Modern Air Situation Picture Display for Air Surveillance Radar Applications

Milovan Stamatović, Miloš Jevtić, *Member, IEEE*, Una Kisić, *Member, IEEE*, Miloš Tatarević, Tanja Pajić, *Member, IEEE*, and Ksenija Marković, *Member, IEEE*

***Abstract* — We have developed a modern air situation picture display applicable in air surveillance radars. The display is based on commercial of-the-shelf hardware and custom software. This paper is focused on the details of the development process. Comparison with a similar commercially available product is given too.**

***Keywords* — Air Surveillance, Plan Position Indicator, Radar, Scan Conversion**

I. Introduction

THE term RADAR was coined as an acronym for ***ra**dio **d**etection **a**nd **r**anging* [1], referring to a system which obtains knowledge about distant objects by transmitting a signal of known waveform and analyzing the received echoes. The acronym lost its capitalization during more than seventy years of its use, becoming a regular noun – *radar*. An important application area of radar is *air surveillance*, defined as "systematic scanning of a portion of the airspace esp. by electronic or visual means to detect and track flying aircraft or missiles" [2]. In the rest of this paper, "radar" will refer to "air surveillance radar" while "targets" will refer to "flying aircraft or missiles".

A typical radar transmits short pulses via a directional antenna which is mechanically rotated at a constant rate. If antenna is directed towards a target, a small portion of transmitted energy is reflected from the target back to the antenna, appearing as a pulse in received signal. A delay between transmitted and received pulses is proportional to distance (or range) of the target. The signal of current antenna direction (*antenna azimuth*) and the demodulated received signal (*radar video*) are a basis for generating the air situation picture.

Older radars presented the air situation picture to operators using a display called ***p**lan **p**osition **i**ndicator* (PPI). In PPI, which is based on **c**athode **r**ay **t**ube (CRT) technology, radar video and antenna azimuth are used directly by the circuitry controlling electron beam steering and intensity, resulting in a map like presentation in polar coordinates of range and azimuth. The display is dark except when echo signals are present, and the phosphor layer covering the CRT ensures the prolonged visibility of target echoes, a feature called *persistence*. An air situation picture generated by using only radar video and antenna azimuth is sometimes called a *raw radar picture*.

While PPI was an impressive invention for its time, it is expensive, bulky and complicated for maintenance. Thanks to advances in digital technology, it is now possible to present raw radar picture using computers and raster displays. The main challenge with this approach is that two specific features of PPI now have to be emulated. The first feature is that PPI is tailored for visualizing the information described with radar video and antenna azimuth. With raster displays this information must be transformed from polar to Cartesian coordinate system, through a process called *scan conversion*. The second feature is persistence.

Besides the raw radar picture, modern radar displays usually visualize two more categories of data. The first includes *plots* and *tracks* generated by ***a**utomatic **d**etection and **t**racking* (ADT) systems inherent to most modern radars [3]. The second includes various maps, markers, and other additional information needed by radar operators.

Thanks to our previous experience with software based radar display techniques [4], a user has asked us to develop a modern radar display which would be used as a cost-effective PPI replacement in legacy radars and operations centers. The product was entitled ***D**igital **R**adar **I**ndicator* (DRI). The rest of the paper is organized as follows. In the second section, the development process of DRI is described in detail. In the third section, DRI is compared with a similar product, and the fourth section concludes the paper.

II. Development Process

To develop DRI, we followed a systematic approach consisting of the following activities:

- Requirements capture,
- Requirements analysis,
- Design,
- Implementation,
- Testing.

A. Requirements capture

Requirements breakdown for DRI is given in Fig. 1 in the form of ***Sys**tems **M**odeling **L**anguage* (SysML) requirement diagram. Original specification is represented with "Source Requirements" package, consisting of six statements, some of which we decomposed into simpler sub-statements.

Ministry of Education, Science and Technological Development of Republic of Serbia supported the work: M. Stamatović, M. Jevtić, U. Kisić, M. Tatarević, T. Pajić and K. Marković by grant TR-32051.

