Class Diagram (Figure 10).

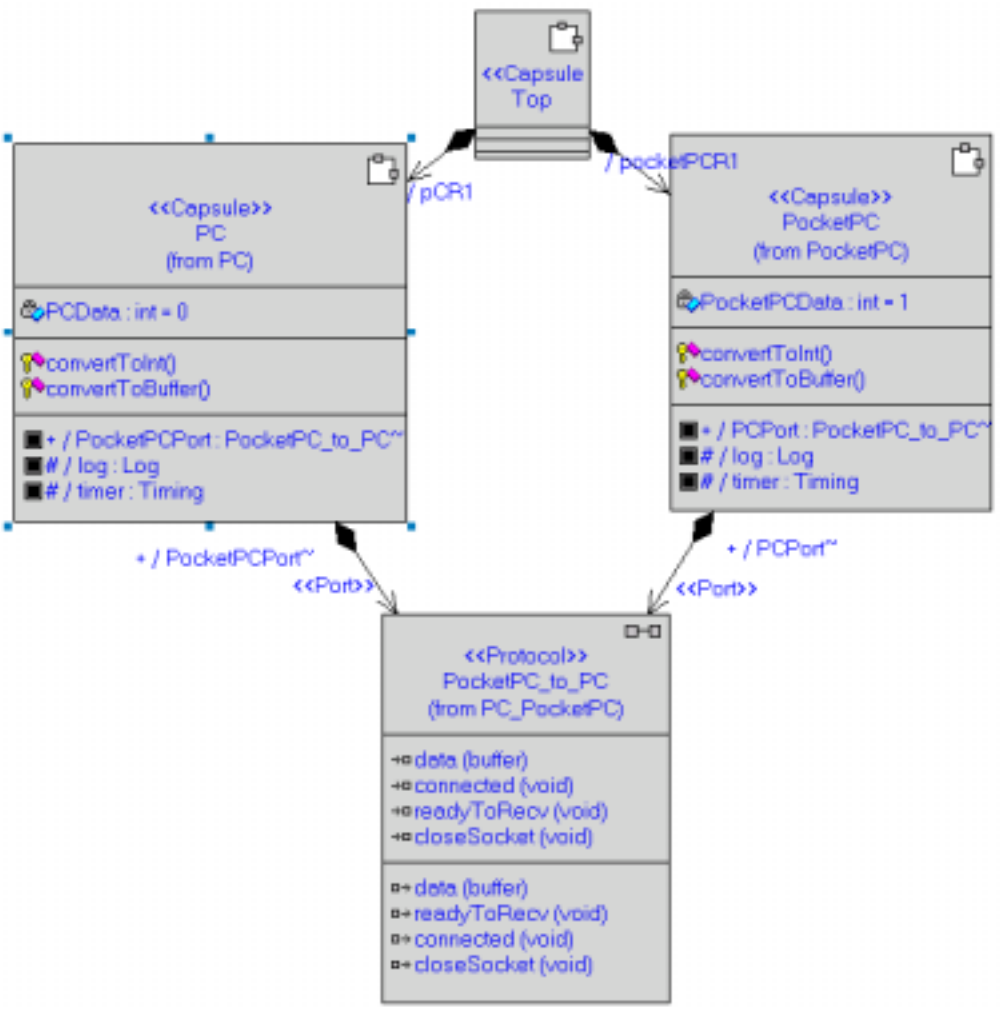


Figure 10 – Class Diagram of the Simple model without infrared. This shows a Top capsule containing a PC and Pocket PC capsule. The communication is done between the two contained capsules via the PocketPC_to_PC protocol. This protocol allows the capsules to indicate the communication state (whether they want to send data or are ready to receive data) and data is sent in a buffer. Using this protocol the capsules can be modelled to talk with each other directly. Later communication can be done via IR capsules by simply connecting the ports to IR capsules. So from the capsules point of view operation is the same whether directly connected or connected to an IR proxy.

Using the IR Capsules, which will be described later, the above design (Figure 10) could be easily adapted to run across IR. To demonstrate this structurally, Figure 11 shows the same single process model (as Figure 10) using the Structure Diagram of the Top Capsule. Figure 12 shows the same Capsules relocated to target specific Top capsules now communicating across IR via IR Capsules.

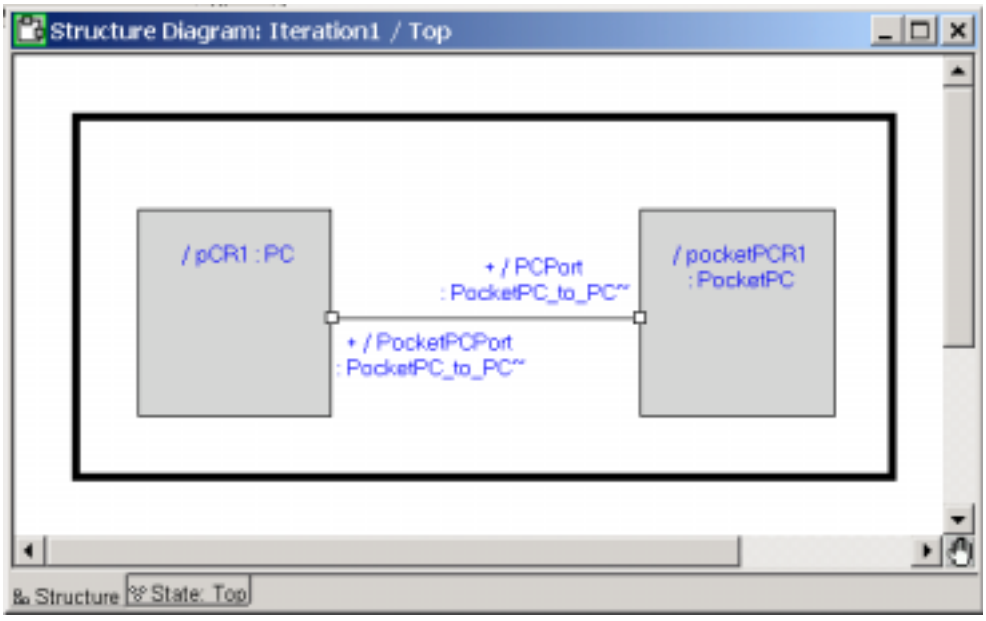


Figure 11 – The shows the same Simple model from Figure 10 but this time showing the Structure Diagram of the Top capsule. This shows a Top capsule containing a PC and Pocket PC capsule. The communication is done between the two contained capsules via the PocketPC_to_PC protocol through the PCPort and PocketPCPort.

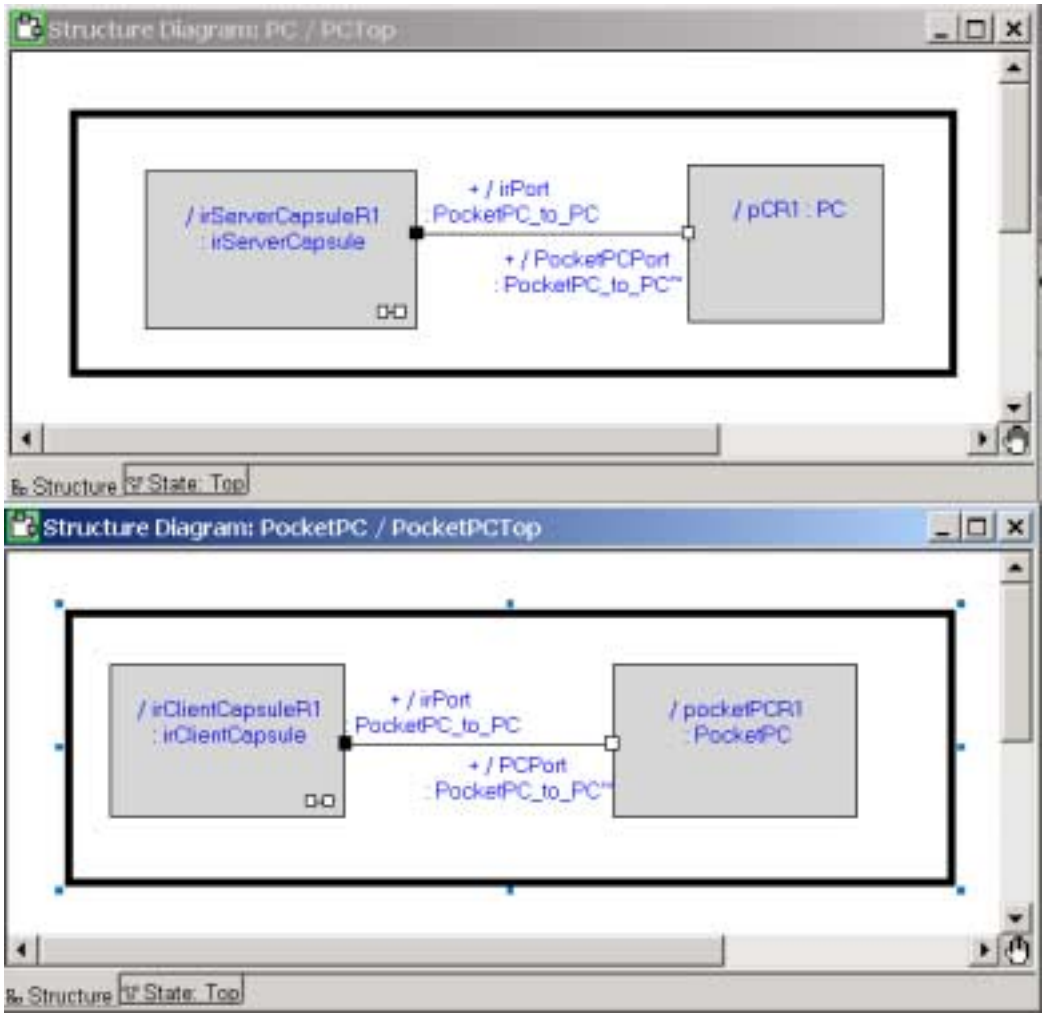


Figure 12 - Simple model now talking over IR, there are now two processes (i.e. running executables) on built for Windows CE (PocketPCTop) and one for the Windows (PCTop). Note that as far as the PocketPC and PC capsules are concerned, the communication is the same via the PCPort or PocketPCPort (depending on the capsule) which still uses the PocketPC_to_PC protocol. All the communication is handled by the irClientCapsule and irServerCapsule.

The IR Capsules are shown in Figure 12. This shows in UML notation that irClientCapsule contains an irClient, and irServerCapsule contains an irServer. The reason for this is that capsules can only be created on a separate thread at run-time. So the irClientCapsule and irServerCapsule simply create, encapsulate and pass messages to the irClient and irServer capsules at run-time. This is required because the irClient and irServer capsules can block (e.g waiting to receive data).

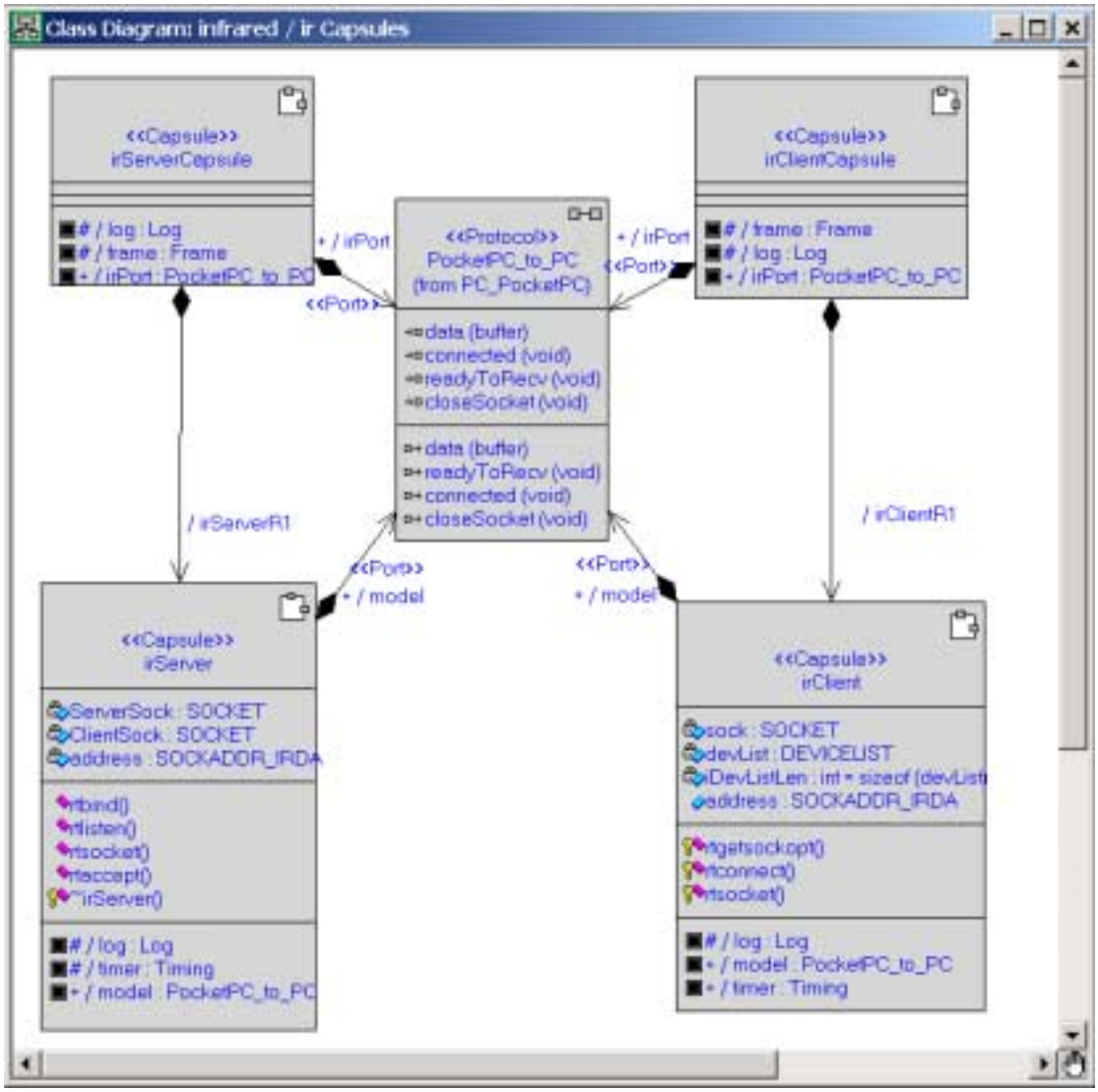


Figure 13 - Class Diagram showing the IR Capsules.

