A Case Study: The development of an embedded realtime Tracking and Control Application for a tracking radar

Paul Thomson
Defence, Peace, Safety and Security
CSIR
Pretoria, South Africa
Email: pthomson@csir.co.za

Bader Almutery
National Program for Electronics, Communications and Photonics
King Abdulaziz City for Science and Technology
Riyadh, Kingdom of Saudi Arabia
Email: bmutery@kacst.edu.sa

Abstract—**This paper describes the process that was followed in the development of a real-time, embedded Tracking and Control Application (TCA) to be incorporated into a pulse-Doppler radar signal processing test and evaluation facility. The complete simulation and development process is described. Functional requirements are discussed, as well as algorithm selection, code development and system simulation. The porting of the TCA to the final target hardware and operating system, as well as the integration with the final system, are examined. An analysis of the success of the process followed is given and recommendations are made.**

Keywords- tracking, radar, C++, vxWorks real-time, embedded

I. INTRODUCTION

A tracking algorithm is implemented in a radar system so as to allow the radar antenna to be steered to keep the target of interest in the beam. Radar measurements are made of the target of interest. In the case of a pulse-Doppler radar, information regarding the range as well as velocity of the target, are provided by the radar signal processor. Furthermore, information regarding the detection of new targets is also provided. The function of the tracking algorithm is to receive and process this detection and tracking (or estimation) information in an intelligent way. The tracking algorithm should be able to utilize tracking information relating to a known target so as to predict the velocity and position of the target at a future point in time. The tracking algorithm should also be able to process detection information so as to establish tracks of new, previously unknown targets. The prediction of a targets future position may be used to steer the antenna so as to ensure that the target of interest is kept within the radar beam.

II. FUNCTIONAL REQUIREMENTS OF TCA

The TCA was required to perform the following main requirements:

- Software was required to be designed in such a way that it may track multiple targets
- Although the algorithm was designed handle multiple targets, the actual implementation was built to handle two targets, a primary and a secondary target
- The algorithm was required to support the inclusion of criteria to select specific waveforms, based on the properties of the target being tracked.
- Dynamic initiation and deletion of tracks, automatically, or under control of the radar operator.
- Detection of splitting and pop-up targets
- Control and steering of the radar antenna positioner
- Memory tracking
- Real-time communication and integration with the radar signal processor. The TCA receives detection and estimation information from the Radar Signal Processor (RSP).
- Integration with the Radar Operator Console (ROC). Information about detected and tracked targets must be relayed to the ROC for display purposes. Control commands from the ROC are to be received and processed.

III. CHOICE OF SOFTWARE DEVELOPMENT PROCESS

A. *Rational Unified Process*

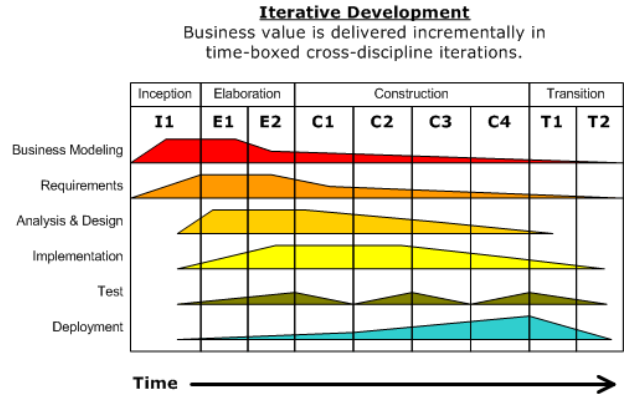

Figure 1: Illustration of Rational Unified Process

The Rational Unified Process (RUP) was chosen as the development process to be followed in the development of the TCA. The RUP is a use-case driven, architecture-centric, iterative and incremental process. Development using RUP is incremental, where each increment consists of each of the following workflows, to a lesser or greater extent:

- Business modeling
- Requirements definition
- Analysis and design
- Testing
- Deployment

The development increments are grouped into four different phases, namely:

- Inception
- Elaboration
- Construction
- Transition

The relationship between iterations, development workflows and development phases is shown in Fig. 1. Each of the development workflows may occur in each of the phases, but the weighting or effort spent on each workflow varies from phase to phase. For example, in the inception phase, which occurs at the beginning of the development process, no testing or deployment occurs, whereas in the transition phase, which occurs at the end of the development process, ideally no business modeling or requirement modeling occurs.