All authors are with University of Belgrade, Institute Mihailo Pupin, Volgina 15, 11060 Belgrade, Serbia. Corresponding author is Miloš Jevtić (e-mail: milos.jevtic@pupin.rs).

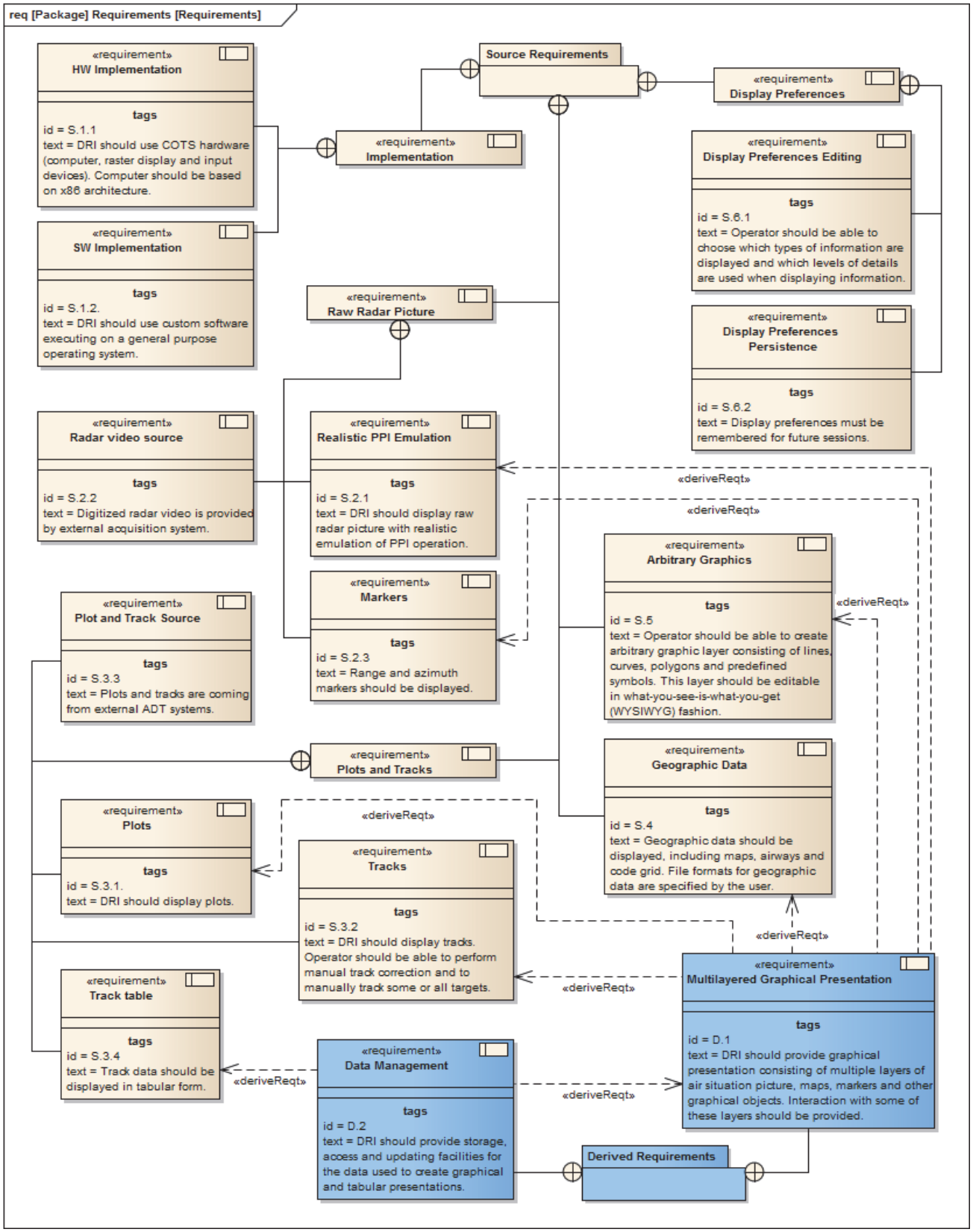

Fig. 1. Requirements breakdown for DRI.