In Figure 14, the behaviour of the irServer capsule can be seen. The State Machines are similar for both Client and Server. The capsule waits in a “ready” state till it receives a message, it acts on that message and then returns to the “ready” state. An example of the effectiveness of using the buffer protocol can be show in an extract from the “recv” transition, when the IR capsule receives a message to receive something (readyToRecv), the following code is all that is needed (error recovery code not included).

```
// create a buffer of the type used in the protocol
buffer recv_buffer;

// fill the buffer with the data from the IR socket
recv (ClientSock, recv_buffer.charbuffer, sizeof
(recv_buffer.charbuffer), 0);

// send the data to the model via the “model” port
model.data(recv_buffer).send();
```

The “send” code is even simpler. When the capsule receives a “data’ message, the “send” transition executes the following (error recovery code not included).

```
// point to the contents of the message just received
const buffer & messageData = *rtdata;

// Send the data across the IR socket
send (ClientSock, messageData.charbuffer, strlen
                      (messageData.charbuffer) + 1, 0)
```

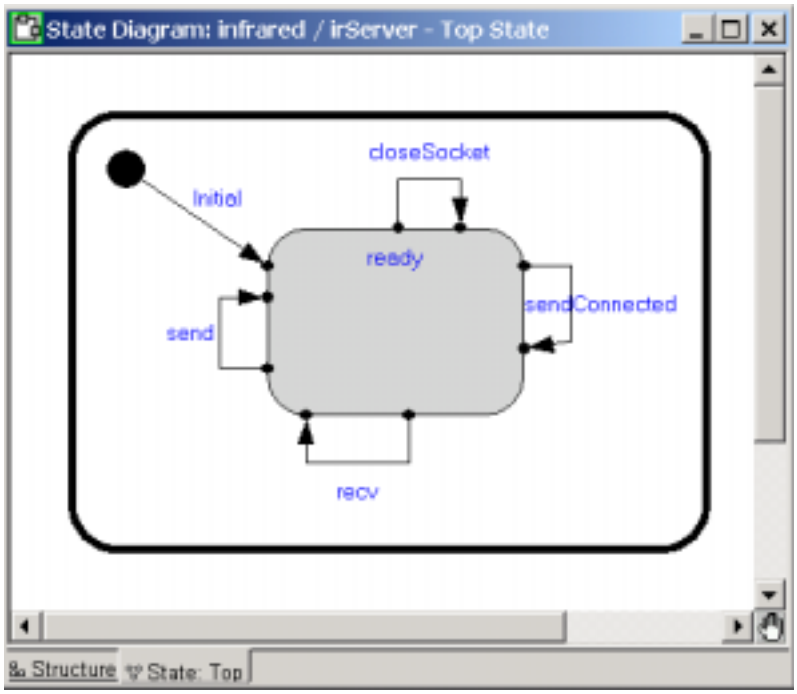


Figure 14 - State Machine of the irServer. “Initial” (in the case of the Server) executes code to listen for connections. When a connection is made, it sends a “connected” message to tell the outside world that its is ready to do things. It will then execute IR send and recv transition code based on the messages that it receives. “send” converts the protocol message data to IR and “recv” converts the IR data to a protocol message.

One of the advantages of Rose RealTime is that it is possible to visually observe, using Target Observability, how the application runs on the target. Rose RealTime also gives the ability to trace the behaviour. This was found to be extremely useful especially in this cross platform situation where the processor speeds were very different. In some cases, delays using “timeouts” (a timing service provided by the targetRTS) needed to be used to make sure the socket connection was ready to use before data was sent. The trace ability also provides a useful way of demonstrating the actual behaviour in this report. The figures below show, an overview of all the capsules concerned (Figure 15), the Sequence Diagram of the PC application (Figure 16) and the PocketPC application (Figure 17), and finally the last pictures in this section show it looked in reality (Figure 18 & 19).

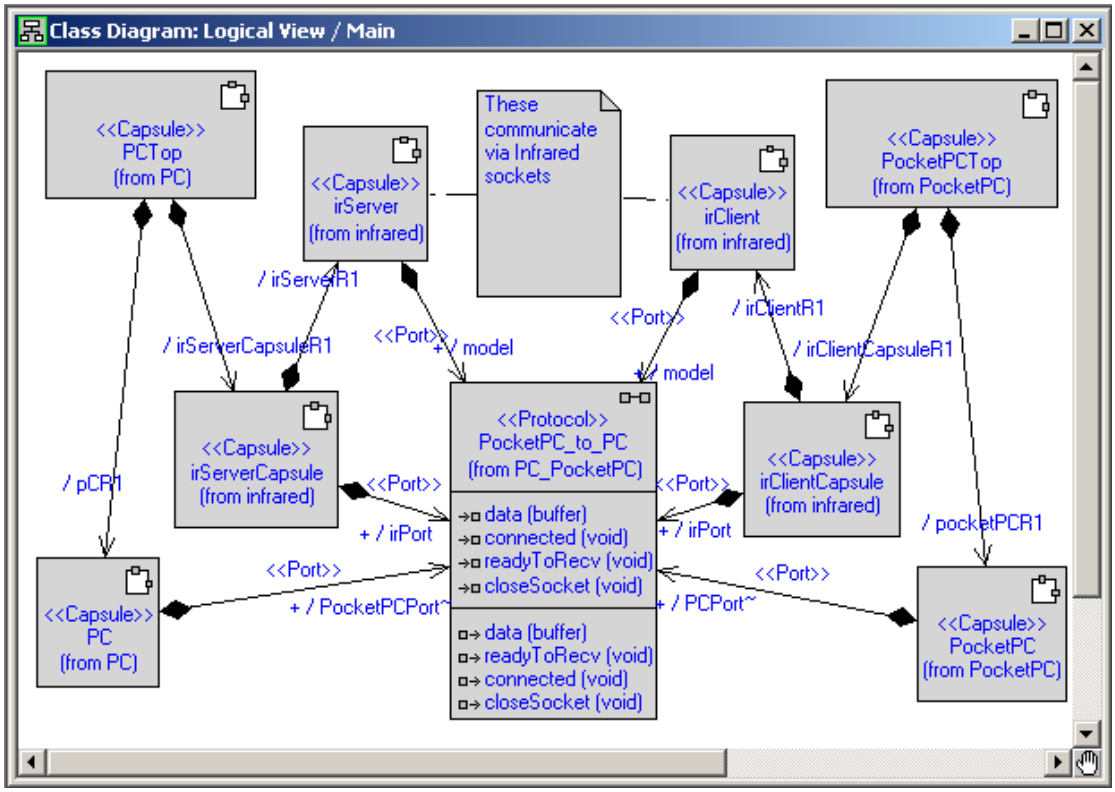


Figure 15 – Class Diagram overview of the complete Infrared application. This diagram shows only the relationships of the capsules to each other and the PocketPC_to_PC protocol.

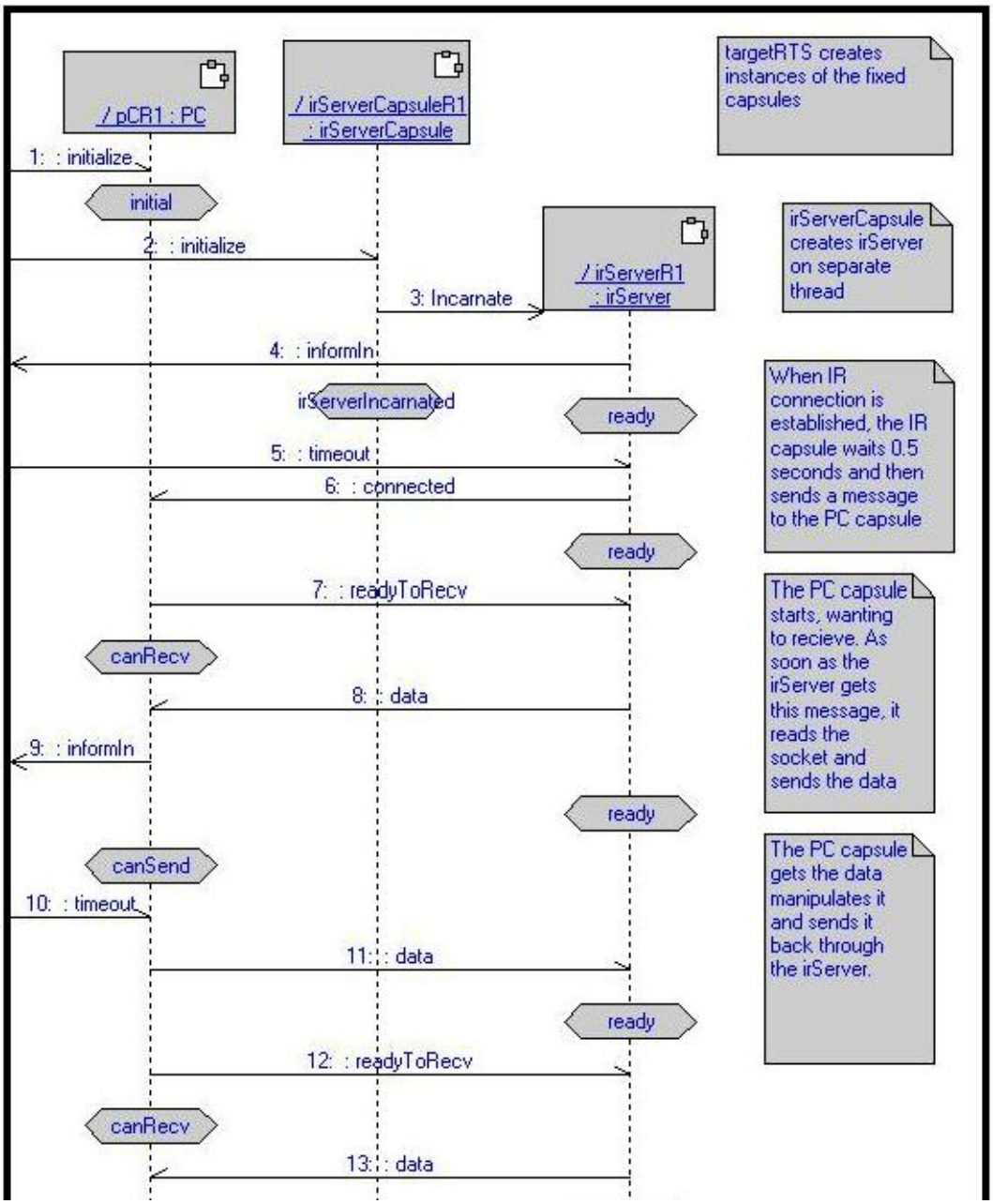


Figure 16 - Sequence Diagram for PC Application. This shows the sequence of events inside the PCTop Capsule captured from a running application. The notes down the right hand side have been manually included to aid understanding but Rose RealTime has generated all other elements. The broken lines represent time, against this are the states of the capsules and the messages between them. Also shown are the targetRTS messages, for example "informIn" being request a "timeout" message and "timeout" being the response).

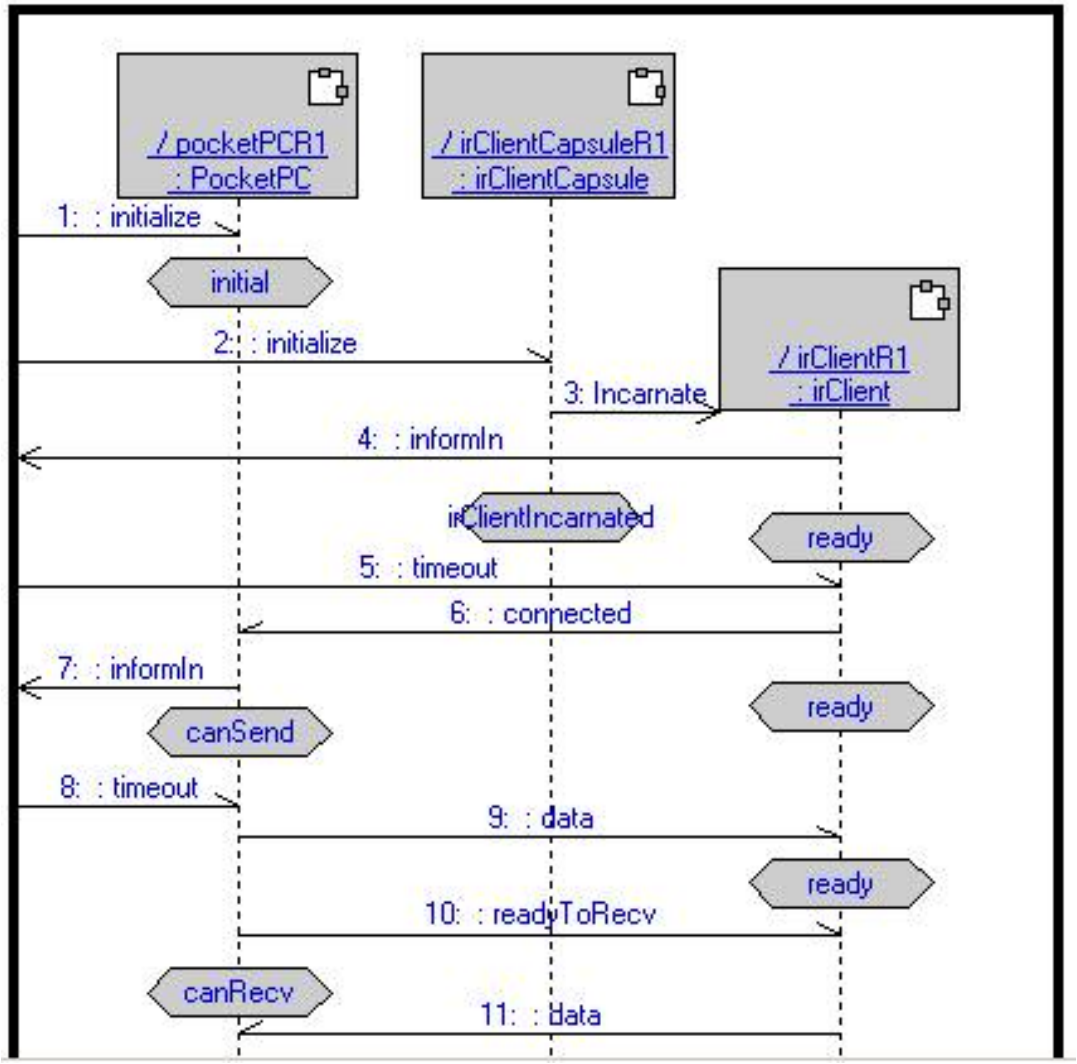


Figure 17 - Sequence Diagram for PocketPC Application. This shows the sequence of events inside the PocketPCTop Capsule captured from a running application. This was achieved across a serial link and captured at the same time as the PC application trace. Comparing the figure above and the PC application (Figure 16) it can be seen that after connected (message 6 for both diagrams) the PocketPC capsule goes to "canSend" state and the PC capsule goes to "canRecv".

Figure 18 - Notebook and Pocket PC IR connection. The position of the IR port on the Notebook and the Pocket PC make it a difficult development environment. The Pocket PC is upside down and behind the Notebook screen out of easy reach, having complete control from Rose RealTime and being able to see what was happening was a big advantage.

