EUROPEAN ORGANISATION FOR NUCLEAR RESEARCH

ATLAS DAQ – DCS
Communication Software

High Level Design

Authors: R. Hart (NIKHEF, Amsterdam)
V. Khomoutnikov (PNPI, St.-Petersburg)
Note Number: 165
Version: 2.1
Date: May 17 2001
Reference: http://atddoc.cern.ch/Atlas/Notes/165/Note165-1.html

1 Introduction

This note presents the high level design for the DAQ–DCS Communication (DDC) software of the ATLAS HLT/DAQ/DCS system [1]. The DDC software is intended to be the interface between DAQ and DCS and to support all needs of run time information exchange. A general outline of those needs is described in [2] and has been formally specified in the User Requirements Document [3]. The domain decomposition defined 3 functions the DDC shall provide:

1) Bi-directional exchange of data like parameters and status values;
2) Transmission of DCS messages, like alarms, to DAQ;
3) Ability for DAQ to issue commands on DCS.

The functions listed above are independent and will be implemented as separate subsystems. Apart from these functions, the DDC may also provide a graphical user interface.

For the design of the DDC the following prerequisites, based on the URD, are taking into account:

- The Online software [4] (formerly known as the Back-end) is the interface point for the DDC on the DAQ side.
- The DCS is implemented by a SCADA[1] system. The PVSS II system of ETM [5] will be utilized for that purpose and its API will be used to connect the DDC.
- Any manipulation with the physics data is beyond the scope of the DDC software.
- The DDC uses its own configuration database, containing the information it needs for its functionality mentioned above.
- The *partition* concept is known to both DAQ and DCS and should be compatible in terms of boundaries and locking of resources. Although DDC is not responsible for this demand, it has some consequences for the design and chosen implementation. From the DAQ point of view a partition is a subset of the experiment capable to run independently. The DCS can be partitioned into vertical slices, which control a subsystem of the detector. Such subsystems are defined as arbitrary parts of the detector.
- The DCS is expected to be operational at all times.

Figure 1 presents the basic context diagram of the DDC software.

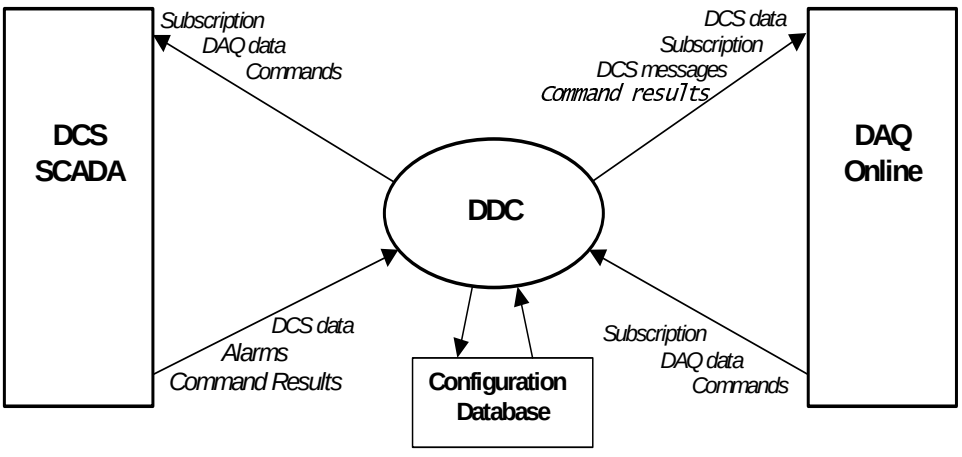

Figure 1. DDC software context diagram.

[1] Supervisory Control And Data Acquisition

Inside the context diagram the term *subscription* is used. Subscription is a way of asynchronous reading. The caller gets the initial value and afterwards it will be notified on all updates.

Each subsystem uses a configuration database to set up itself. It is intended to implement them using the ConfDB component [6] of the Online software, but initially the configuration information will be placed in a common ASCII file.

The SCADA system (PVSS II) contains an internal 'real-time' database based on the *datapoint*[2] concept. These datapoints are accessible by the API of the SCADA system. The PVSS II system has also a 'historical' database, which is not relevant for the design. Wherever the term SCADA database is used in the remainder of the document, the 'real-time' database is meant. Behind the SCADA database there are of course the different PVSS applications, responsible for maintaining their piece of the database. These applications however are not relevant for the design and therefore not shown in the different diagrams and dynamic models.

2 Design of DDC data transfer subsystem

This chapter describes the basic software architecture to provide the bi-directional exchange of data between DAQ and DCS. The subsystem is called the ***DDC Data Transfer (DDC-DT)***. The term *data* is regarded as a value belonging to a DAQ or DCS variable (parameter or status).

