HIAPER Cloud Radar

System Software Overview

1. Introduction

The HIAPER Cloud Radar (HCR) is a compact W band weather radar designed for both ground and aircraft operations. The first phase of the development is a prototype proof of concept, mounted in a modified sea container, with fixed vertical pointing. The second phase will add airborne capability, by mounting the radar in an aircraft under-wing pod. Fixed but selectable pointing will be implemented in the airborne version, with the possible later addition of scanning capability. The implementation of airborne system will require integration of supporting observations of aircraft attitude, position, etc.

The HCR system software covers a broad range of scales, from embedded firmware through Web applications. All computers will be running CentOS Linux. CentOS is a free clone of Red Hat Enterprise Linux, and thus benefits from the stability and security of the commercial product.

The computing parts of HCR are divided into two computer systems: the data acquisition/signal processing module, and the data system. The terminology has evolved such that the first is referred to as the **DRX** (for digital receiver), and the second is known as the **Archiver**. Each of these performs many more functions than are indicated by their respective names. The two systems are nodes on a private HCR ethernet network. This network can be connected to other networks via a router.

There are a few basic tenets that are followed in the HCR software design:

- The system is built as a collection of cooperating processes. These processes utilize a common data distribution scheme for sharing real-time data streams and products. Likewise, a common interprocess communication method will be used for command and status signaling between the processes. Because network protocols are used for data distribution and system control, for the most part the HCR processes can be run on either the DRX or the Archiver, or even another computer for that matter. Of course if a process needs access to specific hardware, such as the radar transceiver card, it must run on the machine hosting that card.
- The data streams carry along all necessary system configuration information required by the downstream processes. This avoids having to create a centralized system configuration server which would have to account for time lags in the data streams. For instance, radar configuration parameters such as the pulse width, system gains, etc. are included with the baseband I/Q data. The downstream moments generator will then have all of the information required in order to calculate the calibrated radar products.
- The system is "data driven". The computational processes are assembled in a pipeline, with one process feeding data to the following process. Note that a process may have multiple input and output streams. Each process behaves as an operator, and the production of an output sample is strictly a function of the input data. This approach eliminates the need for complicated interprocess synchronization schemes. All of the standard processes will be launched at boot time, and will be ready to process data as soon as it appears on the inputs. This will simplify the overall system control.

2. Subsystems

The functions of the HCR DRX and Archiver are summarized in Table 1: Software Responsibilities.

System	Functions
DRX	Radar IF sampling
	Digital down conversion
	Transmit pulse shaping
	Generation of timing signals for other hardware components
	Low level signal processing
	Collection of housekeeping data
	Merging of baseband IQ time series and housekeeping data
	Publication of the baseband merged time series
Archiver	Computation and publication of moments and other derived products
	Archiving of baseband data and generated products
	User interface for system control
	Engineering displays
	Scientific displays

Table 1: Software Responsibilities

2.1 DRX

The DRX is based on a Compact PCI (cPCI) backplane which hosts the following cPCI cards:

- A CPU card with the system solid state disk drive.
- A Pentek P7142 PMC digital transceiver card, mounted on a cPCI carrier.
- A general purpose digital I/O card
- Other I/O cards (ARINC, …?)

Compact PCI is a rugged solution intended for industrial applications in harsh environments. It was selected for the DRX because this component will be pod mounted on the aircraft, since it is necessary for the DRX to be collocated with the radar RF components. All interaction between the DRX and the Archiver will occur via a single ethernet link.

The processor in the DRX is a 1 GHz x86 CPU mounted on a cPCI card. A graphics interface, memory, solid state disk and network interfaces are provided on card. The CPU and and graphics capability provide only mid-range performance, but these are adequate for the few functions of the DRX. The DRX is runing the Linux operating system.

The primary activity of the DRX is to interface with and control the Pentek card. The Pentek samples the 156.25MHz radar IF input, and then performs down conversion and initial signal conditioning. The resulting I/Q data stream is transferred to a process running on the DRX. This process publishes the I/Q data stream to other subscribing processes. The Pentek card also generates the radar transmit pulse, as well as timing signals which are brought off card and used for synchronization of external hardware.