We also derived two new requirements which are shown in light blue color in Fig. 1.

B. Requirements analysis

In this phase we decomposed DRI into functional blocks according to the requirements from previous phase. During the analysis we only considered requirements which are leaves of requirement hierarchy trees. Identified functional blocks, their composition and relations with the requirements are shown in Fig. 2 by means of SysML block definition diagram. Most of the functionality is allocated to the DRI Application block. A list of its sub-blocks, along with their responsibilities, follows:

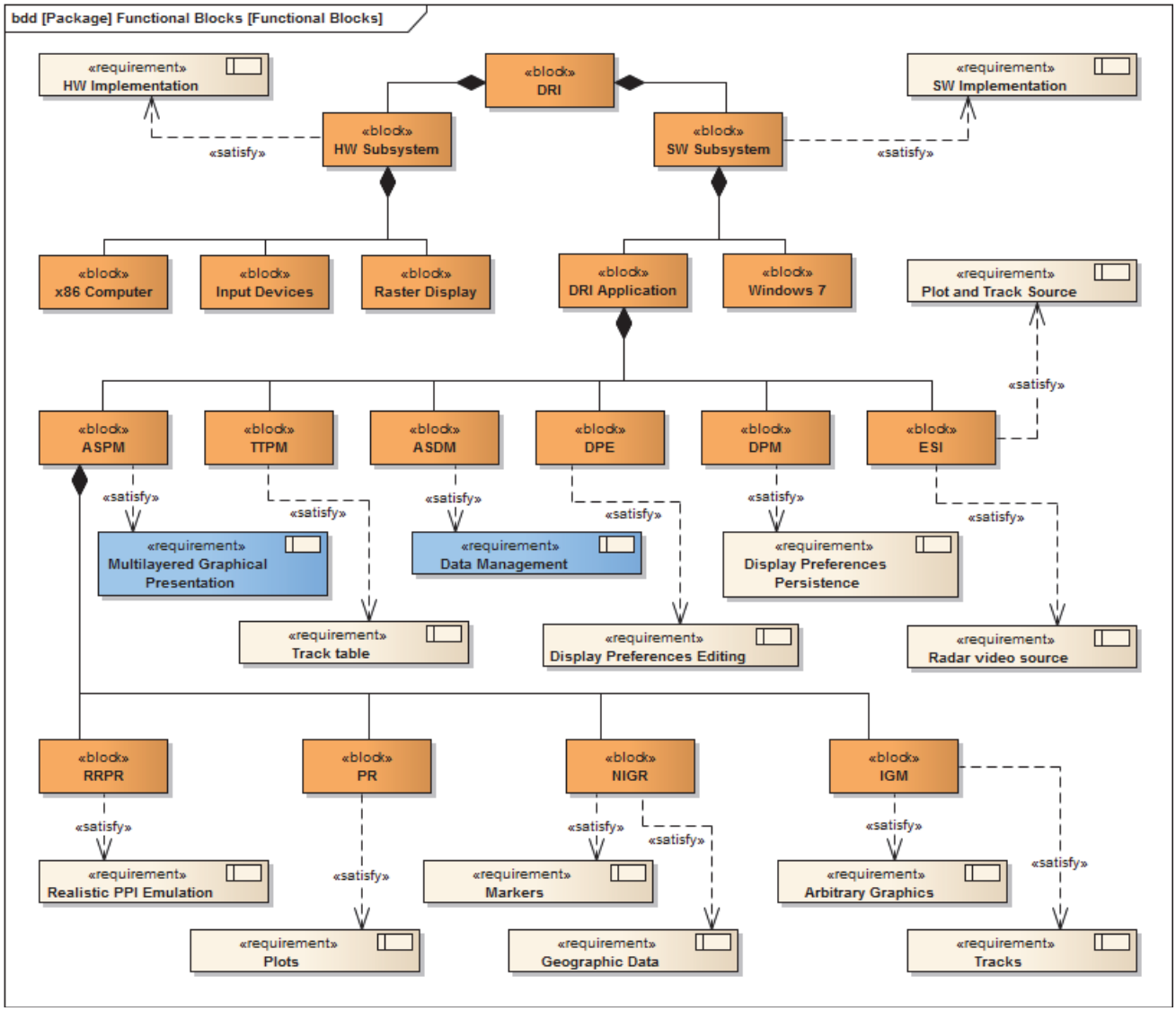

Fig. 2. Functional decomposition of DRI.

- ***A**ir **s**ituation **p**resentation **m**anager* (ASPM) – graphically displaying all elements of air situation picture and handling user interaction. It consists of:
  - ***R**aw **r**adar **p**icture **r**enderer* (RRPR) – rendering scan-converted digitized radar video with emulated persistence.
  - ***P**lots **r**enderer* (PR) – rendering plots.
  - ***N**on-**i**nteractive **g**raphics **r**enderer* (NIGR) – rendering range and azimuth markers, code grid, maps and airways. Parsing geographic data files.
  - ***I**nteractive **g**raphics **m**anager* (IGM) – rendering and handling user interaction with tracks and arbitrary graphics objects.
- ***T**racks **t**able **p**resentation **m**anager* (TTPM) – responsible for displaying tracks in tabular form.
- ***A**ir **s**ituation **d**ata **m**anager* (ASDM) – storing data used by ASPM and TTPM.
- ***D**isplay **p**references **e**ditor* (DPE) – enabling user to edit display preferences.
- ***D**isplay **p**references **m**anager* (DPM) – storing display preferences.
- ***E**xternal **s**ystems **i**nterface* (ESI) – exchanging data (radar video, plots, tracks, manual tracking requests) with acquisition and ADT systems.

C. Design

To design an adequate software architecture, we tried to connect previously identified functional blocks with well known problems, solutions and techniques in software development domain.