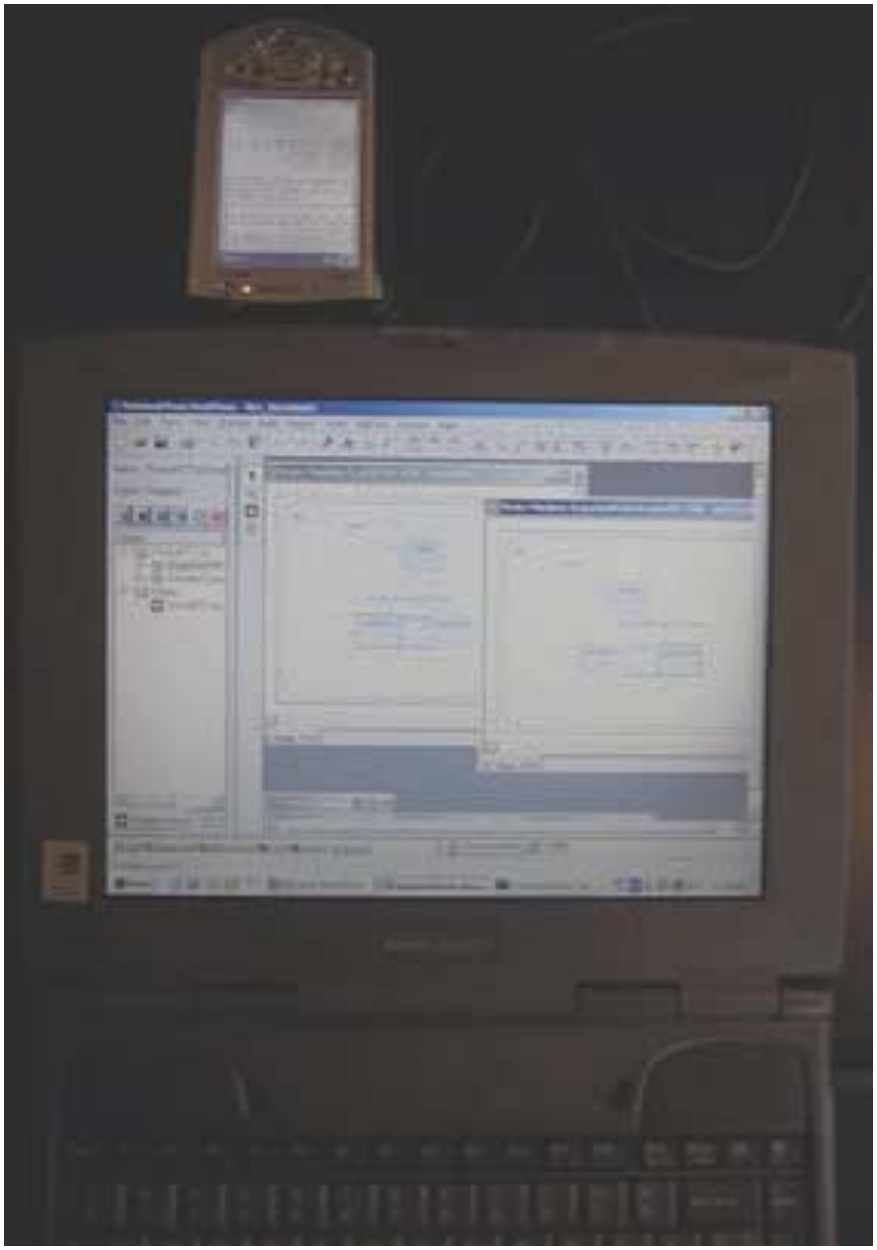


Figure 19 - Notebook and Pocket PC positioned to show applications running and being visually observed in Rose RealTime. The Pocket PC is displaying the RTConsole window which is a targetRTS implementation of a Windows console window (similar to the command prompt). The notebook is showing Rose RealTime monitoring the state machine on the PC (running locally) and the state machine of the Pocket PC (running remotely).

Conclusion: The results of this investigation showed that it was possible and advantageous to build an infrared application using Rose RealTime. Lessons were learnt in how to apply the technology to take best advantage of it. A simple application could be modelled entirely on a PC and then by distributed across targets which communicate via an infrared link. The process of doing this, once the IR capsules were available, required only a few modifications.

An added note to this conclusion, discovered during the writing of this report, is that UML provides a deceptively powerful way to document an application. The application was built completely using UML as the framework and as such was self-documenting as it was created. Anyone understanding UML can understand the application by looking at the UML diagrams presented in this report. As the reader may not be familiar a textural explanation has also been given, it was in the process of writing this it was discovered just how much information about a system had been accumulated. Describing just the basic behaviour of the system in text took more effort than to actually creating it.

6 Revised Development Road Map

The lessons learned from implementing the above simple application were then applied to a more complex application. This was done and again was successful. The application was a simple “biometric” concept device controlled remotely via infrared from the Pocket PC, and will be described in detail in the next section. The following describes the experiences from a process point a view.

6.1 *New Road Map*

From the previous model we now had reusable components. The IR Capsules had been designed to set up a link and communicate across it as instructed from whatever capsule it was doing the communication for. This should mean that the capsules were application independent. With this in mind, the new Road Map, again using iterative development, was as follows:

1. **PC Executable** - Model the complete system on the PC. The capsules, that are to be relocated later, communicate using the PocketPC_to_PC protocol.
2. **PC Executable and Pocket PC Exectuable communicating over infrared** – replace the direct PocketPC_to_PC protocol connection, with the IR Capsule connection and build the individual capsules for their respective targets.

In theory a developer going through the above iterations would spend a majority of the effort in designing the application in iteration 1. This would all be done on the host PC and the need for target testing would not be required. Once the model had been developed, the next iteration should take a matter of minutes, followed by testing that Application behaved the same as on the PC.

6.2 *Bumps in the Road*

The experiences in using the New Road Map on a more realistic application showed some of the flaws in the process and reusable components. The complete application took approximately 9 hours to code, although quick, approximately half the time was spent on infrared problems. In the next section, a new “Final Suggested Road Map” will be described based on the problems solutions described here.

The problems and solutions found while implementing the “Biometric” application were as follows:

- **Unrealistic test application** – the initial development road test used an unrealistic scenario. The application sent data back and forth continuously in a never-ending loop. In reality infrared connections are short lived. The more realistic application showed the need for an infrared socket implementation that did a short task, shutdown the connection and waited until a user prompt to create a new connection. The solution to this was to derive (using inheritance) the original IR Capsules to create the concept of a “one shot” Capsule (Figure 20) which maintained a link only long enough to send and receive a single buffer.

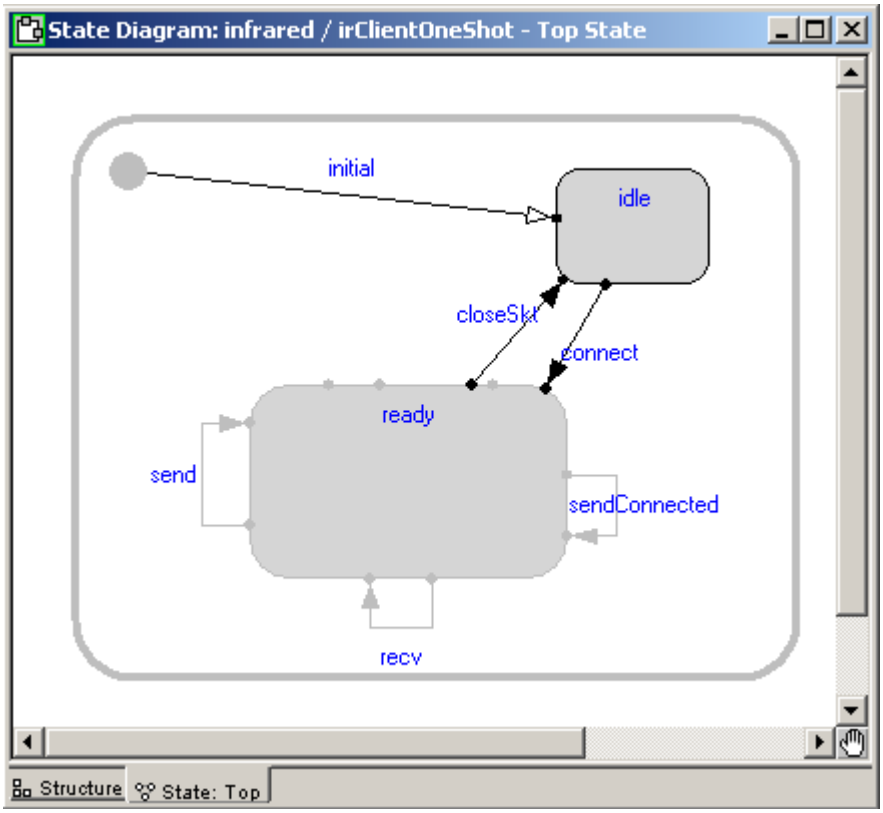


Figure 20 - "One Shot" IR Client Capsule – This is the State Diagram of the irClientOneShot capsule derived from irClient capsule. The grey coloured elements represent the inherited parts and the black elements are the changes (specialisations). This shows a new capsule behaviour which remains idle most of the time, connects only on request and returns to idle when done. The only change to the irServer side (not shown here) is to listen for a new connection.

- **Behaviour of Infrared connections** – when modelling on the PC the PocketPC_to_PC protocol connection is a direct one. This does not accurately recreate the behaviour of infrared. Also the capsules are privy to information that would not be there is the connection was over a real infrared link (e.g. is the other side really ready to receive connected?). The solution used here was to resolve the problems on the real target, which is not desirable. A better solution (suggested but not proven) would be to create an "infrared physical layer" capsule for development purposes. This could be created, also using the PocketPC_to_PC protocol, to mimic the behaviour of the physical media and catch problems before they occur. This would also allow the capsules to require no internal modification and allow both the single and distributed applications to be fully functional valid models.

- **Incomplete Implementation** – The infrared code used was still derived from the example code given with eMbedded Visual Tools [CEDOC2] and therefore the application was still a proof of concept. The application also had no GUI and the user actions were implemented using the Rose RealTime debugger's ability to inject messages. These had no impact on the final application, it was used exactly as designed and the user input could be recreated effectively. However, having "no impact on the final application" also means that these results lack robustness testing for error conditions or the possible side effects that a User Interface may introduce. The only solution to this would be more investigation.

6.3 *Final Suggested Road Map*

Based on the previous experiences the following is the suggested Road Map for future application development.

1. Create a package of reusable and easily derivable base IR Capsules. These should provide basic functionality which can be adapted to suit different physical uses. These capsules should use a common protocol (e.g. the PocketPC_to_PC buffer protocol)
2. Capsules that need to communicate with another capsule via infrared should do so via a buffer protocol (i.e. a signal of type buffer). Having this enables the application to be easily designed on a PC and later made infrared distributed.
3. To model an infrared application on a PC, the infrared medium should also be modelled as a capsule. For example, this report modelled the link directly but a better solution is believed to be that shown in Figure 21.
4. If done correctly, the developer should end up with a library of IR Capsules for different situations and the last phase would be to simple replace the connection with the appropriate capsules and rebuild for the specific targets (again demonstrated in Figure 21).

Note: The Road Map is only suggested as result of previous experiences that are known to work. Other methods of infrared development in Rose RealTime may be more appropriate (e.g. a different way to encapsulate the IR communication or Connexis). These would have to be proven before suggesting using them for a development project based on them.

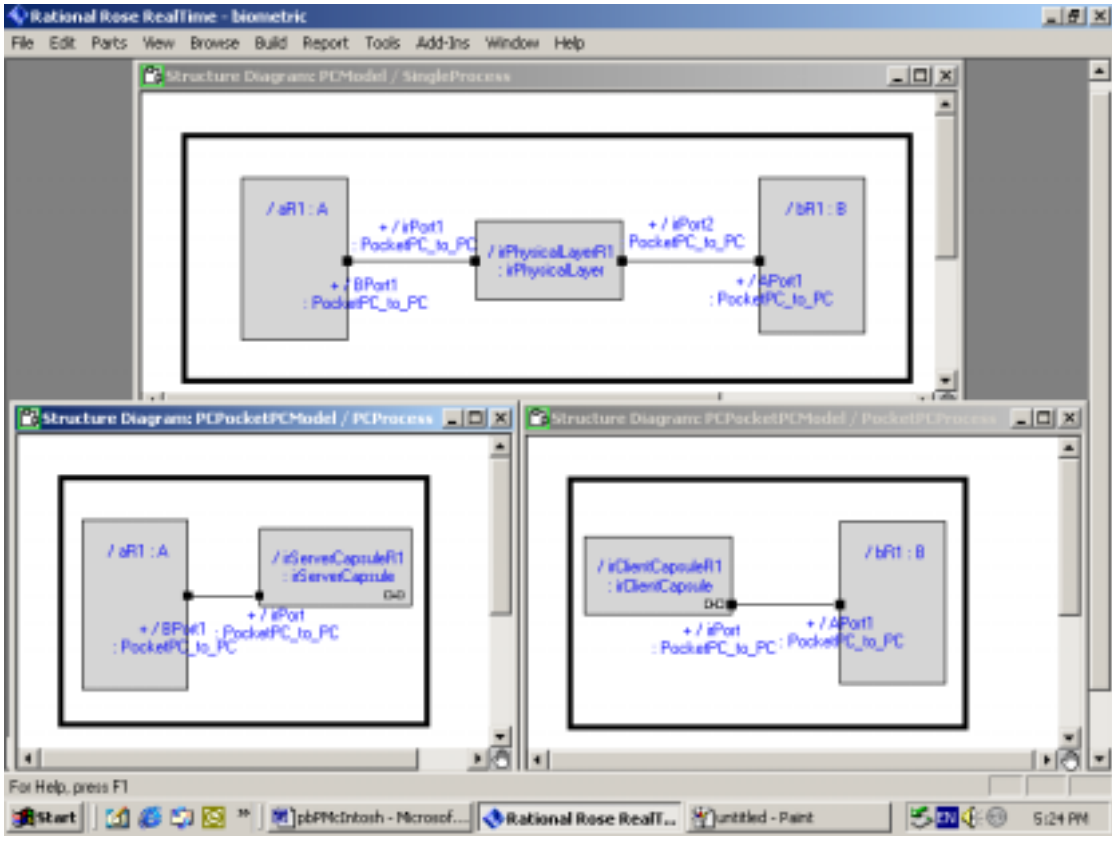


Figure 21 - Suggested Development Road Map example. Ideally, if the IR communication can be modelled as a capsule, then SingleProcess structure diagram (top) and the distributed infrared application (below left and below right) should behave exactly the same. If this can be achieved then distributing a model is as simple as removing the irPhyscialLayer capsule, replacing it with an appropriate IR Client or Server capsule and building for the specific target.

7 Example Application using Revised Development Road Map

For the example application a “Horse Biometric” device was chosen due to the presentation date for the report being Melbourne Cup day (a nationally observed horse race in Australia). The idea behind the application was to answer a need to monitor the health of horse during transportation. The following Use Case Diagram (Figure 22) describes the need to be able to monitor a horse’s heart rate, temperature and to be able to sedate the horse if it threatens to hurt itself. This fictional device used a notebook computer to represent a monitoring device which would be permanently located on the horse. This device kept information about the horse’s health and controlled a sedative injection device. In reality such a device would be much smaller and would more likely be implemented with Windows CE than Windows 2000. The Vet, requiring to tend to a number of horses in a variety of situations, is able to use a Pocket PC to easily access the device on the horse using infrared.