The P7142 has a PCI interface which can be accessed via a Linux driver. It provides multiple channels of A/D conversion for receiver sampling, and a single D/A channel which is used for the transmit pulse generation. Two FPGAs execute firmware which supports the PCI interface and signal processing for the received and transmitted signals. A standard VHDL firmware package is provided by Pentek, and this has been enhanced for HCR specific functions. These functions include digital down conversion, matched filter implementation, radar pulse timing, and transmit pulse timing.

The custom firmware which operates on the P7142 must work in conjunction with software on the DRX linux host. A library of support functions has been implemented which encapsulate communication between the host and the P7142. Host side applications for controlling the P7142 are built upon this library. The combination of the P7142, the VHDL firmware and the host side support library are named the Software Digital Down Converter (SD^{3}C). The SD^{3}C is not specific to the HCR, and is employed in other radar systems.

Another function of the DRX is to collect aircraft parameters and insert this housekeeping information (as metadata) into the I/Q data stream.

The full bandwidth I/Q base band data are published to the ethernet by the main DRX process.

2.2 Archiver

The Archiver is the main compute and storage node for the HCR. It is a rack mounted server, with powerful CPUs and graphics controllers, and significant disk storage. In the HCR airborne configuration, the Archiver will be located in the aircraft cabin. A monitor and keyboard will be available for both ground and airborne configurations.

The functions of the Archiver are:

- Primary host for the user interface.
- I/Q timer series generation
- Derived products generation.
- Derived products archiving.
- Primary display for real-time visualization
-

3. Software Technologies

This section provides a brief description of the software technologies that are utilized in the HCR. Some of these elements make use of commercial 3rd party tools, but for the most part they are supported by standard open source

3.1 VHDL

The P7142 firmware is written in the VHDL language. The Xilinx *Integrated Software Environment (ISE)* provides the development platform for VHDL editing and compiling of bitstreams which can be run on the P7142 FPGAs. ISE can run under both Linux and Windows. In general, ISE will not be deployed on the HCR system. Instead, it is used on the engineer's office system to create a firmware load which can then be written to the EEPROMS on the P7142. The SD^{3}C firmware contains revision numbers that can be read by the host software, and which is typically displayed by the processes which communicate with this card. Care should be taken to verify via the revision number that the firmware is up to date.

The SD^{3}C documentation describes the VHDL firmware architecture, as well as the host side supporting class libraries.

3.2 C++

Almost all of the HCR system software is written in C++. The standard GNU gcc compiler suite is used.

Scons is used as the build system for HCR software. EOL has developed a library of supporting modules based on the scons "tools" framework, and the HCR build scripts make heavy use of these tools. The EOL tools are accessed in the subversion source tree via an external reference to "site_scons".

3.3 OpenDDS

OpenDDS is an open source implementation of the Data Distribution Services (DDS) specification. DDS defines a network oriented data-centric publish and subscribe data communication system. Although DDS is often demonstrated in lower bandwidth examples, it is designed for and is capable of high data transfer rates.

In the DDS scheme, structured data types are defined via an "Interface Definition Language" (IDL). The data types can be very complex, and can carry large payloads. Applications access these datatypes as C++ structures. The applications can "publish" topics containing these data types; likewise applications can "subscribe" to a topic. The publishers and subscribers are not specifically aware of each other. A broker (named DCPSInfoRepo) is responsible for matching publishers and subscribers. Connections can come and go dynamically, and the DDS infrastructure handles this transparently within the applications.

The system has proven to be very flexible and robust. When it is necessary to change a data type, the IDL is modified, and the new structure elements will be available after recompilation to the DDS applications. The application code simply calls a publish method to transmit an item under a given topic, and the data are sent to any attached subscribers. Similarly, a subscriber can simple register for a published topic, and will receive notifications when new items are available.

3.4 Qt

The Nokia Qt package provides an extensive and elegant framework for building graphical applications. All of the HCR graphics applications are constructed using Qt. At the heart of the Qt design is the signal/slot mechanism, which provides a robust and rational system for implementing user interface callbacks and general dynamic signal dispatching between objects.

However, Qt also provides excellent programming abstractions which are generally useful and can be utilized outside of graphical applications. The Qt threading support is especially appealing, and has been used in non-graphical HCR applications, such as the hcrdrx process. Qt is pervasive in the HCR system software and so there is no significant drawback in employing it for a few non-graphics programs.