For example, ASPM and TTPM need to represent the same data (tracks) stored in ASDM, but in different ways (graphical and tabular). This is typically addressed with ***M**odel-**V**iew-**P**resenter* (MVP) design pattern [5]. In MVP, *views* are active windows which display data stored in the *model*, and handle basic user interactions. These interactions are immediately passed to *presenters* (each view is connected with exactly one presenter, forming a presenter-view pair) which perform actions on the model. To further separate data from its presentation, we employed *Passive View* [6] variant of MVP. In this scheme, there are no dependencies between view and model, and a presenter takes the responsibility of updating the view. To enable presenters to be notified of model changes, we applied *Observer* design pattern [7].

In this context, ASDM is the model, ASPM is divided

into ***A**ir **s**ituation **p**resenter* (ASP) and ***A**ir **s**ituation **v**iew* (ASV), and TTPM is divided into ***T**racks **t**able **p**resenter* (TTP) and ***T**racks **t**able **v**iew* (TTV). Since most data changes originate from external sources, we figured that ESI could be treated as a "presenter". For instance, when ADT system updates a track, it is like a virtual "user" interacted with the track via a virtual "view", causing "presenter" (i.e. ESI) to change the model, in turn updating ASV and TTV. Similarly, when a real user interacts with a track via ASV, ASP changes the model, ESI gets notified and updates a virtual "view" informing a virtual "user" (i.e. ADT system) of a manual tracking request.

IGM and PR perform typical presenter tasks, and it was clear at this point that they should actually be subunits of ASP. These two could be blended together, but since there is no user interaction with plots, PR was separated to allow for performance optimization. While being subordinated to ASP, RRPR and NIGR don't follow MVP scheme.

To emulate persistence effects in real-time, RRPR needs to update all pixels of raw radar picture in every frame. For performance reasons, raw radar picture data is not kept in the model but in RRPR, and digitized radar video is transferred from ESI directly to RRPR, so the MVP architecture is bypassed completely. We decided to use reverse scan conversion in RRPR. In this approach, each screen pixel is mapped from Cartesian to polar coordinates and filled with a value of closest radar video sample.