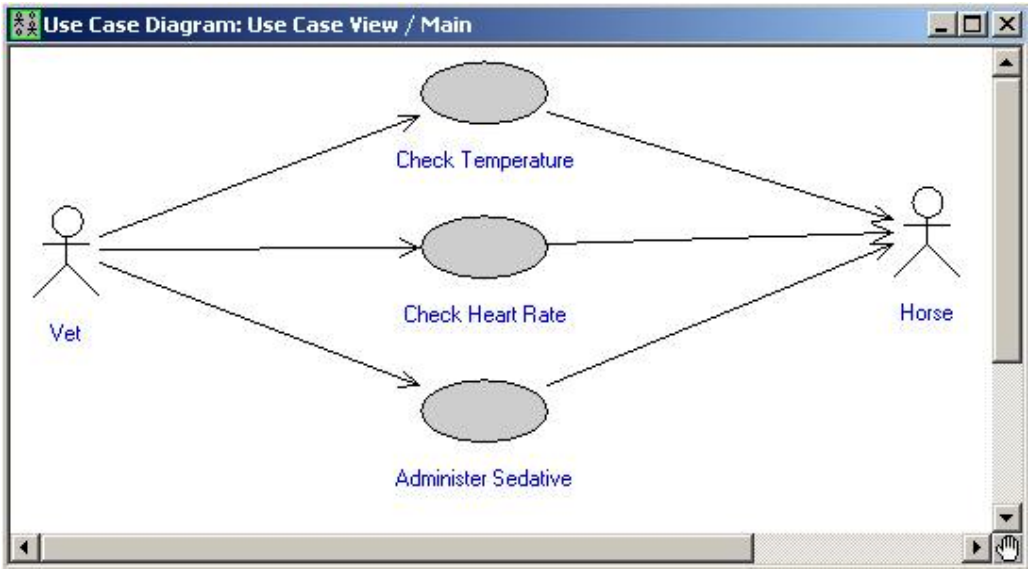


Figure 22 - Use Case Diagram for "Horse Biometric" infrared application.

The Structure Diagram (Figure 23) of the initial PC model best shows the complete application, as this is the system before it has been distributed. The BioControl capsule will eventually be built for the Pocket PC and communicate via infrared capsules. This capsule handles the user requests and communicates with the rest of the capsules via the BioRemoteControl capsule to complete the user request. The BioRemoteControl, BioMonitor, Intravenous, Heart and Temperature capsules will eventually be built for the notebook, as the “horse side” part of the application. These do the following:

- Heart – Mimics a horses heart and heart rate monitor device. This capsule reacts to a sedative message from the Intravenous capsule and updates the BioMonitor capsule with a heart rate measurement every 5 seconds.

- Temperature – represents a temperature measurement device which updates the BioMonitor capsule with a temperature reading every 9 seconds.
- BioMonitor – records the temperature and heart rate. It sends a status report on request.
- BioRemoteControl – Handles user requests and communicates with BioMonitor and Intravenous capsules to get health readings and control sedation.
- Intravenous – Mimics a sedation injection device. When “on”, at the request of the user, it sends a sedation value to the heart every 12 seconds.

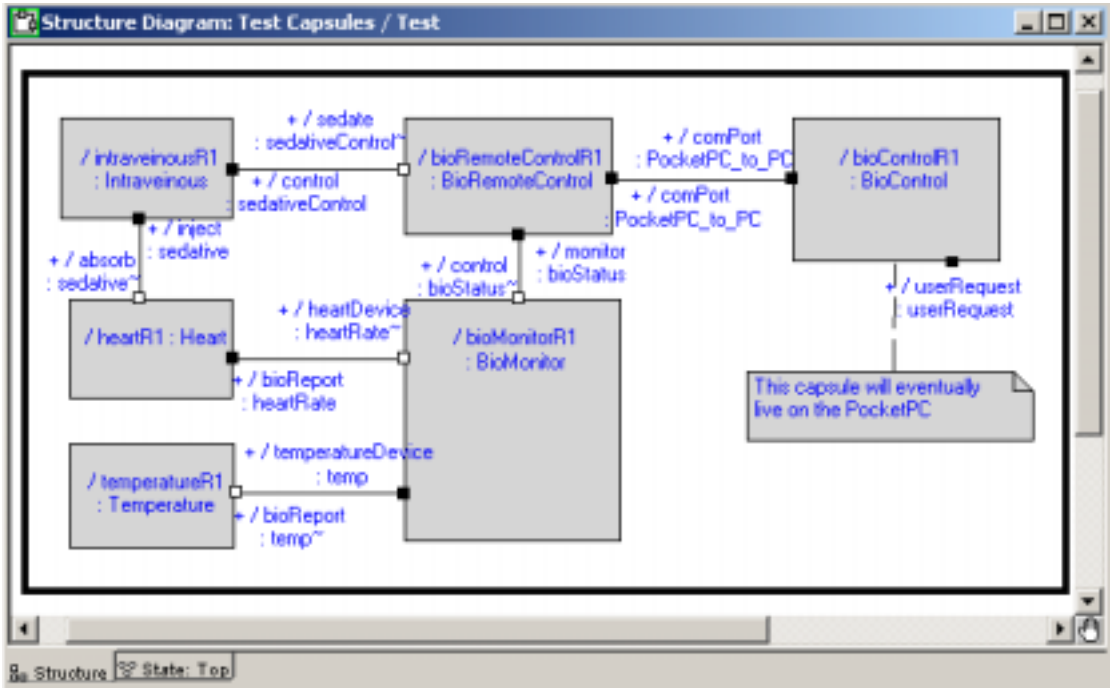


Figure 23 - Structure Diagram of complete Horse Biometric application, before being distributed over infrared.

During the development of this model it was found that a permanent infrared connection did not meet the user’s requirements. To address this, the capsules which communicated successfully in the test application where generalised to allow reuse through inheritance. A BufferChat capsule was created which had state behaviour common to both server side and client side. From this capsule were derived server side and client side specific capsules, these could then be derived further to fit the application need. The class diagram (Figure 24) is given below and the state diagrams, of the BioRemoteControl and BioControl (Figure 25), demonstrate the behaviour of the final “leaf” capsules.

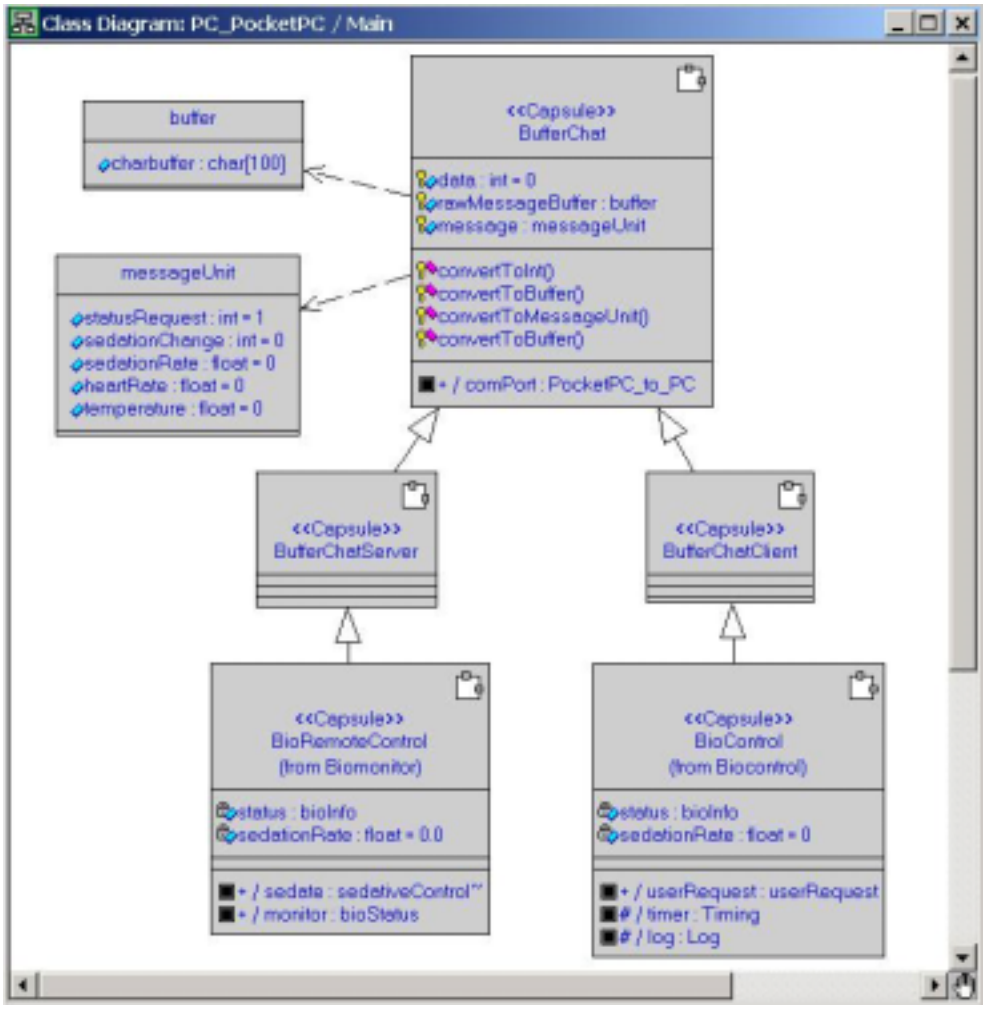


Figure 24 - Class Diagram showing strategy for component reuse. Note: The messageUnit class, this was used to encapsulate the data so it was common across Server and Client.

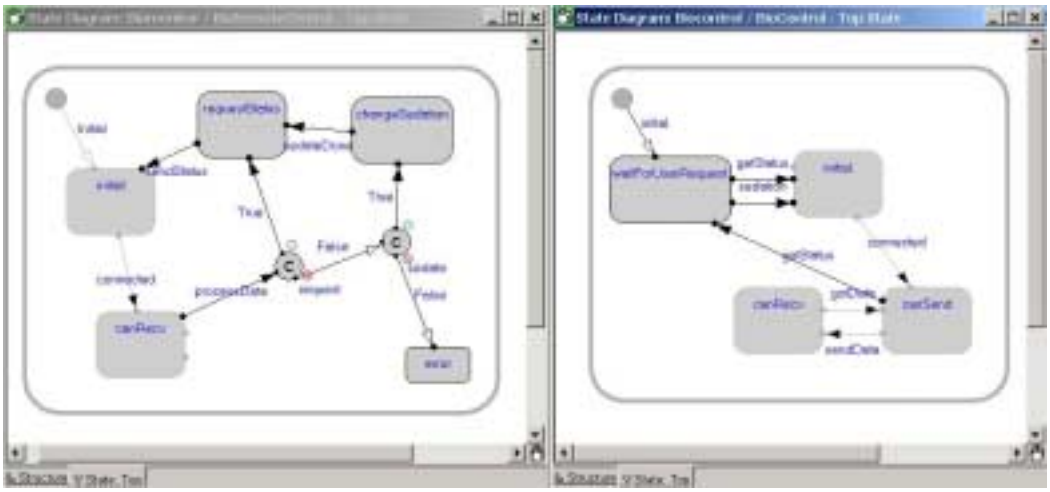


Figure 25 - State Diagrams showing the behaviour of the BioRemoteControl and BioControl capsules. These are the capsules that communicate via infrared.

The operation was designed so that the Vet could easily carry the handheld device and maintain information about a horse. When the Vet felt the need to get a new status or sedate the horse, he or she would then issue the request and “then” point the device at the horse. As you can see from the image below (Figure 26) maintaining a permanent infrared connection and using the handheld at the same time would not be easy (remembering that the IR port could reside on the collar of an overexcited horse).

Figure 26 – Using the Biometric application. The user needs to physically line up the IR ports, so the application was designed to have the request and connection actions separate. The application did not require the user to interact with the handheld while trying to establish an IR connection. After the request, the user would just point and wait for an audible indication that the request had been completed.

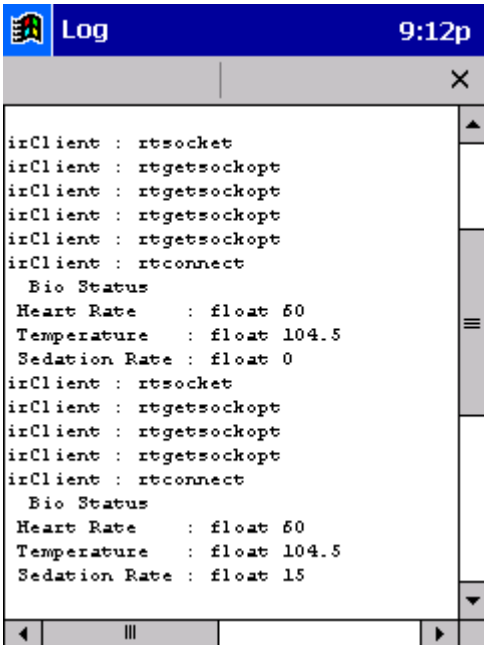


Figure 27 – Biometric Pocket PC screen shot. This has been instrumented to show the behaviour of the irClient socket. This shows the results of request for "Bio Status" and a request to change sedation rate, which also returns a "Bio Status" as confirmation. In reality this would use a GUI interface.

Conclusion: Although still a very simple application, development of this showed itself to be a valuable exercise to undertake. Not only did this prove that development of infrared applications with Rose RealTime was practical but it also uncovered requirements of real-world infrared applications that I had not previously thought of.

This complete application took an experienced Rose RealTime user approximately 9 hours to create. Using the "Final Suggested Road Map" mentioned in section 6.3 the development time could probably be halved. This is based on the fact that half the time was spent resolving problems should not occur with the suggested road map. In addition to the quick development time the design was self-documenting and reusable.

8 Conclusions

At the conclusion of this report it can be stated that Rose RealTime can successfully build an application for a Windows CE 3.0 ARM target. Rose RealTime can also provide all the development functionality given to other target processors, so the advantages Rose RealTime gives to those targets extends to the Windows CE 3.0 ARM. With the configuration described in this report a developer is able to design in UML Windows CE 3.0 real-time embedded applications, generate code from the design, compile, debug and deploy, all from within the one tool.

At completion of this research the final environment versions are as follows:

Rational Rose RealTime – 2001a.04.00 Patch 2 (6.3.120.0)
eMbedded Visual Tools – 3.00.0099.0
clarm.exe ARM compiler version - 12.01.8569 for ARM
Visual C++ – 6.0
cl.exe Host compiler version – 12.00.8168 for 80x86
ActiveSync - 3.5 (Build 1176)
Windows OS – 2000 SP2
Host – Toshiba Satellite 2520CDT notebook/ Dell Precision 220 Workstation
Windows CE – 3.0.9348 (Build 9616)
Pocket PC – Compaq iPAQ HP 3630 ROM Revision 1.77.00 ENG

The above represents the complete environment configuration in which development was undertaken. In the last week of the writing of this report, the latest version of Rose RealTime (2002.05.00) was released. Although time was limited it was possible to confirm the following against the current Rose RealTime 2002.05.00 (6.4.87.0) release.

- TechNote – “Porting RoseRT C++ TargetRTS to Windows CE 3.0 - ARM processor” [TN18596] is still valid for the new release (with one amendment that “#define WIN32_PLATFORM_PSPC” is no longer in the RTLibSet.h file).
- TechNote – “Target Observability to a Windows CE device using Serial Cable” [TN20458] is still valid for the new release.
- The Windows CE 3.0 - ARM IR and MFC models created in the 2001a.04.00 release still load, build and can be debugged in the new release (Figure 28). The models also worked as they did previously, including the Biometric application.