B. *Classes and use-cases*

As mentioned above, the RUP is a use-case driven, architecture-centric process. This means that the system requirements are captured by means of use-cases. A use-case is a "case of use" or a way in which the system may be used. A use-case is a textual description of a "case of use" of the system. The system requirements are captured by means of use-cases. Ideally, by the end of the elaboration phase, most of the system requirements should have been captured as use-cases. Use-cases describe the interaction between entities known as actors. In the case of the TCA, the actors would typically include entities such as the TCA, the RSP, the positioner and the operator. As the RUP makes use of the Unified Modeling Language (UML) as its design notation, the software architecture is defined by, amongst other possible diagrams, a class diagram. An example of a class diagram for the TCA is given in Fig. 2.

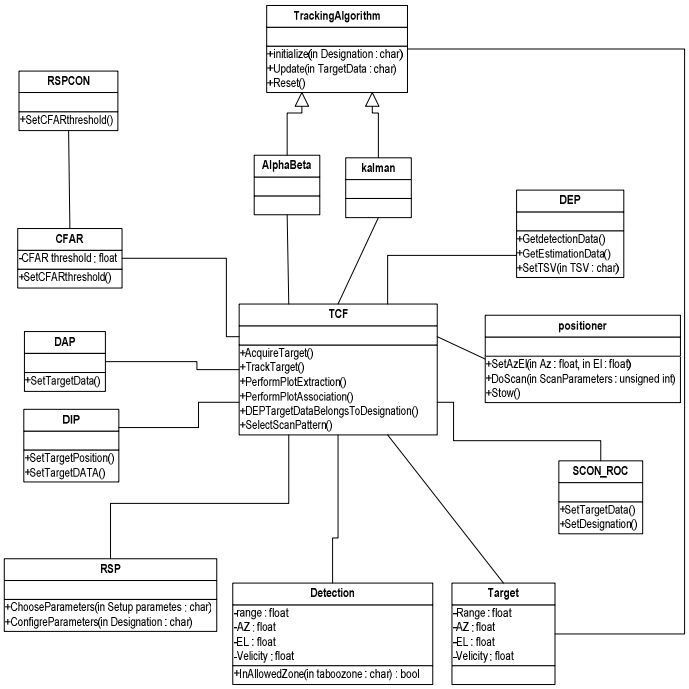

Figure 2: Example of class diagram as used for design of TCA application

C. *Implementation*

The implementation workflow occurs mostly in the construction iterations of the development process. If the preceding iterations have been performed thoroughly, the construction phase should proceed smoothly. The RUP makes provision for workflows other than implementation to be performed in the construction phase iterations, however, the weighting of these workflows should decrease as the construction phase progresses.

IV. SELECTION AND MODELING OF ALGORITHMS

A. *Alpha beta filter*

It was decided that for initial implementation of the TCA, that the alpha-beta filter should be used. The alpha-beta is a smoothing filter which is derived from the Kalman filter but whereas the Kalman filter recursively updates its coefficients, those of the alpha-beta filter are fixed. The advantage of this is

that the alpha-beta filter is less demanding of computational resources and is easier to implement. The alpha-beta filter is discussed in many books and papers including [1].