3.5 XML RPC

<describe XMLRPC here>

3.6 Linux

The CentOS Linux distribution is used on the DRX and the Archiver. In general, the operating system can be routinely updated without problem on the Archiver and DRX. However, if an updated kernel is installed on the DRX, the kernel must be patched and rebuilt with the *bigphysmem* extension, and the P7142 driver must be recompiled as well.

3.7 QtToolbox

<describe QtToolbox here>

4. Programs

The HCR applications are listed in Table 2: HCR Programs. A diagram of the network relations is presented in Illustration 1: HCR Process Overview.

Process	Deployed To	Function	Configuration File
hcrdrx	DRX	The digital transceiver controller	*~/.config/HcrDrx/HcrDrx.ini*
hcrprod	Archiver	The derived products generator	*~/.config/HcrProd/HcrProd.ini*
tsArchiver	Archiver	The archiver	*~/.config/TsArchiver/TsArchiver.ini*
hcrscope	DRX, Archiver, Other	An ascope display of I, Q and derived products	*~/.config/HcrScope/HcrScope.ini*
bscan	Archiver, Other	A bscan type display of derived products	*~/.config/Bscan/Bscan.ini*
hcrui	Archiver, Other	The user interface for HCR control and status monitoring	*~/.config/HcrUi/HcrUi.ini*
hcrEdit	Archiver, Other	The HCR configuration editor	*~/.config/hcredit/hcrEdit.ini*
DCPSInfoRepo	Archiver	Connects publishers and subscribers	