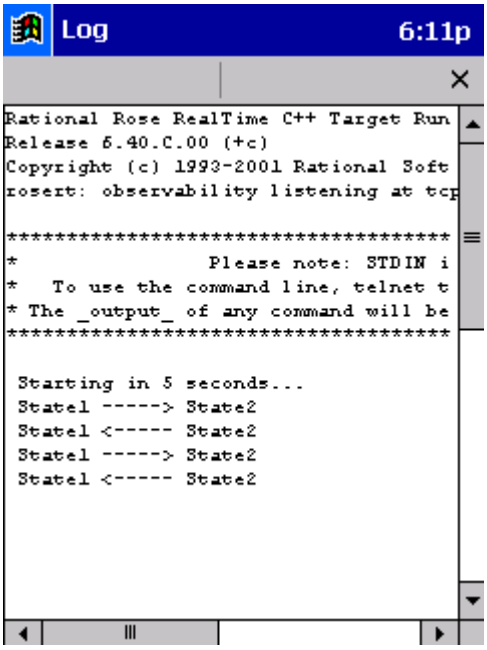


Figure 28 – Screen shot of "Hello World" model loaded, built and running using the latest Rose RealTime 2002.05.00 release. Operation is exactly as before apart from the output stating, "Release 6.40.C.00 (c+)" which is the internal build version of the latest targetRTS.

- TechNote – "Porting Connexis Libraries to Windows CE 3.0 - ARM processor" [TN21927]] is still valid for the new release.
- The Windows CE 3.0 - "BasicTest" and "PingPong" Connexis models created in the 2001a.04.00 release still load, build and can be debugged in the new release.

In addition to the above “last minute” validations in this report, eMbedded Visual Tools 4.0 Beta and the Pocket PC 2002 SDK were successfully ported to the latest release. The original Porting Technical Note [TN18596] was derived and new Porting Technical Note [23788] (Appendix F) created. As the Pocket PC 2002 is not available, this can only show that, in theory, that this report will also be valid for the future release of Windows CE. The Technical Note has been left as an internal document until it is validated fully.

8.1 *Rose RealTime for Infrared*

For infrared, Rose RealTime has been proven to allow development of infrared applications. Although, infrared applications can vary in physical needs (i.e. permanent connections or one off connections) component reuse is still possible with inheritance. Also, as different development projects are undertaken, a “library” of reusable infrared components will quickly form with components being reusable “as is”.

The infrared examples here were merely proof of concept. Although, development was very easy, other methods may be more appropriate. Connexis for example may provide a better solution and negate the need for much of the design work. Another option may be to have a capsule that fully encapsulates infrared by decoding the protocol message directly (i.e. instead of using buffer protocols, have a capsule which can encode the message as the targetRTS would).

Having the ability to trace and monitor the capsule behaviour, aided greatly in debugging a distributed system to work. Both applications could be traced and the sequence of events compared. Although I have not debugged infrared code by other means, my experiences with TCP/IP debugging would be similar, and the tracing ability proved valuable.

Note: The UML models referred to in this report have been “Webpublished” to Paul McIntosh’s Tech Page [WEB11] www.internetscooter.com/tech *to enable the reader access the full context of the models discussed.*

9 Recommendations

At completion of this report the following are some recommend directions for future research in this area:

USB Target Observability

With understanding of how the Windows CE USB driver works, that is the USB driver which resides on the host, Target Observability should be possible.

QUtilities Target Control

QUtilities, a Windows CE window manager, should provide a more elegant way to shutdown applications remotely. If information can be found of how this works it could be utilised in Target Control.

MFC Integration

The ability to generate MFC applications from Rose RealTime was proven in this report. This may be an opportunity to extend this to a complete Rose RealTime MFC framework, for both Windows and Windows CE application development.

Re-usable MFC Components

Even if complete MFC frameworks, mentioned above, are not practical. Reusable GUI components would be of benefit for Windows CE development in Rose RealTime.

IR Target Observability

Remote Access Services may also be available via IR. If, so it may be possible to do complete application development across infrared. This may not be practical for a device which has other connection options but may provide an interesting development solution (e.g. development on a device that only has IR).

Connexis over IR

Connexis over IR should be possible but whether it would lend itself to the physical operation characteristics of IR would require investigation.

Full Encapsulated IR Capsules

Messages between capsules are encoded and decoded. It may be possible to utilise code already existing in the TargetRTS to enable a capsule to translate messages to IR transparently.

Bluetooth

The latest iPAQ Pocket PC's now come with Bluetooth and investigation similar to that in this report should be possible.

10 Glossary of Terms

Where possible definitions have been taken directly from the Rose RealTime On-Line documentation [RTDOC1] or the Microsoft eMbedded Visual C++ 3.0 On-Line documentation [CEDOC2].

ARM	Advanced RISC Machine processor. An embedded processor which uses a Reduced Instruction Set.
Biometric	Biological Measurement. In this report, it refers to a device which can measure heart rate and temperature.
Bluetooth	Short range wireless communication standard.
Capsule	Capsules are the fundamental modelling element of real-time systems. A capsule represents independent flows of control in a system.
Class	A class is a design-time specification for one or more distinct objects with common structure, attributes, behavior, and operations.
Class Diagram	Class diagrams show the static structure of the model.
Connexis	A Rose RealTime library, which is an add-in product that provides connectivity for Unified Modelling Language (UML) models.
Decode	Convert from a string of character in to a data type.
Encode	Convert from a data type into a string of characters.
GUI	Graphical User Interface
Host	The computer system which is used for development purposes (as opposed to “target” which is the system be developed for). In this report the host is a desktop PC and the target is the Pocket PC.
IDE	Integrated Development Environment – A single Windows interface for a number of software development applications.
Infrared	Light emissions, close to the colour red, outside the visible spectrum, used for data transmission.
IR	Infrared
IrDA	Infrared Data Association – IrDA is an International Organization that creates and promotes interoperable,

	low cost infrared data interconnection standards that support a walk-up, point-to-point user model.
MFC	Microsoft Foundation Classes
Microsoft Foundation Classes	The Windows CE version of the C++ class library that Microsoft provides with its C++ compiler for creating Windows-based applications.
Model	The UML representation of a software system.
OS	Operating System – The software layer between the hardware and the application (e.g. Windows 2000, Windows CE, Unix).
PC	Personal Computer – In this report a Desktop or Notebook system.
Perl	Scripting language.
Physical Layer	The ISO/OSI model layer which handles the binary code to physical media signals transition, in the communication hierarchy.
Pocket PC	Handheld device which supports Windows CE 3.0 – in this report a Compaq iPAQ Pocket PC.
Port	a) Ports are objects whose purpose is to send and receive messages to and from capsule instances. b) The process of rebuilding code to work with a different target processor. c) The hardware device for infrared connections.
Protocol	The set of messages exchanged between two objects conforms to some communication pattern called a protocol.
RAS	Remote Access Services - A Windows NT feature by which a single serial connection provides a remote workstation with host connectivity.
Real-Time	The term given to time dependant applications.
Rose	UML Modelling Tool produced by Rational Software
Rose RealTime	Real-time UML Modelling Tool produced by Rational Software
Sequence Diagram	An interaction is a pattern of communication among objects at run-time.
SH3	An embedded processor created by Hitachi and used in the HP Jornado.

Socket	The name for a TCP/IP or IR connection.
State Machine	State machines are used to model the dynamic aspects of a system.
States	A state is a condition during the lifetime of an object where it is ready to process events.
Structure Diagram	A diagram that shows the collaboration of capsules for a particular system.
Target	The computer system is being developed for (as opposed to “host” which is the system the development occurs). In this report the host is a desktop PC and the target is the Pocket PC.
Target Control	The ability to control deployment and execution of a developed application from within Rose RealTime.
Target Observability	The ability to visually debug a UML application at run-time.
TargetRTS	Target Run Time Services library – The Rose RealTime library which is a framework for real-time UML modelling of applications.
TCP/IP	Transmission Control Protocol/Internet Protocol – A connection oriented method of communication between applications across a network.
Tool Chain	In developing for target applications, there are a number of components which are required to build and deploy and executable. The complete set of these components is called a Tool Chain.
Transition	A transition is a relationship between two states: a source state and a destination state. It specifies that when an object in the source state receives a specified event and certain conditions are met, the behavior will move from the source state to the destination state.
UDP/IP	User Datagram Protocol/Internet Protocol - A connectionless method of communication between applications across a network.
UML	*Unified Modelling Language – a graphical language for visualizing, specifying, constructing, documenting, and executing software systems.*
Windows CE	Microsoft’s embedded real-time version of Windows operating system.

Winsock	Windows Sockets - A programming interface used to provide socket communication.

11 Bibliography

Below is a list of literature referenced for this research, also included in section 12 is the literature review, submitted as part of the initial project proposal.

[RTDOC1] Rational Rose RealTime - Release v2001a.04.00 Product Documentation (2001)

[RTDOC2] Developing Real-Time Software With Rational Rose RealTime - v2001.03.00 Training Documentation (2000)

[RTDOC3] White Paper: Designing Software using a UML Case Tool that Supports Software Distribution. Terrence Barrington, Gustave Lamperez. Lucent Technologies, Advanced Optical Networking Center (2000)

[RTDOC4] Designing for Concurrency and Distribution with Rational Rose RealTime. Garth Gullekson. Rational Software White Paper (2000).

[CEDOC1] Windows CE 3.0 Application Programming. Nick Grattan, Marshall Brian. Prentice Hall (2001)

[CEDOC2] Microsoft eMbedded Visual C++ 3.0 - 3.00.0099.0 Microsoft Product Documentation (2000)

[CEDOC3] Get Your Windows CE Device Talking With IrDA. Michael Heydt. Microsoft Internet Developer Article (1999)

[CEDOC4] The Windows CE Technology Tutorial. Chris Muench. Addison-Wesley (2000)

[PKDOC1] iPAQ H3000 Pocket PC Reference Guide, Second Edition - Compaq Product Documentation (2000).

[PKDOC2] How Pocket PC "Talks" with a Cell Phone. Arne Hess. Club PocketPC Article (2000)

[IRDOC1] Technical Summary of "IrDA Data" and "IrDA CONTROL". Infrared Data Association. (2000)

[IRDOC2] IrDA Infrared Communications: An Overview. Patrick Megowan, David Suvak, Charles Knutson. (1998)

[IRDOC3] Serial Infrared Physical Layer Specification, Version 1.3. Infrared Data Association (1998)

[IRDOC4] Serial Infrared Link Access Protocol (IrLAP), Version 1.1. Infrared Data Association (1996)

[IRDOC5] Link Management Protocol, Version 1.1. Infrared Data Association (1996)

[IRDOC6] 'Tiny TP': A Flow-Control Mechanism for use with IrLMP, Version 1.1. Infrared Data Association (1996)

[IRDOC7] 'IrCOMM': Serial and Parallel Port Emulation over IR (Wire Replacement), Version 1.0. Infrared Data Association (1995)

[IRDOC8] IrDA Object Exchange Protocol, IrOBEX, Version 1.2. Infrared Data Association (1999)

[IRDOC9] Minimal IrDA Protocol Implementation (IrDA Lite), Version 1.0. Infrared Data Association (1996)

[IRDOC10] LAN Access Extensions for Link, Management Protocol, IrLAN, Version 1.0. Infrared Data Association (1997)

[IRDOC11] Programming With Infrared Sockets - Whitepaper, Prasanna .V, California Software Laboratories (1998)

[IRDOC12] Connecting Windows and Non-Windows Devices with IrDA. Mike Zintel (2000)

[WEB1] Rational Rose RealTime (Rational)
http://www.rational.com/products/rosert/

[WEB2] The Rational Edge (Rational)
www.therationaledge.com

[WEB3] The Infrared Data Association
www.irda.org

[WEB4] Pocket PC (Microsoft)
www.microsoft.com/mobile/pocketpc/

[WEB5] Windows Embedded Developer Center (Microsoft)
http://msdn.microsoft.com/embedded/

[WEB6] Windows CE (Microsoft)
www.microsoft.com/windows/embedded/ce/

[WEB7] IPAQ Pocket PC (Compaq)
www.compaq.com/products/handhelds/pocketpc/

[WEB8] Windows CEcity
www.wincecity.com

[WEB9] ActiveSync
http://www.microsoft.com/mobile/pocketpc/downloads/activesync35.asp

[WEB10] Visual eMbedded Tools
http://www.microsoft.com/mobile/developer/downloads/emvt30/

[WEB11] Paul McIntosh's Tech Page
http://www.internetscooter.com/tech

[COM1] Stephen Rooks, Rational Software. Email Wed 13/06/2001 to rose-rt@rational.com internal email list. Subject: RE: Using connexis to connect OSE 4.31. and WinCE 3.0 Targets?
[COM2] Stephen Rooks, Rational Software. Email Tue 26/06/2001 to rose-rt@rational.com internal email list. Subject: Demo platform

[COM3] Steve Saunders, Rational Software. Email Tue 28/08/2001 to Paul McIntosh, Rational Software. Subject: RE: WinCE Target Control Scripts

[COM4] Daphne, Infrared Data Association. Email Wed 11/04/2001 to Paul McIntosh, Rational Software. Subject: Re: Can I use the IrDA specs for my Uni Assignment

[TN18596] Porting RoseRT C++ TargetRTS to Windows CE 3.0 - ARM processor. Rational Technical Note. 26-Jun-2001. http://www.rational.com/support/technotes/index.jsp

[TN19967] Using links to start Windows CE applications with Rose RT command options. Rational Technical Note. 6-Aug-2001. http://www.rational.com/support/technotes/index.jsp

[TN20458] Target Observability to a Windows CE device using Serial Cable. Rational Technical Note. 23-Aug-2001. http://www.rational.com/support/technotes/index.jsp

[TN21927] Porting Connexis Libraries to Windows CE 3.0 - ARM processor. Rational Technical Note. 8-Oct-2001 . http://www.rational.com/support/technotes/index.jsp

[TN23788] Porting RoseRT C++ TargetRTS to Windows CE 4.0 - ARM processor. Rational Technical Note. 30-Nov-2001. Internal Draft Technical Note

[UN1] An Overview of Common Research Approaches. John Hughes. UTS IT Research Methods Study Guide (2001).

12 Literature Review for Rational Rose RealTime, Windows CE 3.0, Compaq iPAQ Pocket PC and Infrared

The literature review is presented in two parts, a "Brief Review" and "Extended Review". The intention is that the Brief Review presents the review in the form that can be scanned and, if desired, addition information can be found in the Extended Review. Also reviewed are the Web Resources from which some of the literature has been taken, this provides a context to how the information has been delivered.

A general comment on the literature reviewed. Most of the literature was generated by a company/association promoting or documenting a product/standard/concept. As the success of the things being promoted is dependant on adoption by new users, the literature is well written, well thought out and accurate. The information though will be biased towards the positive aspects of the product/standard/concept. For the purpose of this project proposal though, the merits of the products and standards are not under question and therefore the literature is reviewed against usefulness with regard to the Thesis topic.