Data used to render non-interactive graphics objects is constant during one DRI session, so there is no need for it to be kept in the model. To render a graphical representation of this data, a large number of graphics primitives must be drawn, which is a time consuming operation. On the other hand, this representation changes only upon user action (a change of display preferences). Therefore we designed NIGR to render graphics objects into a bitmap, and to update the bitmap only when display preferences change. The bitmap is at ASP's disposal to blend it with other layers when updating the ASV. We envisioned NIGR to have a separate thread in which drawing to the bitmap is done. Rendering is partitioned between separate renderers for markers, code grid, airways and maps. Maps and airways renderers are responsible for parsing geographic data files too.

DPM was designed as a globally accessible entity which employs Observer pattern to notify presenters and renderers of a display preferences change. DPE is partitioned over several windows which consist of generic and custom **g**raphical **u**ser **i**nterface (GUI) controls.

D. Implementation

For performance reasons DRI Application was written in C++ programming language. To simplify and streamline the development, we employed Qt application framework [8], which is the most modern framework for C++ development, besides being open-source and multi-platform. To achieve credible emulation of PPI including persistence effects, lots of pixels need to be updated in every frame, and the frame rate itself needs to be high. This implied the use of hardware acceleration, and we chose OpenGL as an **a**pplication **p**rogramming **i**nterface (API) to accelerated graphics. OpenGL was chosen because it is an industry standard supported on many platforms and it integrates well with Qt. UML class diagram showing DRI Application's implementation is given in Fig. 3.

Qt's ***G**raphics **V**iew **F**ramework* (GVF) was a basis for implementing functionalities related to interactive graphics. In this implementation, ASV is a class derived from *QGraphicsView*, while IGM is partitioned over several *QGraphicsItem* derived classes (responsible for rendering graphics objects on the view and handling user interaction with them) and a *QGraphicsScene* derived class (responsible for keeping items together and storing their external state, e.g. selection and focus). To update ASV, IGM just updates QGraphicsItems, while actual rendering to ASV is performed at a later time, initiated by the GVF. In contrast, PR and RRPR bypass GVF and render directly to ASV. The same goes for ASP when rendering a bitmap generated by NIGR.

The main challenges in RRPR implementation were emulation of persistence effects and scan conversion. In PPI, a point on the CRT remains illuminated for some time after it was hit by an electron beam thanks to phosphorescence effect, but this illumination slowly fades away as time passes. We can think of the phosphor layer as some form of storage. To emulate persistence effects in digital domain, we also need storage, sometimes called *polar store*. It is easiest to think of polar store as of a matrix of radar video samples, with vertical coordinate being azimuth and horizontal coordinate being range. Eventually, thanks to scan conversion, the value of each radar video sample in polar store will affect the illumination of one or more pixels on the raster display. Therefore to emulate a fading effect, the values of samples in polar store must be decremented as time passes. There are two approaches to doing this: *scan-based* (the value of a sample is decremented only when antenna revisits that sample's azimuth after making a full rotation), and *real-time* (each sample's value is decremented in regular time intervals regardless of antenna rotation). We opted for real-time variant as it enables emulation of nonlinear fading effects, in turn providing very realistic imitation of PPI as required by the user. In our implementation, polar store is a set of textures in video memory, where each texel stores a value of one radar video sample. Decrementing of sample values and updating of the textures with new samples is performed by a ***f**ragment **p**rogram* (FP). Scan conversion is achieved by mapping polar store's textures to meshes which approximate circle sectors and then rendering the meshes. Interpolation inherent to OpenGL rasterizer assures that there are no holes in final picture. To paint a screen pixel in a preferred color, another FP is utilized.

The most complex part of NIGR implementation was maps renderer. Requirements specified that geographic data should be read from files supplied by user. The format of these files is the ESRI Shapefile format (*Shapefile*) [9], and map projection used is ***U**niversal **T**ransverse **M**ercator* (UTM).