Table 2: HCR Programs

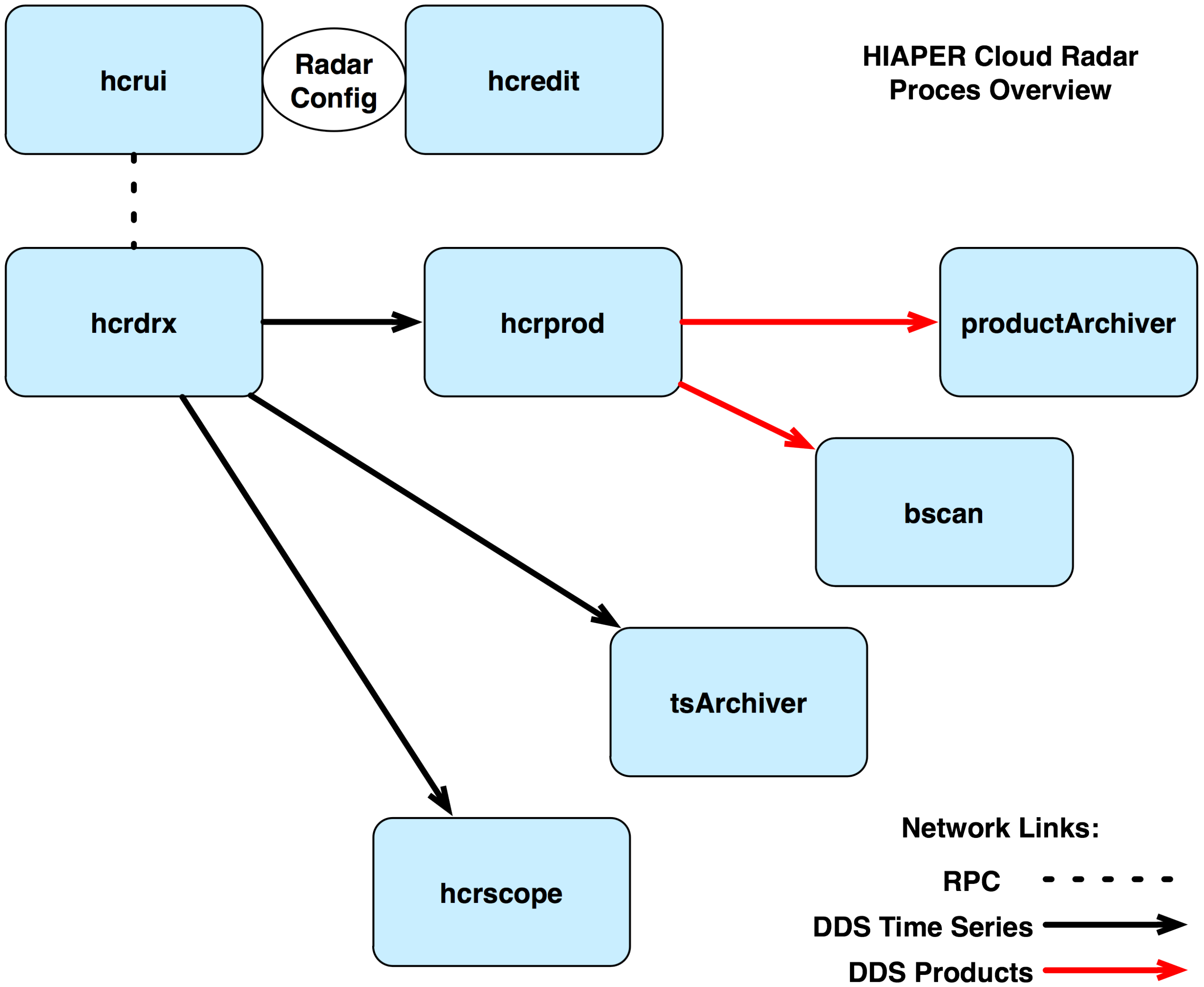

Illustration 1: HCR Process Overview

4.1 Program Configuration

The HCR processes follow a common configuration methodology. There are three mechanisms for defining the operational parameters of a given process.

The first method, which is implemented in all HCR processes, uses a text file containing key-value pairs. The QtToolbox configuration management class QtConfig provides the underlying support for this[1]. All users have individualized copies of the configuration for each program.These are stored in *.ini* files in directories below *~/.config/*, with one directory per program. For instance, the configuration for hcrdrx is kept in *~/.config/HcrDrx/HcrDrx.ini.* The purpose of the configuration file is to allow a program to run and be completely functional even without receiving commands from the user interface. The *.ini* file can be thought of as the default configuration for the program.

There are some subtle features of the configuration file scheme. One is that the programs will create

1. Other methods, such as XML, could be used here. QtConfig could easily be replaced by a different method simply by overloading the QtConfig functionality with a new class.

their own configuration files if they don't exist, using default values that are compiled in the source code. If the configuration for a program seems to be problematic, it is safe to erase the *.ini* file, and rerun the program to create a new default configuration. Also, some of the graphical programs will dynamically write characteristics to the configuration file, such as when a user changes a scale limit or a color table. This allows the user selections to be preserved between runs of the program.

The second configuration mechanism is the use of command line switches. These will support a subset of the parameters in the *.ini* file. Command line parameters override those that are specified in the *.ini* file. Note that some programs may not provide any command line switches.

The third configuration technique, employed by only a few HCR programs, occurs via the RPC mechanism. An RPC call specifying operational parameters can be made to a running process. For instance, this is used to reconfigure the number of gates in the running hcrdrx process. The default *.ini* configuration is not updated by an RPC configuration call.

4.2 DDS and DCPSInfoRepo

The DCPSInfoRepo process acts as the broker between DDS publishers and subscribers. It runs on the Archiver.

A publisher contacts DCPSInfoRepo in order to announce that it has a specific topic available for publication. Likewise, a subscriber contacts DCPSInfoRepo to request connection to the publisher of a specific topic.

There is no required sequence for announcing or requesting a DDS topic. DCPSInfoRepo keeps track of all publishers and subscribers, and when a match occurs, it instructs the two processes to make a direct connection and to begin transferring the topic. A publisher that has no subscribers will simply discard data that it is publishing. A subscriber without an attached publisher just waits until a publisher becomes available on that topic.

Configuration of DCPSInfoRepo and DDS clients is arcane, to say the least. Guidance can be found in the OpenDDS Developer's Guide, the OpenDDS examples, and the HCR applications. The standard method makes use of configuration files, although recent releases of OpenDDS have suggested that configuration can also be accomplished at runtime. HCR processes that make use of DDS share a common collection of DDS configuration files, which are located in the directory referenced by the DDS_CONF environment variable, which is nominally set to */opt/ddsconf.* Some HCR scripts make use of the DDS_CONF variable in order to locate the configuration files. For HCR programs which use the QtConfig configuration mechanism, the configuration file path is specified in the associated *.ini* file.