The alpha-beta filter is defined by the following equations:

$$x_s(n) = x_p(n) + \alpha(x_0(n) - x_p(n)) \quad \text{Eq. 1}$$

$$v_s(n) = v_s(n-1) + \frac{\beta}{T}(x_0(n) - x_p(n)) \quad \text{Eq. 2}$$

$$x_p(n) = x_s(n-1) + Tv_s(n-1) \quad \text{Eq. 3}$$

B. Modification to standard alpha-beta filter

The TCA was to be used in conjunction with a pulse-Doppler signal processor which measures the Doppler velocity of the target. As the standard alpha-beta filter does not make use rate of change of the tracked variable. E.g. velocity, it was decided to modify the standard alpha-beta filter to make use of the Doppler velocity measurement provided by the RSP. The standard alpha-beta filter equations were modified to make use of the measured target velocity in the prediction phase of the filter.

C. Implementation of tracking filter

The alpha-beta filter was implemented as a C++ class. The advantage of using an object oriented approach such as this is that multiple instances of the class may be instantiated without having to write any additional source code. The C++ class may also be reused easily in future applications.

V. SIMULATION OF RADAR ENVIRONMENT

As the development of the TCA occurred before the target hardware was available, it was necessary to implement a simulation environment in which to test the functionality of the TCA, in terms of communications as well computational accuracy. Initially the TCA was developed as a windows application, incorporating both the computational code as well as a Graphical User Interface (GUI). The GUI consisted of various controls as well as plots of various parameters of the tracked target. Fig. 3 provides a high level block diagram of the simulation environment.

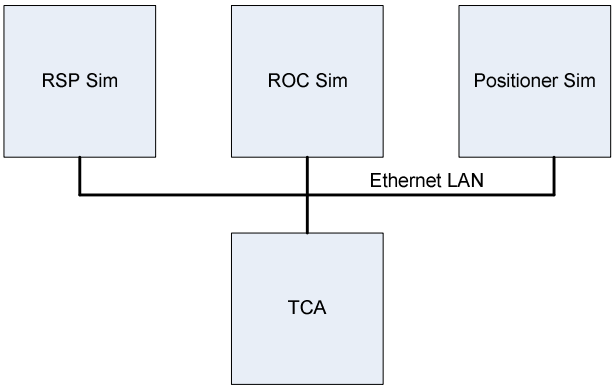

Figure 3: TCA simulation environment

The Simulation environment consisted of the following entities:

- RSP simulator
- Radar Operator Console (ROC) simulator
- Radar positioner simulator
- TCA

A. RSP simulator

The purpose of the RSP simulator was to simulate the interaction with the RSP which the TCA would encounter in the final system. In other words, it implemented the messages as defined in the Communication Control Document (CCD) which would exist in the final system. It provided the TCA with target detection and estimation data messages. The RSP simulator was developed as a windows application. The RSP simulator provided a GUI which allowed the user to setup parameters for up two simulated targets. The GUI allowed the user to specify the following parameters for each of the targets:

- Target(s) starting position
- Target(s) starting velocity
- Path type (inbound, outbound, fly past, circular around target)

The RSP simulator also allowed the user to select a flight profile to be read from a previously generated file. The RSP simulator also allowed the user to specify whether or not the simulated target was in the notional radar beam. This feature allowed the user to simulate loss of target skin echo so as to test the TCA's ability to perform memory tracking and acquisition of pop-up targets. A screen capture of the RSP simulator GUI is given in Fig. 4.

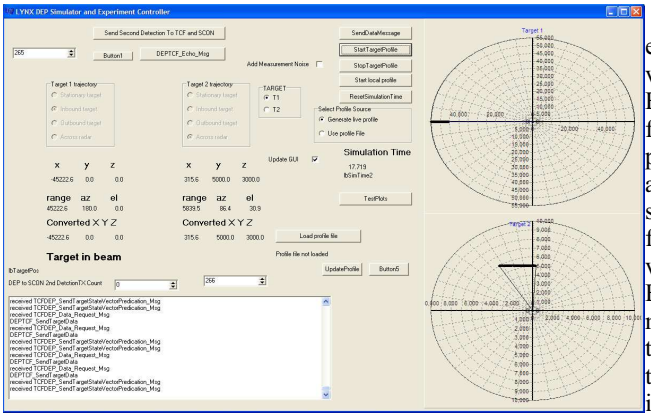

Figure 4: GUI of RSP simulator

B. *Radar Operator Console simulator*

The ROC simulation represented the ROC in the final system. Its function was to provide an indication of the current radar mode, as controlled by the TCA, to allow to user to command the system to a specific mode, the provide a plot of the target(s) being tracked, to provide a list of potential targets which are available for designation, as received from the TCA.