Reviewer's Background: I am a Senior Technical Support Engineer at Rational Software supporting Rose RealTime and other embedded development products. I have a broad software/hardware knowledge developed from 13 years industry experience mainly in test and integration. I have no serious development experience of Windows applications. My research interests are in applying technology to improve processes, particularly to improving software development processes with visual software utilities.

12.1 *Rose RealTime Literature*

A general comment of the literature associated with Rose RealTime. As most readers would be unfamiliar with the concepts around Rose RealTime, all the literature reviewed (including this project proposal itself) has information targeted at anyone from novice to expert. The authors have done this task well but novice readers should be aware that Rose RealTime, as a Development concept, would require the benefit of practical experience with the toolset to fully appreciate what is being relayed in the more expert topics. On the flip side, expert users will find the introductory concepts repeated.

In addition to this, the limitations of static text and images have an impact on the readability of the information. Rose RealTime is a visual modelling tool and the application development process is a visual experience. The visual development experience is not static, the developer builds a system by "drawing" it, building it up and seeing the system state behaviour visually in real-time at each step. Unless the reader has witnessed this first hand, matching up the textual description with the events can be deceptively difficult.

Rational Rose RealTime - Release v2001a.04.00 Product Documentation (2001) [RTDOC1]

Brief: The product documentation contains several hundred pages of user information on different components of Rose RealTime. The information is the complete guide to Rose RealTime and is essential to all users of Rose RealTime whether experienced or not.

Extended: The information is provided as on-line help and hardcopy. The product documentation defines the user specifications of Rose RealTime and therefore accurate and under constant review (by users daily). The nature of the documentation being a grouping of different material (e.g. Connexis, Modelling Guide, Target Porting Guide, etc.) means that the style can vary from pure reference material to step-by-step tutorials. The documentation is good at all user levels, providing animated images for new users and detailed information where needed for experienced users.

The on-line version contains example models demonstrating various modelling techniques. Of particular interest to this project is the Socket Interface Model, which describes how to maintain message processing with threads that block (due to sockets). Also of interest are the Connexis “Ping Pong” models and associated tutorial, which will be the base test model for the Connexis part of this project.

Developing Real-Time Software With Rational Rose RealTime - v2001.03.00 Training Documentation (2000) [RTDOC2]

Brief: This literature is only of value in combination with the training course. Combined with the training course this provides valuable information and experiences for a new user.

Extended: As this is training documentation the information is summarised and expanded upon during the training (through the lecture notes). The literature is therefore not useful for a reference point of view. However, anyone having done the course and exercises may find it useful as a “memory jogger”. The course provides a starting point for new users, taking them through Object-Oriented Analysis and Design through to creating a Telephone Call Model example. As stated in the general comment, experience is valuable in learning Rose RealTime and the course provides practical experience.

White Paper: Designing Software using a UML Case Tool that Supports Software Distribution. Terrence Barrington, Gustave Lamperez. Lucent Technologies, Advanced Optical Networking Center (2000) [RTDOC3]
http://www.rational.com/media/whitepapers/RoseRT_Case_Study_Lucent.pdf

Brief: A good short 5 page summary of using Rose RealTime and Connexis from a non-Rational Software users point of view. The information is based on an old release of Rose RealTime so some of the information is no longer valid but it is still useful.

Extended: Lucent has taken time to assess Rose RealTime's good and bad points. The paper provides valuable information as it gives real life user experiences and lessons learned. Unfortunately the "publicly" available whitepaper does not contain the complete picture and references it gives to Lucent internal documents are not accessible.

Designing for Concurrency and Distribution with Rational Rose RealTime. Garth Gullekson. Rational Software White Paper (2000). [RTDOC4]
http://www.rational.com/media/whitepapers/concurrencyroserealtime.pdf

Brief: Good information for someone unfamiliar with Rose RealTime and the concepts of using UML for embedded systems. People familiar with the concepts will also find useful information in the form a capsule design advice.

Extended: Garth Gullekson has many years experience with real-time systems and also the toolset. This paper is a polished article which takes the reader through the problems, solutions that active capsules and other real-time UML extensions provide, how Rose RealTime operates and how to approach design. This has good coverage of the problem domain, which gives depth to the explanation of the solutions. Also a good white paper to read for anyone who does not have access to Rose RealTime as it does not concentrate on the toolset functionality.

12.2 *Windows CE 3.0 Literature*

Windows CE 3.0 Application Programming. Nick Grattan, Marshall Brian. Prentice Hall (2001) [CEDOC1]

Brief: A good source for most elements of Windows CE programming excluding the user interface. This book also includes a CD which has Microsoft eMbedded Visual C++ 3.0 and working example code to go with the book.

Extended: Infrared communication is approached in this book only to a small extent but enough information is provided to be able develop a simple infrared application. There is also infrared example code provided which displays a list of infrared devices detected through the infrared port.

Although this book lacks information on infrared, it provides a good ground for the other information that would be needed for developing a Windows CE 3.0 application such as creating processes and handling files. Having eMbedded Visual C++ 3.0 and example code also enables the reader to correlate the books information with working examples. The infrared examples do work and demonstrate a simple application quite effectively without the need for a remote application to communicate with.

Microsoft eMbedded Visual C++ 3.0 - 3.00.0099.0 Microsoft Product Documentation (2000) [CEDOC2]

Brief: Comprehensive and well set out information on developing Windows CE applications using eMbedded Visual C++. Provides detail on infrared and has client/server example code.

Extended: Especially considering that eMbedded Visual C++ is provided free, the quality of the documentation is impressive. The documentation provides information on all areas of application development on Windows CE 3.0 and does so clearly.

Infrared communication is explained and example code given. The documentation is searchable and information about infrared applications can be found in several areas. The documentation mainly focuses in WinSock for infrared communications. It does mention raw IR, lower level control, but unfortunately does not provide much information about how to use it.

In addition to the basic information, the documentation puts infrared communications in perspective with the International Organization for Standardization Open Systems Interconnection (ISO/OSI) model. The documentation provides diagrams, which show simply and clearly the Windows CE communications services against the 7 OSI layers.

Get Your Windows CE Device Talking With IrDA. Michael Heydt. Microsoft Internet Developer Article (1999) [CEDOC3]
http://www.microsoft.com/mind/0599/wince/wince.htm

Brief: This article is now dated and many of the issues documented are now fixed with the current releases of the OS's. I would not recommend reading this but I would recommend locating it and keeping it for reference as it still has information that may be required for troubleshooting.

Extended: This was written against Windows CE 2.0 and Windows 2000 beta 2. The author explains the difficulties encountered due to lack of IrDA support. IrDA support is provided with Windows 2000 and these difficulties should not be encountered. The article should be noted though for the coverage of practical issues that would be useful for troubleshooting. There is information on installing IrDA drivers, verifying installation and a client/server application explained in some detail.

The Windows CE Technology Tutorial. Chris Muench. Addison-Wesley (2000) [CEDOC4]

Brief: This book is pertaining to Windows CE 2.0 knowledge, it contained good information, which was used to implement Remote Access Services.

Extended: This was written against Windows CE 2.0 and was originally not looked at during the project proposal literature review. However, it was found that some of the Windows CE technology, which was still present in 3.0, escaped documentation. This book provided a good source for this information and provided clear step-by-step instructions.

12.3 *Compaq iPAQ Pocket PC Literature*

iPAQ H3000 Pocket PC Reference Guide, Second Edition - Compaq Product Documentation (2000). [PKDOC1]

Brief: Basic user manual for Pocket PC owners. This is required for knowing the capabilities of the Pocket PC model.

Extended: An 116 page document, which provides basic user information. The manual is provided in the form of a PDF with the Pocket PC CD. Included is a section called "Getting Connected" which gives detail at a user level on infrared communication. The document also provides the specifications for the Pocket PC.

How Pocket PC "Talks" with a Cell Phone. Arne Hess. Club PocketPC Article (2000) [PKDOC2]
http://www.microsoft.com/mobile/pocketpc/columns/ppcomm.asp

Brief: Only contains a few paragraphs on Infrared. Of value only in the fact that it indicates a practical use and states that it is the “most convenient way to connect”.

Extended: Although the article was sparse on detail it did influence the decision on the use of infrared for this project. Initially many communication mediums were being looked at but as the article pointed out, “all Pocket PCs have an infrared eye” and other mechanisms required additional hardware (cables or cards). So by using infrared any application developed could be used on all Pocket PCs.

12.4 *Infrared Literature*

A general comment on Infrared Data Association specifications. The specifications include state diagrams and state charts, which are ideal for Rose RealTime modelling. The reason that they are ideal is that the state machines can be defined within Rose RealTime exactly as they appear in the specification, code generated from them and runtime behaviour observed. However the ability to generate the code from the design and have the design (in this case state machines) exactly copy the specifications introduces some unique copyright issues. Fortunately the Infrared Data Association is approachable and will give consent as long as information from their specifications is acknowledged (this has been done for this project [COM4]).

Also to note, is that the specifications contain a much higher level of detail than an application designer would require. The literature reviewed for Windows CE indicated that the WinSock services where used to access the infrared communications, this hides most of the detail given in the IrDA specifications. The “IrDA Infrared Communications: An Overview” by Megowan, Suvak and Knutson [IRDOC2] distils most of the information and is recommended as the best source, from the literature reviewed, of high level IrDA information for an application designer.

The Infrared Data Association has two main standards IrDA Data and IrDA Control. IrDA Data is of interest to this project as it deals with data transmission. IrDA Control is aimed at low throughput remote control devices (keyboards, tv controls, mice) and therefore not looked at as part of this project and review.

Technical Summary of "IrDA Data" and "IrDA CONTROL". Infrared Data Association. (2000) [IRDOC1] **http://www.irda.org/standards/standards.asp**

Brief: As the title suggest as summary of both IrDA Data and Control specifications. Nice 5 page description, which puts everything in context.

Extended: The amount of IrDA specifications makes this a very useful document. It is recommended that anyone tackling the IrDA protocols keep a copy of this to maintain a top-level view of where each specification fits in the overall plan.

IrDA Infrared Communications: An Overview. Patrick Megowan, David Suvak, Charles Knutson. (1998) [IRDOC2] **http://www.irda.org/use/pubs/Overview.PDF**

Brief: This 21 page overview provides a good starting point for understanding the IrDA specifications. The information is also targeted at embedded systems.

Extended: The authors draw on experience not only from the Infrared Data Association but also their industry experience (Counterpoint Systems Foundry). Other overviews reference this paper and may even just present the same information without the Counterpoint name. The IrDA itself use this document as the sole document under their “Using IrDA” - “How it Works” website link [WEB3]. As there is effort required in understanding the IrDA protocol stack as a whole, this document is a must and should be read before any of the specifications.

Serial Infrared Physical Layer Specification, Version 1.3. Infrared Data Association (1998) [IRDOC3]

Brief: As the name of the specification suggests, this document describes the physical characteristics of an IrDA link. This information, although important, is probably not required from a programming perspective.

Extended: This document is written to describe to a hardware designer the physical specifications of a infrared link and communication nodes. From a programming perspective, there is information of value but it has to be searched for. Information contained which maybe of use are related to things which may impact the application usability e.g. spatial positioning of IR nodes. Although it has been stated that this is probably not required, it is still recommended as especially in embedded development, it is always important to understand the underlying hardware.

Serial Infrared Link Access Protocol (IrLAP), Version 1.1. Infrared Data Association (1996) [IRDOC4]

Brief: This document describes the Link Access Protocol, which is the layer above the Physical Layer. The information provided is important but is not easy to read. It is recommended that this document be skimmed if information is required.

Extended: Information in this specification is set out in an order that does not flow. Readers of this that do not have understanding of the specification will find this difficult to follow. It is recommended that the reader first understand the summarised version, (IrDA Infrared Communications: An Overview. Patrick Megowan, David Suvak, Charles Knutson) [IRDOC2], before attempting to read this. Once the reader understands the basics, the specification should be skimmed as it does contain information about performance of data links and other characteristics, which impact application development.

Link Management Protocol, Version 1.1. Infrared Data Association (1996) [IRDOC5]

Brief: This document describes the Link Management Protocol, which is the layer above the Link Access Protocol. Again the information provided is more than what an application designer would want but it is still recommended that this document be skimmed.

Extended: This is more readable than the Link Access Protocol but contains a lot of information, which would not be relevant to an application designer. There are important parts which should be noted but these are covered in the “IrDA Infrared Communications: An Overview” [IRDOC2]. It is recommended that the document be skimmed and the structure and content be mentally noted for future reference. This document contains information about the flow control limitations, multiplexing of infrared links and Information Access Services, which should be looked at more carefully.

'Tiny TP': A Flow-Control Mechanism for use with IrLMP, Version 1.1. Infrared Data Association (1996) [IRDOC6]

Brief: This document describes how flow control is added to the Link Management Protocol. Again the information is provided in the “IrDA Infrared Communications: An Overview” [IRDOC2].

Extended: This is a short document (23 pages) but still has too much detail for an application designer. The recommendation is to note that this document exists for future reference.

'IrCOMM': Serial and Parallel Port Emulation over IR (Wire Replacement), Version 1.0. Infrared Data Association (1995) [IRDOC7]

Brief: IrComm is the specification to be only used for legacy applications. The specification itself recommends that IrComm should not be used.

Extended: This information would be of definite use for a legacy application but for this project this is not the case. The reasons for not using IrComm are due to mapping “wired” protocols to infrared. Due to the document not having relevance the overview was all that was read.

IrDA Object Exchange Protocol, IrOBEX, Version 1.2. Infrared Data Association (1999) [IRDOC8]

Brief: This protocol allows communication of arbitrary data objects between applications. From an application design point of view this contains valuable information and also structured to be readable for application designer.

Extended: It should be first stated the one of the goals of this project is to apply Connexis to infrared communications. If this is successful then the IrOBEX protocol would not be required as the protocols between applications would be defined in the Rose RealTime toolset. However, the IrOBEX protocol is still worth knowing about and this specification provides a lot of good information. There are not only specifications but also examples of how the protocol has been applied and test guidelines. The information would also be valuable as a complementary protocol to Connexis or for communication with no Rose RealTime applications.

Minimal IrDA Protocol Implementation (IrDA Lite), Version 1.0. Infrared Data Association (1996) [IRDOC9]

Brief: This is really an Appendix to the Serial Infrared Link Access Protocol and Link Management Protocol specifications. Without these, this document will not make sense (e.g. acronyms not defined). The specification gives ways to cut down on IrDA functionality but still maintain some compatibility with other IrDA devices.

Extended: This specification would only be of use if the reader, as an application designer, needed information on the minimal functionality an IrDA Lite device had. It states “This specification is intended to be a companion document to the IrDA IrLAP and IrLMP specifications”, this is very true as it launches into descriptions of NDM and NRM without giving any indication to what NDM and NRM are. To be able to read this document the specifications mentioned will be required. This specification should be noted as existing but in most cases would not need to be referenced.

LAN Access Extensions for Link, Management Protocol, IrLAN, Version 1.0. Infrared Data Association (1997) [IRDOC10]

Brief: IrLAN is not supported on Windows CE and is designed for higher speed infrared connections. In the future when high-speed links for Pocket PCs may be standard this would provide useful information.