The interface point for the DDC-DT on the DAQ side will be the Information Service (IS) [7] component from the Online software. The design of DDC-DT subsystem follows the main philosophy of the IS:

> *No subsystem sends its data directly to other subsystems. Rather, it places them into a common repository (Information Service) making those data available for any subsystem that needs them.*

2.1 DDC-DT Class Diagram

The class diagram of the DDC-DT subsystem is shown in figure 2.

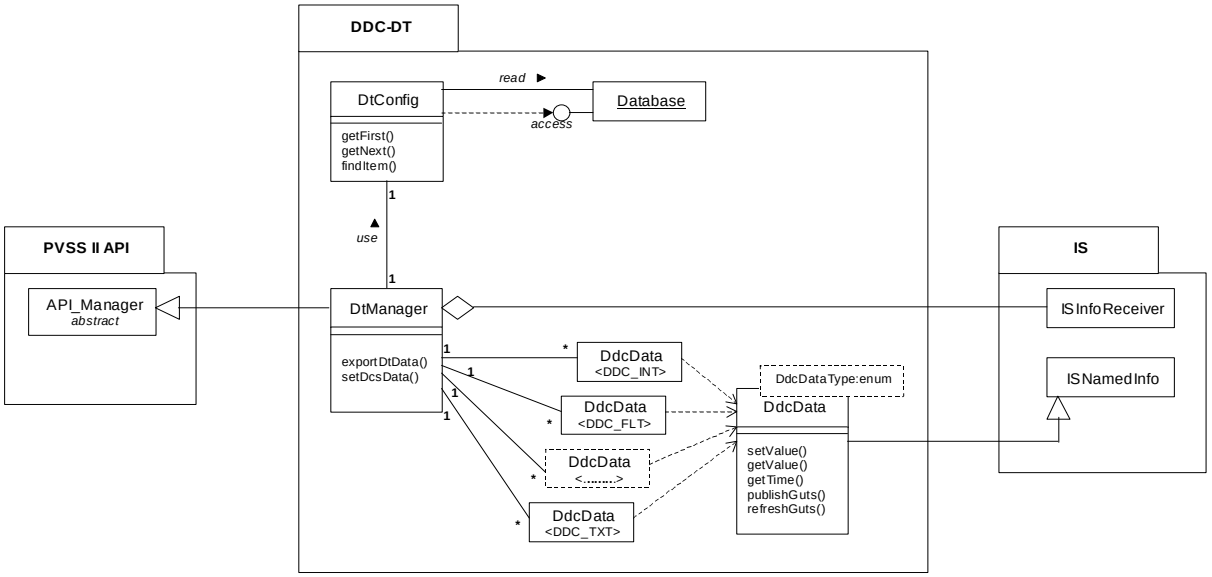

Figure 2. DDC-DT class diagram.

[2] Basic data container of a variable inside the 'real-time' database of PVSS II.

The key class of the DDC-DT subsystem is the ***DtManager*** class. It inherits the abstract ***API_Manager*** class of the SCADA's API, which provides full access - read, write and subscription- to the SCADA database. On the DAQ side it uses an instance of the ***ISInfoReceiver*** class to provide subscription management on the IS. The template class ***DdcData***, inherited from ***IsNamedInfo***, represents the parameterized data structure to be used for the data exchange. The ***IsNamedInfo*** class provides the ability to create, read and write an information object[3] of the IS. The ***DtConfig*** class interfaces the configuration database of the DDC-DT subsystem.

2.2 DDC-DT Dynamic Model

From the bi-directional exchange of data two distinct use cases can be distinguished. One is the transfer of data from DCS to DAQ, the other from DAQ to DCS. An overview of the collaboration between the subsystems participating in the DDC-DT for these two cases is given in figure 3.

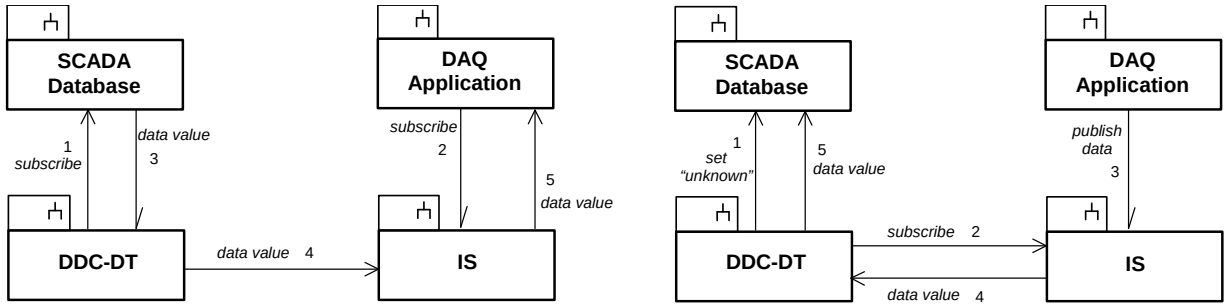

Figure 3. DDC-DT collaboration diagram.

In case of transfer of data from DCS to DAQ the DDC-DT subsystem shall go through the following steps.