- Display of tentative track information
- Display/plotting of tracked target information
- Display of current radar mode
- Control of current radar mode

C. *Radar positioner simulator*

The radar positioner simulator represents the radar positioner which is responsible for controlling the angular orientation of the radar antenna. The radar positioner simulator provides a means to verify the valid positioner control commands were being sent by the TCA to the positioner.

D. *TCA development application*

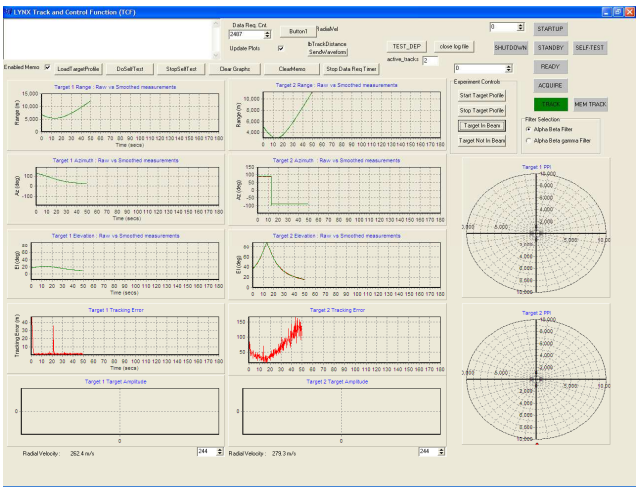

Figure 5: GUI of TCA development application

Although the TCA was intended to be used as a real-time embedded application, it was decided to initially develop and verify the algorithm as a windows application using the Borland C++ Builder development environment. The reason for this was the ease of developing a GUI including real-time plotting of various target parameters. The GUI of the stand-alone TCA application is shown in Fig 5. This version of the software allowed the developers to simulate various target flight profiles, control the mode of the TCA, observe plots of various measured and smoothed target parameters. The C++ Builder environment also allows easy debugging of the code by means of break points etc. An additional advantage of using this environment is that the developers could develop and test the code without having access to a license server computer, as is the case with vxWorks.

Although the TCA was implemented as a windows based application, the developers attempted to completely separate the algorithm code from the GUI code wherever possible so as to ease the porting of the application to the final real-time operating system.

VI. PRELIMINARY INTEGRATION AND TESTING

After initial stand-alone development and testing, the TCA was modified to implement the actual communications as specified for the final system. The TCA was then tested further in the simulation environment as described above.

VII. PORTING OF CODE TO REAL-TIME OPERATING SYSTEM

After testing of the windows based version of the TCA, it was necessary to port the algorithm component of the TCA to the final real-time operating system (vxWorks). As care was taken to keep the GUI code separate from the algorithm code, the porting exercise was easily accomplished. A number of changes were required to be made to the code, especially in the cases where the windows version of the code made use of mechanisms specific to the windows operating system. E.g. use of the WIN32 multimedia timer. The vxWorks version of the TCA was first tested on PowerPC based single board computer. This version of the code was tested and verified in the simulation environment described above.

VIII. DEPLOYMENT OF CODE ON EMBEDDED PROCESSOR IN REALTIME SYSTEM

After verification of the vxWorks version of the TCA in the simulation environment, the next step was to deploy the code on the final target platform. The final platform in this case was a PowerPC processor on Commercial Off-The-Shelf (COTS) Single Board Computer (SBC). The SBC had previously been configured to run the vxWorks operating system. Deploying the code on the final target hardware proved to be a simple matter of rebuilding the application for the final BSP. After deploying the code on the final target processor, the application was once again tested and verified in the simulation environment.