Extended: The IrLAN specification details “Ethernet style” infrared connections over high-speed 1.15 and 4 Mbps links. Unfortunately the Pocket PC is limited to 115 kbs and therefore this specification does not apply.

Programming With Infrared Sockets - Whitepaper, Prasanna .V, California Software Laboratories [IRDOC11]
http://www.cswl.com/whiteppr/white/infrared.html (1998)

Brief: A good whitepaper but the information is dated. It is worth noting that it is there as it does cover some things like “raw IR” in more detail than elsewhere. As far as programming Infrared, based on the content of this document, it appears that any information pre 1999 will be outdated.

Extended: Since this was written, infrared sockets are now accessible from Windows Sockets. With the new support for infrared, any previous methods should be avoided to maintain code portability to future releases. So indirectly, this whitepaper indicates the “use by” date of infrared programming literature with respect to Windows. The information is still valuable and should be noted, but for current programming practices it cannot be relied on.

Connecting Windows and Non-Windows Devices with IrDA. Mike Zintel
http://www.irda.org/design/WindowsNonWindowsIrda.PDF (2000) [IRDOC12]

Brief: Very good information to have but there are accuracy issues. This article gives a run down on IrDA from an application programmer’s perspective and is recommended to read.

Extended: Lots of useful information and easy to read. Unfortunately the information is inaccurate at some points, which brings into question the accuracy of the entire document. For example, it states “Information Advertising Service (IAS)” which is incorrect, IAS stands for “Information Access Service” as defined in the Link Management Protocol reference [IRDOC5] (in both this review and the article). Although not totally accurate, this article is well worth reading and will impart valuable information to an application designer.

12.5 *Web Resources*

Rational Rose RealTime (Rational) [WEB1]
http://www.rational.com/products/rosert/

This section of the Rational website provides an overview of the capabilities of Rose RealTime and also gives access (via the Support link) to "Tech Notes" which give detailed user information on a variety of topics. The information is intended for prospective customers (overview) and current customers that are looking for a solution to a question or usage problem (Tech Notes).

The Rational Edge (Rational) [WEB2]
www.therationaledge.com

This website is a Rational e-zine. This contains articles from Rose RealTime experts from within Rational describing aspects of RealTime development and using Rose RealTime for developing software. This information, although not directly useful for IrDA development, does provide insight to the issues related to software development using Rose RealTime.

The Infrared Data Association [WEB3]
www.irda.org

This website is an essential resource for finding information on IrDA. The site contains IrDA specifications freely available for download. Also provided is information for developers and users, such as guidelines. According to their resume, the "IrDA is an International Organization that creates and promotes interoperable, low cost infrared data interconnection standards that support a walk-up, point-to-point user model." and the site content confirms that statement.

Pocket PC (Microsoft) [WEB4]
www.microsoft.com/mobile/pocketpc/

This section of the Microsoft website provides good information on the Pocket PC and has several articles on using IrDA and connectivity options. Though not too in-depth it still provides valuable information. The articles are well structured and give the background of the authors.

Windows Embedded Developer Center (Microsoft) [WEB5]
http://msdn.microsoft.com/embedded/

This provides on-line information in the form of the MSDN (Microsoft Developer Network) online library. This site has a very large pool of information and should also be considered an essential resource for in-depth Windows CE information.

Windows CE (Microsoft) [WEB6]
www.microsoft.com/windows/embedded/ce/

This section of the Microsoft website provides "infomercial" style information. Of limited value apart from the ability to download Visual embedded Tools (the development environment for Windows CE 3.0). The site is aimed at selling Windows CE rather than providing information about it.

IPAQ Pocket PC (Compaq) [WEB7]
www.compaq.com/products/handhelds/pocketpc/

This website provides basic information about the IPAQ Pocket PC. It also lists the compatible connectivity options and links the vendor's sites. Although it has limited content, it provides easy to get to and uncluttered information.

Windows CEcity [WEB8]
www.wincecity.com

This website is active location for Window CE and Pocket PC information. Lots of information available but has to be searched for. The discussion groups are also very active and informative.

Appendix A - Experiences with Action Research

Action Research Methodology

The research goals are to apply the Rose RealTime to a particular problem and, in the process of doing this, add to the knowledge how to apply Rose RealTime to a problem. The end result will be a specific problem solved (or reasons why it wasn't solved) and lessons learned, which could be specific to the problem or generic.

Action Research was chosen as the research methodology as applies an iterative cycle to the research process. The iterative nature of Action Research also lends itself to Rose RealTime, which promotes iterative development as the development methodology.

As stated in "An Overview of Common Research Approaches" (UTS IT Research Methods Study Guide, John Hughes) " [UN1] A significant feature of action research is that it operates in cycles or involves a spiral process - action research proceeds by doing and by making mistakes in a self-reflective spiral of planning, acting, observing, reflecting, planning etc. This spiral is one in which feedback is going on in many ways at once and the research adapts to new influences and is modified and changed by events - the complex and creative business of real life can be accommodated."

In this the case of this project, Action Research provides the most effective means of extracting knowledge and making the research methodology adaptive enough to generate valid results. The methodology does this by both allowing a qualitative aspect (the researcher makes value judgements as a user) and allowing the process to adapt to a complex real world situation.

Action Research addresses the following real world complex problems:

Unknown behaviour – The researcher has no control over the internal workings of the items under research. Behaviour of the Operating System, Toolset, Compiler can only be found through practice. Action Research allows the behaviour to be discovered and actions taken based on what was discovered.

User perspective is required – The researcher needs to play a dual role of researcher and participant. During the process, the researcher needs to make qualitative decisions based on a users aspect. For example, infrared may be possible to use for development but there is no point in using it if there is some human aspect that makes it unusable a real world situation.

External influences – There are external influences, which may need to be reacted to. To maintain validity the research cannot exist in a "bubble", the results will need to apply to the current situation as it evolves. For example, it is expected that some or all of the products under investigation will change over the lifetime of the project as a result of bug fixes and upgrades. If these are not absorbed into the research, the final results could be invalid because they can't be applied to the current situation.

The above text is taken directly from my project proposal. The use of this unconventional approach to the research problem generated some debate for the project proposal and final project presentation. Although, I do not claim to be an authority in the area of research methodologies, a summary of my experiences are reported here due to the interest generated:

- When thinking about applying Action Research for a similar project be prepared to state a case for it's use and be able to demonstrate understanding of the mythology compared others.
- I believe the use of Action Research for this project was possible because the following attributes of the problem existed:
  - A User's perspective was required. Rose RealTime is really a "hard coded process" designed to help users develop complex computer systems more effectively than other methods by using human terms. If the Rose RealTime cannot do this effectively for a human, then the answer to "can it be done", is "No" although all the technical aspects may be possible.
  - The researcher "was" a user and therefore value judgements would be accurate. I have 13 years experience with embedded technology and for the past 2 years have been responsible for helping other users with Rose RealTime problems. I could not give accurate value judgements as a user if I did not have this background (e.g. I could not apply Action Research to a project based on developing Cobol as I have no experience with Cobol).
  - Small user base. As the project added new functionality, the only user was myself and therefore my value judgements were accurate across all users. In the future as more users adapt this information, better approaches will be developed but at the time this report is published it is valid. I could not apply Action Research as I have if there was a large user base, I could not make a value judgement on behalf of thousands of users. Applying Action Research is still possible but would require collaborating with a number of users in a real environment and possibly undertaking a survey to confirm that the result was valid for larger number of users.
  - Small amount of known knowledge of the subject. One of the problems with Action Research is validity (i.e. being able to guarantee the findings are correct.), this is because the researcher is not in control of the influences on the environment. So Action Research uses the opposite approach to enable validity and controls the influences on the researcher, this is done by specifying a framework. In the case of this research, because the body of literature was quite precise and limited, someone with access to the literature referenced and this report can validate it's findings. You could not use Action Research if the results cannot be validated against a definable body of knowledge.
- I found that micro documenting the findings, in this case as Technical Notes, advantageous. Having the knowledge partitioned as standalone units, made more sense. They could be more easily adapted, validated or disposed of with out affecting the entire knowledge base.
- Keeping a diary was also important, the environment changed and the direction of research changed due to outside influences. Without keeping a diary I would be impossible to keep a track of why the research took a different direction from the initial planned direction. This also aids in maintaining references to communication.

Conclusion: Although a case for using Action Research has to be argued. The use of it benefited my research greatly. Had I chosen straight experimental research, my research may have stopped at the simple test application, with a statement “Yes, Rose RealTime can be used for Infrared Development”. However, addressing it using a more human research methodology allowed me to uncover issues preventing the statement being valid from an end user’s point of view.

Using this report, and the framework of the body of knowledge referenced from this report, the reader should be able to recreate the findings documented.

Appendix B – Technical Note #18596

Document ID:
18596

TITLE: Porting RoseRT C++ TargetRTS to Windows CE 3.0 - ARM processor
PRODUCT: Rational Rose RealTime C++ 6.3
OS: Host platform : NT, Win2000
DEFECT #:
PATCH #:
REFERENCES: Rational Rose RealTime C++ Porting Guide, MicrosofteMbedded Visual C++ 3.0 On-Line Help
CREATED: 26-Jun-2001
REVISED:

QUESTION:
How do I port the Rose RealTime TargetRTS library to Windows CE 3.0 - ARM processor (e.g. iPAQ Pocket PC)?

ANSWER:
Rose RealTime provides "support" for Windows CE 3.0 - SH3 processor. What this means is that Rational Technical Support can attend to any problems with this port. It is however, an easy process to port the SH3 RTS library to a different processor, though Rational Technical Support will NOT be able to provide support for this (unless the same problem occurs for the SH3 processor). The porting process is a customization and also not supported. This information is provided for customer's convenience.

The porting process involves customizing the Windows CE 3.0 - SH3 port to the ARM processor. The steps will be explained as per the "Phases of a Port" outlined in the "C++ Porting Guide".

1) Performing pre-port steps (see Before starting a port).

Before doing any port consult the "C++ Porting Guide" to see the issues that need to be addressed. For this port, the minimum that needs to be done before porting is:

- Confirm that Microsoft eMbedded Visual C++ 3.0 is installed and working for both SH3 and ARM targets by compiling a simple application.
- Confirm that a Rose RT compilation is possible for WINCE300T.sh3-eMVisualC++-3.0 (ensure that all system environment variables are set to locate the compiler and the processor specific libraries)

2) Naming the platform (see Choose a configuration name).

By convention the name of the platform should be called:

WINCE300T.arm-eMVisualC++-3.0

3) Defining the setup script (see Create a setup script).

As the SH3 port already exists, the setup script can be derived from that port by the following process:

- Make a copy of the $RTS_HOME/config/ WINCE300T.sh3-eMVisualC++-3.0 directory and save it as $RTS_HOME/config/WINCE300T.arm-eMVisualC++-3.0
- In the new directory alter the contents of the setup.pl script to replace the SH3 specific declarations to be ARM (below)

```
# my $target       = 'sh3';
my $target       = 'arm';

# $ENV{'CC'}        = 'shcl.exe';
# $ENV{'TARGETCPU'} = 'SH3';
$ENV{'CC'}        = 'clarm.exe';
$ENV{'TARGETCPU'} = 'ARM';

# $supported    = 'Yes';
$supported    = 'Custom';
```

Note: $RTS_HOME = $ROSERT_HOME/C++/TargetRTS

4) Defining the platform-specific makefiles (see TargetRTS makefiles).

As the SH3 port already exists, the makefile script can be derived also:

- Copy the $RTS_HOME/libset/sh3-eMVisualC++-3.0 directory to $RTS_HOME/libset/arm-eMVisualC++-3.0
- Alter the contents of the libset.mk script to replace the SH3 specific declarations to be ARM (below)

```
# CC            = shcl
CC            = clarm

# LIBSETCCFLAGS = /nologo /GF /MD /TP  note:/MD not applicable
LIBSETCCFLAGS = /nologo /GF /TP

# LIBSETLDFLAGS = /nologo /subsystem:windowsce,3.00 /MACHINE:SH3
LIBSETLDFLAGS = /nologo /subsystem:windowsce,3.00 /MACHINE:ARM
```

5) Defining the platform-specific header files (see Porting the TargetRTS for C++).

Again the SH3 header files can be derived to ARM:

- Edit the RTLibSet.h in $RTS_HOME/libset/arm-eMVisualC++-3.0 (below)

```
/* old
#define SHx
#define SH3
#define _SH3_
#define WIN32_PLATFORM_PSPC
*/
```

#define ARM
#define WIN32_PLATFORM_PSPC

6) Defining the platform-specific implementation of TargetRTS features (see Preprocessor definitions).

This step is not required for this port as the SH3 implementation is suitable for the ARM.

7) Building the new TargetRTS and fixing compile and link problems (see Building the new TargetRTS).

From the $RTS_HOME/src build the Target RTS:

```
NMAKE /nologo /f Makefile CONFIG=WINCE300T.arm-eMVisualC++-3.0
```

The TargetRTS should compile successfully (there will be warnings about "unreferenced formal parameter" from the MFC code, these can be ignored).

8) Testing the new TargetRTS using test model updates (see Testing the TargetRTS port).

To test the TargetRTS, start up Rose RT, take the SH3 test that was successfully compiled in phase 1and change the component specification to select the new TargetRTS configuration. The component should now compile for the ARM processor.

It should be noted that there are many files that are common across the processors and Windows (if Visual Studio is also installed). Therefore the environment MUST be set for the particular processor to avoid linking against libraries of different machine types or versions of MFC.

For more information contact Rational Technical Support.

Appendix C – Technical Note #21927

Document ID:
21927

TITLE:	Porting Connexis Libraries to Windows CE 3.0 - ARM processor
PRODUCT:	Rational Rose RealTime C++ 2001a.04.00
OS:	Host platform : Win2000 Target platform: WinCE 3.0/Pocket PC
DEFECT #:	N/A
PATCH #:	N/A
REFERENCES:	Connexis User's Guide, Technical Note 18596 - PortingRoseRT C++ TargetRTS to Windows CE 3.0 - ARM processor
CREATED:	08-OCT-2001
REVISED:	10-OCT-2001

QUESTION:

How do I create a Connexis Library for Windows CE 3.0 - ARM processor?

ANSWER:

If you have created a Target RTS port to Windows CE - Arm Processor (see Technical Note 18596 - Porting RoseRT C++ TargetRTS to Windows CE 3.0 - ARM processor) you may want to go a step further and port the Connexis libraries as well.

Instructions on how to do this are described in the on-line documentation on "Porting the DCS to a New Target Configuration" from the Connexis User's Guide. Please reference those while following the steps outlined below.

Note: The porting process is a customization and also not supported. This information is provided for customer's convenience and should only be undertaken by an experienced user.

To port the DCS to a new target configuration:

1. Create a TargetRTS library for the new target configuration.

- See TN 18596 -Porting RoseRT C++ TargetRTS to Windows CE 3.0 - ARM processor

2. Create DCS target specific header files for the new target configuration.

- These are already defined in the $RTS_HOME\target\WINCE300T directory as part of the sh3 install (the supported Windows CE target processor)

3. Load the DCS model.

- As with any customisations, first copy the DCS.rtmdl model found in the $CONNEXIS_HOME/Model directory, to a workarea (avoid spaces in the directory path) so that you always have the original.
- Load the copied model.