Within DDS applications, the DDS configuration file names are passed directly to DDS functions, which then presumably parse the files. There are other DDS parameters which are not specified in a file, but rather are passed in the nature of command line parameters.

4.3 hcrdrx

The hcrdrx program interfaces the P7142 card. The following functions are provided:

- Configuration of the the downconverter, upconverter and timers.
- Start and stop of downconversion.
- Start and stop of the transmit pulse generation.
- Publication of the baseband I/Q data via DDS.

- Status reporting.

The hcrdrx can be configured to use from one to four downconverter channels on the P7142, and one upconverter channel.

The program's *main()* will create one thread to handle the RPC communications, and one thread for each of the downconverters. After the threads are created, the main thread will enter a loop where it periodically prints program status and statistics. If hcrdrx has been configured for autostart, it will command the P7142 to start downconversion and the transmit pulse before entering the loop.

A single DDS publisher is created by *main(),* and shared with the downconverter threads. Each downconverter thread creates a DDS writer. The downconverter reads baseband I/Q samples from a channel specific P7142 device, such as */dev/pentek/p7142/0/dn/1BR* and performs consistency checking. The reads from the P7142 device are blocking. The I/Q data is then published via the DDS writer. Note within each downconverter thread the DDS infrastructure creates its own threads.

Each downconverter thread maintains statistics on the data bandwidth, synchronization errors, and DDS errors. This information is retrieved via an access function.

The RPC thread provides the interface for the user interface control of the process. Three RPC calls are supported:

- Stop: Instruct the downconverter threads to stop reading data and to wait for a start command. Disable the timers. This is the initial state for the downconverter threads.
- Start: Instruct the downconverter threads to reconfigure themselves with the supplied configuration parameters, and waiting for data. The timers are reconfigured and then enabled. The transmit pulse will be generated, and data will start to flow from the P7142 downconverter channels.
- Status: Return status information for each downconverter thread, plus other status.

<insert hcrdrx UML here>

4.4 hcrprod

<Text needed here>

4.5 tsArchiver

<Text needed here>

4.6 productArchiver

<Text needed here>

4.7 bscan

<Text needed here>

4.8 hcrscope

<Text needed here>

4.9 hcrui

The hcrui program provides a user interface for controlling and monitoring the HCR. It is graphical application built with the Nokia Qt windowing toolkit. The following functions are implemented:

- Control: Select operating configuration, stop and start the radar.
- Status: Real-time monitoring and display of status information from other HCR processes. Alerts are created when abnormal conditions are detected.
- Configuration: Displays the current operational parameters, and allows a configuration editor to be instantiated on command.
- Logging: HCR process status and Linux status can be logged, along with a history of user commands.
- Maintenance: Individual HCR processes can be forcibly stopped and started if necessary.

The Qt framework provides an excellent framework for this application. The threading support allows for separation of background functions from the user interface management. Qt allows the signal and slot facility to communicate safely between threads.

Hcrui will function as both an RPC server and client. As a client, it can issue calls to the other HCR processes, for sending commands or polling for status. As a server, it can receive RPC calls in order to accept messages from the other HCR processes.

Because visualization is provided by other HCR programs, the user interface only provides a few functions. The main action is for the user to select a named radar configuration, and then command the radar to start. This will cause the configuration and a start command to be transferred to the hcrdrx process. In keeping with the data driven design of the HCR software, the downstream process will automatically receive the data stream. The user can also command the radar to stop. This simply causes a stop command to be sent to the hcrdrx process.

The hcrui configuration is implemented with the standard QtConfig mechanism. This configuration contains information such as the IP names for the other HCR processes. It does not contain any information relating to the radar configuration.

The named radar configurations are managed by the hcredit program, but the configuration itself is available to hcrui. The radar configuration file will default to *~/.config/HcrRadar/*HcrRadar.ini, but others files can be selected within hcrui. The specifics of the radar configuration are covered in the discussion of hcredit.

Since the radar operating configuration is important to both hcrui and hcredit, there exist configuration management support classes that are used by both programs. These are described in hcredit.