IX. INTEGRATION OF TCA WITH REAL COMPONENTS

After deploying the TCA on the final target processor and verifying its functionality in the simulation environment, the next step was to replace the simulated components in the simulation with the actual components. In principle, this only required a change of IP address in the TCA source code, to enable the TCA to communicate with the real module rather than the simulated component. The TCA was integrated in this way with the real ROC and the real RSP.

X. INTEGRATION WITH RADAR SIGNAL PROCESOR

As mentioned above, the TCA was modified to integrate with the RSP by means of replacing the RSP simulator's Internet Protocol (IP_ address with the IP address of the actual RSP. In principle, this was all that was required to be done. In reality, this was only the start of an extended effort to successfully integrate the TCA with the signal processor, requiring the TCA developers to work with the developer of the RSP embedded software, and thoroughly test and debug the required functions.

The reasons for the difficulty of integration are discussed below.

XI. LESSONS LEARNED

A. Benefits of simulation

The simulation driven approach proved to be very successful in the sense that development and implementation of the algorithms could proceed before all of the other software and hardware modules were available. It allowed the developers to experiment with and refine the algorithms in an environment resembling the final system. What made the simulation approach possible in the case of the TCA development is that the TCA communicated with the rest of the system over Ethernet. Thus, in principle, if one simulated the other modules in the system to a high degree of fidelity, it would make no difference whether the TCA was communicating with a real module of a simulation thereof.

B. Pitfalls of simulation

Although the TCA had been successfully integrated with a simulation of the RSP, the integration with the actual RSP proved to be challenging. This can arguably be explained by an inadequate level of fidelity of the RSP simulation, with which the TCA was integrated. With hindsight, the RSP simulation should have been designed to more accurately model the real RSP. The RSP software designers should have been involved more in the development of the simulation. In the case of the TCA, there was a significant interval between the integration of the TCA with the simulation and the integration with the real RSP. The integration with the RSP occurred towards the end of the system development, contributing to a delay in delivery of the complete system.

The lesson learned regarding the use of simulation during the TCA development is that one should go to great lengths to ensure that the simulations accurately reflect reality, possibly be involving the developer(s) of the modules being simulated, at an earlier stage. One must also avoid becoming complacent, and remember that a module has only been successfully developed when it has been verified in the real system, notwithstanding its apparent proper functioning with simulators.

C. Importance of module testing

During the effort to integrate the TCA with the RSP, the inability of the TCA to successfully acquire and track targets was often assumed to be caused by an error in the TCA algorithm while this was not necessarily the case. The lesson learned from this is not to assume that a particular module is at fault and to wherever possible, test each module as thoroughly as possible independently.

D. Importance of proper identification of system bugs

This follows on the previous comment that having tested systems modules independently as far as possible, it is important to keep an open mind about what may be causing undesired behavior. Rather put more effort into correctly identifying the source of a system bug than the waste time trying to solve a bug in a module where it doesn't exist.

XII. CONCLUSION

The successful development of a real-time embedded application has been described. The requirements of the software were given. A model based use-case driven, iterative approach was followed. The software application was first tested using a general purpose operating systems and development environment. The application was tested in a simulation environment before being ported to the final real-time operating system and target processor. A review is given of the benefits and pitfalls of using a simulation based approach.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the National Program for Electronics, Communications and Photonics at the King Abdulaziz City for Science and Technology for supporting this work.

REFERENCES

[1] R. Penoyer, "The Alpha-Beta Filter", *C User's Journal,* vol. 11, no. 7, pp. 73-86, 1993.

[2] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, 2001.

[3] M. Fowler, and K. Scott, UML Distilled, 2nd ed. Addison-Wesley,2000.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.

[5] M.I. Skolnik, Introduction to Radar Systems, International Student Edition. New York, NY, USA: McGraw-Hill, 1962, pp. 1-18, pp. 72-162, pp. 570-601I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.