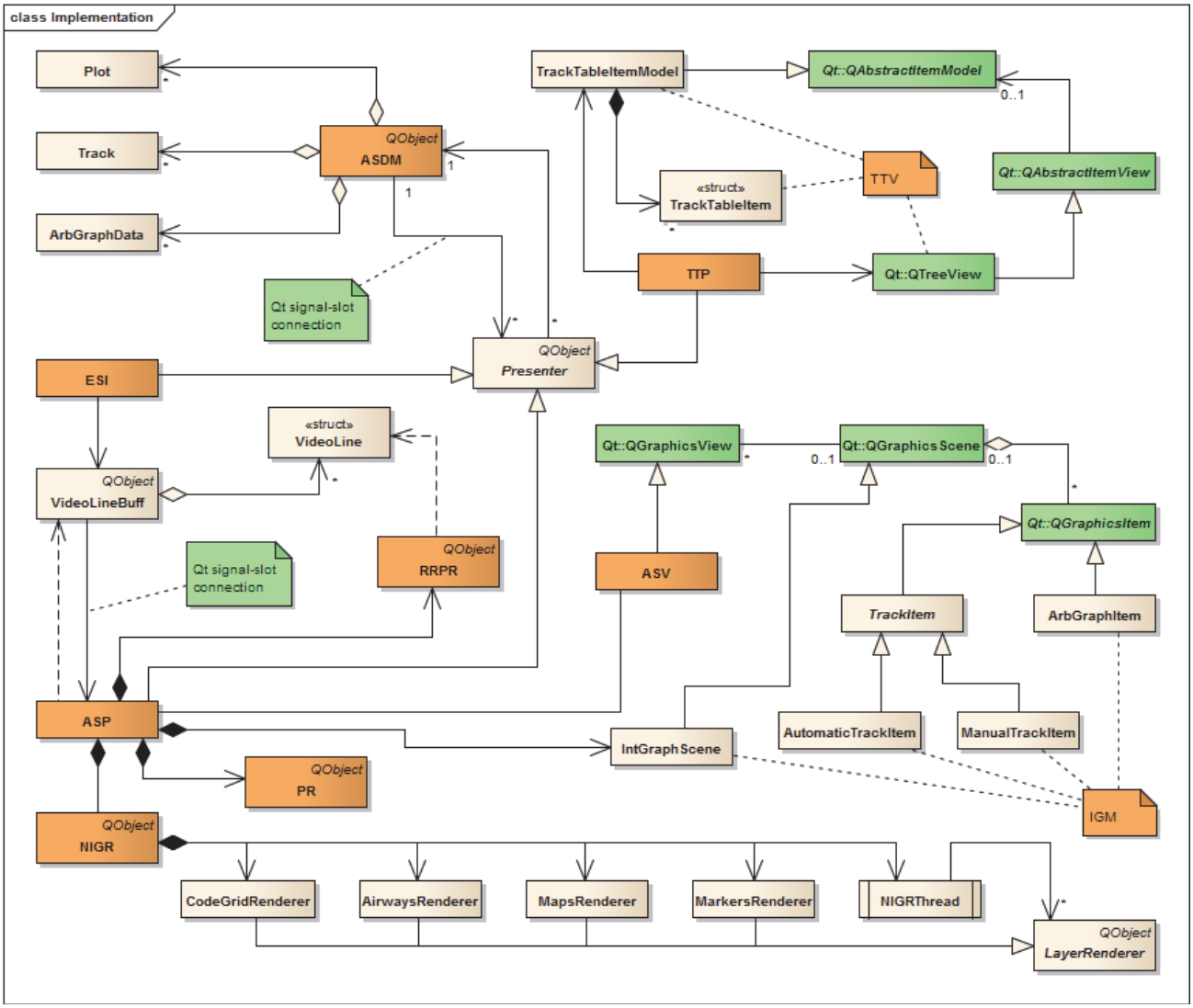

Fig. 3. Implementation of DRI Application.