<insert hcrui UML here>

4.10 hcredit

The hcredit program is a graphical editor used for managing HCR radar operating configurations. It provides the following functions:

- User interface: a clean and logical interface for editing radar configuration information.
- Persistence: infrastructure which allows configuration information to be saved, and shared among other applications.

- Validation: logic to verify that the specified radar parameters are self consistent.

Hcredit is a Qt application. It is based on the Qt Model/View architecture. The Qt documentation has extensive documentation on the Model/View architecture and the supporting Qt classes.

Hcredit uses the standard QtConfig scheme for the underlying storage. This fulfills the requirement that the radar configuration information is stored in a text file that can be viewed and manipulated with a standard text editor.

Note that there are two configurations associated with hcredit. *HcrEdit.ini* is used to specify characteristics of the editor itself. *HcrRadar.ini* contains the radar specific configuration. The two are kept distinct so that the radar configuration can be archived, version controlled, and copied among users and computers.

As discussed for hcrui, each user has a private version of the radar configuration, typically saved under *~/.config*. However, hcredit can select and operate on any specified *HcrRadar.ini* file.

In the Qt Model/View framework, an abstract representation of the data are represented in the model, and the view is used to render the data in a desired manner. The advantage of this design is that there can be multiple (and simultaneous) views of the same data. The model does not necessarily contain the actual data elements; it can be acting as an accessor to a separate data source. For hcredit, the data source is the QtConfig based *HcrRadar.ini* file. The Qt model within hcredit encapsulates access to this file.

<insert hcredit UML here>

Classes:

- QStandardItemModel::RadarModel: represents the radar configuration data in a tree structure.
- QTreeView::RadarTreeView: displays the complete radar configuration in a tree display
- QAbstractItemView::RadarUserView: displays the RadarModel in a more user friendly format.
- QStyledItemDelegate::RadarItemSimpleDelegate: A delegate which will perform simple validation on an edited item, using a validator supplied at creation time.
- QStyledItemDelegate::RadarItemModelDelegate: A delegate which will call the model in order to perform validation of the edited item. This method is used when an edited item needs to be validated against other items in the model.

5. Implementation Details

5.1 Command and Status

<topics:

RPC (XML or other)
Would be desirable to have an "IDL" scheme such as DDS provides
In general, RPC servers would run in a separate thread in those processes that are using it. The RPC thread communicates with other threads via? Qt slots would be nice.
A "hung" process should not be able to hang a calling process.
All process can be commanded to shut down.
Some processes can respond to a "flush" command.
>

5.2 Time

The system time on the DRX should be as accurate as possible. Although time accuracy on the Archiver is not particularly critical, it is best if it also keeps an accurate time.

HCR will have a NTP time server which is synchronized via GPS. The DRX and Archiver use NTP to maintain to system clock. The *crony* package is used in place of the standard Linux NTP package. *Crony* was designed for use in systems which are not powered on continuously, and testing has shown that it is able to discipline the Linux system clock much more rapidly than the standard package.

5.3 Boot Sequence

A number of scripts run at boot time in order to start the permanent HCR processes. The scripts located in */etc/init.d/init.d* are linked from */etc/rc5.d* so that they are enabled for runlevel 5.

/etc/init.d/ Script	Link	Machine	Function
pentek	*/etc/init.d/rc5.d/S98pentek*	DRX	Load the Pentek ptk7140 module
dds	*/etc/init.d/rc5.d/S98dds*	Archiver	Start the DCPSInfoRepo process
hcr	*/etc/init.d/rc5.d/S99hcr*	Archiver	Start these processes: • hcrdrx • hcrprod • tsArchiver • productArchiver
~hcr/runhcr		Archiver	This script is run by <?> when the hcr user logins in on the primary console, so that an operator interface is automatically created.

Table 3: HCR Startup Scripts

5.4 The Pentek Driver

<describe patching and building the Pentek driver, and include details about setting the buffer sizes>

6. Network Configuration

<insert network diagram showing DRX, Archiver, router and WAN>

7. Appendix

7.1 Metadata

Metadata Name	Used By
PRF	HCRProd, HCRArchiver
Gate	

Table 4: Meta Data

7.2 Radar Configuration File

<insert text from a typical *HcrRadar.ini* file>

7.3 Software Repository Structure

<describe the svn repository structure for HCR, including externals>