4. Create a new C++ library component for the new target configuration.

- Create a package named "WINCE300-arm-emVisualCpp-3-0" in the Component View.
- Duplicate the DCSlib_Std_* component in the "WINCE300-sh3-emVisualCpp-3-0" package and move the copy to your newly created "WINCE300-arm-emVisualCpp-3-0" package.

5. Configure and customize the C++ library component settings.

- Change the TargetConfiguration in the C++ Compilation Tab to "WINCE300T.ARM-eMVisualC++-3.0"
- RTD_CONNEXIS_BUILD should be changed from $(CNX_BUILD_NUM) to a user-defined integer e.g. RTD_CONNEXIS_BUILD=1. This constant's definition can be found in the component's "C++ Compilation" > "Compile Arguments" field.

6. Configure the DCS CDR encoding/decoding for the new target configuration.

- This is the same as for the sh3 component so you don't need to look at this.

7. Build the DCS library for the new target configuration.

- Right-click on the component and select "build". If you have your environment set correctly (e.g. include path etc), the build should be successful.
- Copy the newly generated DCS.lib to the Connexis lib directory, using a suitable storage directory name (e.g. $ROSERT_HOME\Connexis\C++\lib\WINCE300T.arm-eMVisualC++-3.0)

8. Test the new target configuration.

- Use the BasicTest.rtmdl supplied ($ROSERT_HOME \Connexis\C++\examples) and copy it to a workarea location.
- Following the naming conventions of the other tests, create a package "WINCE300-arm-emVisualCpp-3-0" package in the Component View.
- Duplicate the components from "WINCE300-sh3-emVisualCpp-3-0" package and move the new copies to the newly created "WINCE300-arm-emVisualCpp-3-0" package.
- Change the TargetConfiguration in the C++ Compilation Tab to "WINCE300T.ARM-eMVisualC++-3.0" for both TestClient and TestServer.

You should now be able to build and test these.

Note: You will need to set up the command line parameters appropriately in the Component Instance Specification.

-CNXep=9100 -CNXui=basicTestClient_41Instance
-scdm://10.192.3.81:9900 -obslisten=30076
-CNXep=9900 -CNXui=basicTestServer_41Instance -obslisten=30046
For more information, contact Rational Software Technical Support.

Appendix D – Technical Note #20458

Document ID:
20458

TITLE:	Target Observability to a Windows CE device using Serial Cable
PRODUCT:	Rational Rose RealTime C++ 2001a.04.00
OS:	Host platform : Win2000 Target platform: WinCE 3.0/Pocket PC
DEFECT #:	N/A
PATCH #:	N/A
REFERENCES:	Microsoft ActiveSync On-Line Documentation 3.5, TheWindows CE Technology Tutorial (Chris Muench/Addison Wesley/2000)
CREATED:	23-AUG-2001
REVISED:	10-SEP-2001

QUESTION:
Is it possible to get Target Observability of a Pocket PC working across a Serial cable?

ANSWER:
Yes.

Background:

Target Observability allows the user to observe the execution of a model on the target. Information from the target is passed to the toolset over a TCP/IP connection.

Normally communication is via an ethernet card, however, in some cases an ethernet card may not be possible. For example, in a Windows CE/Pocket PC, an ethernet card is extra hardware (at additional cost). Even if an ethernet card is available, it occupies an expansion slot (provided in an expansion pack) which may be required by another card (if for example a GPS application is being developed).

There are other connections, such as the Serial cable which should provide another way of TCP/IP communication with the target. The problem is that ActiveSync 3.5, which provides a link between the PC and the Pocket PC, does not allow outside TCP/IP communication.

Solution:

The solution is to bypass the ActiveSync default Serial cable communication and use the older style Remote Access Services (RAS). This requires a bit of setting up (hence the reason it is no longer the default).

Note: These steps assume that connection is normally done via a

Serial Cable and not via a USB (Universal Serial Bus) cable.

The steps are as follows:

1) Define a communications link between the PC and Pocket PC

- Control Panel -> Phone and Modem Options
- In the "Modems" tab, press "Add" and select "Don't detect my modem"
- Select "Communications cable between two computers" and then "Next"
- From the list of COM ports select the port where your Pocket PC will be connected.

2) Set communications link to the highest speed

The default setting for serial links is 19200, for Target Observability this will give poor performance. Fortunately the link can be set to 115200, which will give good performance.

- Select the "Communications cable between two computers" from above in Phone and Modem Options, Modems Tab
- Select Properties
- Set "Maximum Port Speed" to 115200

3) Disable the ActiveSync connection

Assuming that ActiveSync is running with it's default, the serial connection is already in use. This connection will need to be shut down and the new RAS connection established.

- From ActiveSync File -> Connection Settings... , uncheck "Allow serial cable or infrared connection to this COM port" then OK

ActiveSync will now display "Not Connected" and the Pocket PC will also disconnect.

4) Create a RAS connection

RAS enables the Host PC to provide an IP address to the Client in a similar fashion to dialing into a network. This will mean that the Pocket PC will connect to the Host via the Serial cable and have an IP address assigned.

- Control Panel -> Network and Dial-Up Connections
- Select "Make New Connection" then "Next"
- Select "Accept Incoming Connections"
- For "Devices for Incoming Connections" choose the communications link from step 1
- The default "Do not allow virtual private connections" can be left as is
- For "Allowed Users" select "guest"
- TCP/IP connection properties can be left as is or set-up to suit (e.g. defining a static IP address to use)
- Finish

There will now be a Connection called "Incoming Connections".

5) The default setting for the Serial connection rate on the Pocket PC will now need to be changed.

- Go to Start -> Settings -> Connections Tab
- Select the "PC" connection
- Set the default to '115200 default and then "OK"
- Switch the device on and off (it should connect automatically)

The Pocket PC should now be connected and a connection called "<Unauthenticated User>" should be visible in the "Network and Dial-Up Connections" folder. ActiveSync will also show status as "Connected".

6) Test out the connection

Get the required information:
- From "Network and Dial-Up Connections" folder, double click on the Pocket PC connection "<Unauthenticated User>".
- In the "Details" tab, find the Client IP address (e.g. 192.229.52.109)

- In the Rose RT Component Instance Specification note the obslisten port (e.g. -obslisten=30819)

Set-up the Processor information in Rose RT
- In the Deployment View -> Processor Specification Detail Tab, set the IP "Address" to the Pocket PC (e.g. 192.229.52.109)

Set-up the Target Application to run with the correct command parameters (TechNote 19967)
- Assuming that the executable is placed in "\My Documents\RoseRT_Build\" create a link to the executable with the obslisten parameter.
e.g. 59#"\My Documents\RoseRT_Build\HiCapsule.EXE" -obslisten=30819

Run the Application and connect Rose RT to it.
- On the Pocket PC select the link above to start the application
- On Rose RT select "Attach Target" from the Component Instance to connect.

Rose RT will now have the RTS Browser up and allow you to have Target Observability. The Pocket PC will be displaying the standard "rosert: observability listening" text. The application can be started and observed through the toolset.

Note: The above gives the ability to get Target Observability working manually. Although this does not give full Target Observability support, ActiveSync services are still available, so with further configuration it should be possible to download, run and shutdown the model all from within the Toolset.

For more information contact Rational Technical Support.

Appendix E – Technical Note #19967

Document ID:
19967

TITLE:	Using links to start Windows CE applications with Rose RT command options
PRODUCT:	Rational Rose RealTime C++ 2001a.04.00
OS:	Target platform: WinCE 3.0/Pocket PC
DEFECT #:	N/A
PATCH #:	N/A
REFERENCES:	N/A
CREATED:	6-AUG-2001
REVISED:	10-SEP-2001

QUESTION:

How can I easily start applications on a Windows CE device with Rose RealTime command line options?

ANSWER:

It is possible to use links to set-up command line options on Windows CE. This will allow the application to be started with a simple tap on the screen.

The steps to do this are as follows:

1) Copy the executable to the Pocket PC directory (e.g. My Documents/RoseRT_Build)

2) Using the ActiveSync Explorer, right click on the executable and select "Create Shortcut".

3) There will now be a shortcut which can be renamed appropriately (e.g. "RoseRT Hi" or "RoseRT Hi TO")

4) Drag and drop the short cut to the PC desktop and then open it with a text editor (it will have a .lnk extension).

5) The text will look like the following:

42#"\My Documents\RoseRT_Build\HiCapsule.EXE"

The "42#" is the number of characters in the line excluding the "42#". This can now be changed to include the command line options.

e.g.
To run an executable without Target Observability (i.e. start straight away)

61#"\My Documents\RoseRT_Build\HiCapsule.EXE" -URTS_DEBUG=quit

To run with Target Observability (i.e. wait for Rose RT to connect to it)

59#"\My Documents\RoseRT_Build\HiCapsule.EXE" -obslisten=30819

Note: The leading number has to be correct. Quotes are required for paths with spaces and these are included in the character count.

7) Save the link back to the Pocket PC. This can now be used to start the Rose RT application.

For more information contact Rational Technical Support.

Appendix F - Technical Note #23788

Document ID:
23788

TITLE: Porting RoseRT C++ TargetRTS to Windows CE 4.0 - ARM processor
PRODUCT: Rational Rose RealTime C++ 2002.05.00
OS: Host platform : NT, Win2000
DEFECT #:
PATCH #:
REFERENCES: Rational Rose RealTime C++ Porting Guide, MicrosofteMbedded Visual C++ 4.0 Beta On-Line Help
CREATED: 30-NOV-2001
REVISED:

QUESTION:

How do I port the Rose RealTime TargetRTS library to Windows CE 4.0 - ARM processor (e.g. iPAQ Pocket PC 2002)?

Note: Windows CE 4.0 is also refered to as WinCE.Net or Talisker. Pocket PC 2002. At the time of creating this TechNote, Visual eMbedded Tools 4.0 was in beta release and the Pocket PC 2002 firmware upgrade was not available. The code compiles but has not been tested on a real target and this should be revised when the production release components are available. This TechNote should remain internal until that point.

ANSWER:

Rose RealTime provides "support" for Windows CE 3.0 - SH3 processor. What this means is that Rational Technical Support can attend to any problems with this port. It is however, an easy process to port the SH3 RTS library to a different processor, though Rational Technical Support will NOT be able to provide support for this (unless the same problem occurs for the SH3 processor). The porting process is a customization and also not supported. This information is provided for customer's convenience.

The porting process involves customizing the Windows CE 3.0 - SH3 port to the ARM processor. The steps will be explained as per the "Phases of a Port" outlined in the "C++ Porting Guide".

1) Performing pre-port steps (see Before starting a port).

Before doing any port consult the "C++ Porting Guide" to see the issues that need to be addressed. For this port, the minimum that needs to be done before porting is:

- Confirm that Microsoft eMbedded Visual C++ 4.0 is installed and working for both SH3 and ARM targets by compiling a simple application.
- Confirm that a Rose RT compilation is possible for WINCE300T.sh3-eMVisualC++-3.0 (ensure that all system environment variables are set to locate the compiler and the processor specific libraries)

2) Naming the platform (see Choose a configuration name).

By convention the name of the platform should be called:

3) Defining the setup script (see Create a setup script).

As the SH3 port already exists, the setup script can be derived from that port by the following process:

- Make a copy of the $RTS_HOME/config/ WINCE300T.sh3-eMVisualC++-3.0 directory and save it as $RTS_HOME/config/WINCE400T.arm-eMVisualC++-4.0
- In the new directory alter the contents of the setup.pl script to replace the SH3 specific declarations to be ARM (below)

```
# $sdkKey = 'HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\CEStudio\\3.0\\Setup';
$sdkKey = 'HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\CEStudio\\4.0\\Setup';

# $OSVERSION = 'WCE300';
# $PLATFORM  = 'ms pocket pc';
$OSVERSION = 'WCE400';
$PLATFORM  = 'Pocket PC 2002';

# my $target       = 'sh3';
my $target       = 'arm';

# $ENV{'CC'}        = 'shcl.exe';
# $ENV{'TARGETCPU'} = 'SH3';
$ENV{'CC'}        = 'clarm.exe';
$ENV{'TARGETCPU'} = 'ARM';

# $supported    = 'Yes';
$supported    = 'Custom';
```

Note: $RTS_HOME = $ROSERT_HOME/C++/TargetRTS

4) Defining the platform-specific makefiles (see TargetRTS makefiles).

As the SH3 port already exists, the makefile script can be derived also:

- Copy the $RTS_HOME/libset/sh3-eMVisualC++-3.0 directory to $RTS_HOME/libset/arm-eMVisualC++-4.0
- Alter the contents of the libset.mk script to replace the SH3 specific declarations to be ARM (below)

```
# CC           = shcl
CC           = clarm

# LIBSETCCFLAGS = /nologo /GF /MD /TP  note:/MD not applicable
LIBSETCCFLAGS = /nologo /GF /TP

# LIBSETLDFLAGS = /nologo /subsystem:windowsce,3.00 /MACHINE:SH3
LIBSETLDFLAGS = /nologo /subsystem:windowsce,4.00 /MACHINE:ARM
```

5) Defining the platform-specific header files (see Porting the TargetRTS for C++).

Again the SH3 header files can be derived to ARM:

- Edit the RTLibSet.h in $RTS_HOME/libset/arm-eMVisualC++-4.0 (below)

```
/* old
#define SHx
#define SH3
#define _SH3_
*/

#define ARM
```

6) Defining the platform-specific implementation of TargetRTS features (see Preprocessor definitions).

Copy the $RTS_HOME/target/WINCE300T directory and name it $RTS_HOME/target/WINCE400T.
Change the following:

In RTTarget.h

```
// #define UNDER_CE   300
#define UNDER_CE   400

//#define _WIN32_WCE 300
#define _WIN32_WCE 400

//#define TARGET_WINCE 300
#define TARGET_WINCE 400
```

7) Building the new TargetRTS and fixing compile and link problems (see Building the new TargetRTS).

From the $RTS_HOME/src build the Target RTS:

```
NMAKE /nologo /f Makefile CONFIG=WINCE400T.arm-eMVisualC++-4.0
```

The TargetRTS should compile successfully (there will be warnings about "unreferenced formal parameter" from the MFC code, these can be ignored).

Note: I found I had to physically move the "C:\Windows CE Tools\wce400" directory to "C:\Windows CE Tools\eVC4\wce400\". I am not sure if this is the correct approach but it worked. If this is the final location in the official release of eMbedded Tools then the setup.pl (in step 3) should be changed to point to the correct location.

8) Testing the new TargetRTS using test model updates (see Testing the TargetRTS port).

To test the TargetRTS, start up Rose RT, take the SH3 test that was successfully compiled in phase 1and change the component specification to select the new TargetRTS configuration. The component should now compile for the ARM processor.

It should be noted that there are many files that are common across the processors and Windows (if Visual Studio is also installed). Therefore the environment MUST be set for the particular processor to avoid linking against libraries of different machine types or versions of MFC.

For more information contact Rational Technical Support.