Shapefile stores spatial geographical data as geometry and attribute values in compact data sets. Geometry values are simple primitive geometrical data types of points, lines and polygons and do not contain topological data which make it suitable for fast drawing modes and edit functions.

A raw radar picture is naturally presented in a plane tangential to radar position. To accurately present maps over the raw radar picture, we first transform UTM coordinates of zone numbers, easting and northing to geodetic coordinates of longitude and latitude and then project them on a mentioned plane. Since geographic data files can contain multiple types of geographic data, such as cities, rivers, lakes, etc., we organized the data structures in a multilevel manner. The *Geocontainer* class is a top level structure which contains various types of geographic elements. Each geographic element – *GeoEntity*, can consist of different geometry object types supported in the Shapefile format. GeoEntity is an array of simple objects that are defined by their pixel representation in a calculated zoom value and attribute data.

Fig. 4 shows a screenshot of DRI with various elements of air situation picture.

E. Testing

The first intended use of DRI was in a system called ***M**obile **A**ir **S**urveillance **C**enter* (MASC). MASC can be connected to several types of legacy radars and it includes ADT system and multiple operator posts (implemented with DRIs). MASC prototype successfully passed acceptance tests performed by the user. During these tests, DRI's performance was evaluated by comparison with several reference displays including legacy radar PPIs and other air situation displays. DRI fulfilled all the user requirements and its performance was satisfactory.

III. Related Work

We compared DRI with a similar product, *RadarView* from Cambridge Pixel Ltd [10]. Both DRI and RadarView are based on COTS hardware and custom software running under a general purpose operating system. We compared the features of both products and summarized comparison results in Table 1. It must be noted that some features of RadarView were omitted from the comparison as they are not relevant for an air surveillance radar display.

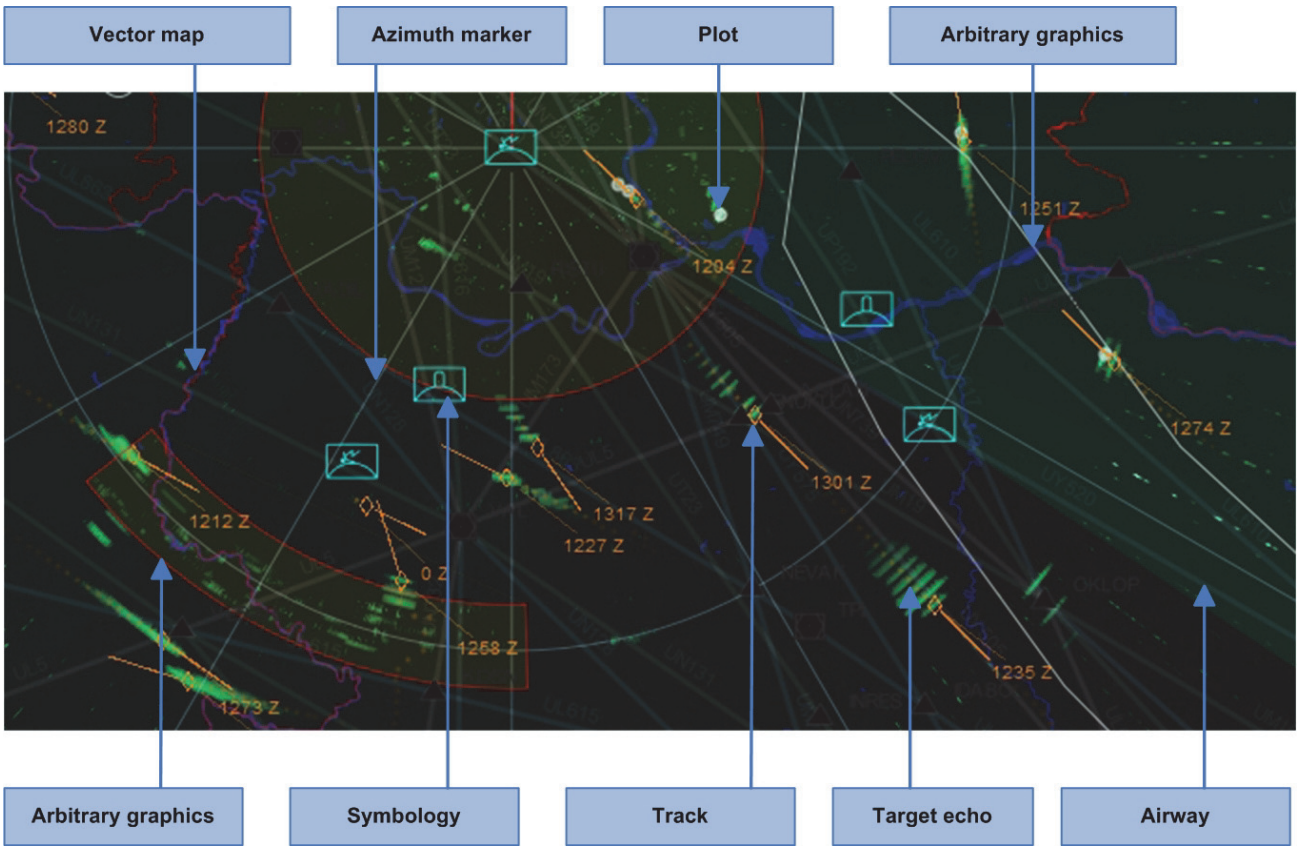

Fig. 4. Elements of the air situation picture as displayed by DRI.

It can be seen from Table 1 that there are no substantial differences between DRI and RadarView, though the latter has a richer set of features. This is not surprising since RadarView is a commercially available product with a broad application scope, while DRI was developed from specific user requirements.

TABLE 1: COMPARISON WITH SIMILAR PRODUCT.

	RadarView	***DRI***
Raw radar picture with programmable color, persistence emulation and retained trails on view change	yes	yes
Tracks	yes	yes
Max. num. of air situation windows	5	1
Max. num. of radar video channels per window	2	1
A-scope	yes	yes
B-scope	optional	no
Raster maps	yes	no
Vector maps	yes	yes
Range and azimuth markers	yes	yes
Editable arbitrary graphics	no	yes

IV. CONCLUSION

A modern air situation picture display based on COTS hardware and custom software has been developed, by following a systematic approach which we describe in detail. A comparison of developed system with a commercially available radar display which employs similar implementation principles didn't reveal substantial differences when core features are considered. While the developed system is intended for air surveillance radar applications, it could be modified for use in other types of radars which operate on a similar principle (such as civil marine radars or vessel tracking service radars) or in fusion centers of distributed surveillance systems. These or similar modifications could be the subject of future work.

REFERENCES

[1] D. Barton, S. Leonov, *Radar Technology Encyclopedia*. Norwood, MA: Artech House, 1997, pp. 320-321.

[2] *Merriam-Webster Online Dictionary*. Available: http://www.merriam-webster.com

[3] W. G. Bath, G. V. Trunk, "Automatic detection, tracking, and sensor integration," in *Radar Handbook*, 3rd ed., M. I. Skolnik, Ed. New York: McGraw-Hill, 2008.

[4] M. Jevtić, M. Stamatović, „Radar Data Processing and Visualization on Desktop Platforms," in *Proc. 17th Telecommunications forum TELFOR 2009*, pp. 1315-1318.

[5] M. Potel, "MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java," Taligent, Inc., 1996.

[6] M. Fowler. (2012, Sep., 21.) *Development of Further Patterns of Enterprise Application Architecture* [online]. Available: http://www.martinfowler.com/eaaDev/index.html

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley Pub Co, 1995, pp. 293-304.

[8] J. Blanchette, M. Summerfield, *C++ GUI Programming with Qt 4*, 2nd ed., Prentice Hall, 2008.

[9] *Esri Shapefile Technical Description*, Esri, Redlands, CA, July 1998.

[10] *RadarView - Primary Radar Visualisation*, Cambridge Pixel Ltd, UK, 2012.