1. Read the configuration database, containing for this purpose a list of DCS variables, which should be made available to any DAQ application.
2. Subscribe each variable from the list inside the SCADA Database.
3. Get current value, and afterwards all updates, and publish the value to the IS.
4. From this point any DAQ application is able to monitor the value of a DCS variable.

The above listed sequence is visualized in figure 4.

[3] Information is the object which has a name, a type and a value.

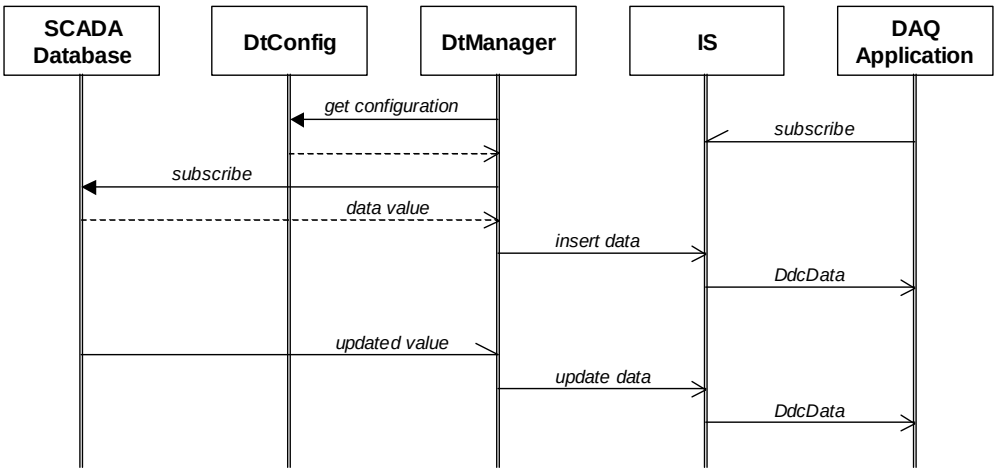

Figure 4. DDC-DT sequence diagram (*DCS → DAQ data transfer*).

The other use case, sending data from DAQ to DCS, is almost symmetric. The main difference between both cases is the fact that the SCADA system is supposed to be available at all times, while the DAQ applications are not. This implies that the SCADA system has not always the availability of the variables, which are to be maintained by DAQ applications. The variables which are not supported (yet) by a DAQ application, are set to "unknown". This value is also set when the corresponding DAQ application terminates or the data object is removed from the IS. The following steps are carried out and visualized in figure 5:

1. Read the configuration database, containing for this purpose a list of DAQ variables, which will be made available to the SCADA system.
2. Subscribe for each variable (an information object) inside the IS. Initially the variables are set to "unknown" inside the SCADA database.
3. Wait for DAQ application, which supports the variable. Get current value, and afterwards all updates, and copy the value to the SCADA Database.
4. Set value to "unknown" if DAQ application terminates.

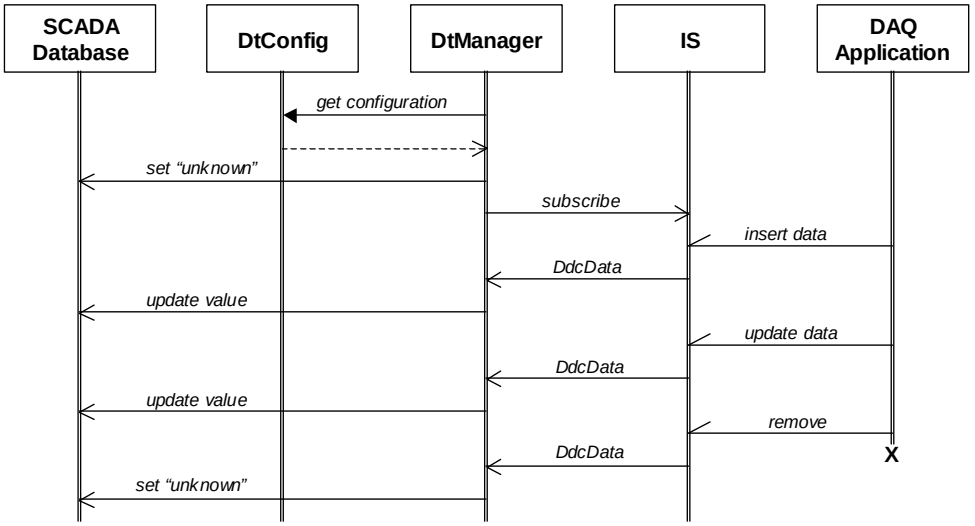

Figure 5. DDC-DT sequence diagram (*DAQ → DCS data transfer*).

3 Design of transporting DCS messages to DAQ

This chapter describes the basic software architecture to provide the transport of DCS messages, like alarms, to any DAQ application. The subsystem is called the ***DDC Message Transport (DDC-MT***. The term *message* is regarded as a piece of information packed into a string.

The interface point for the DDC-MT on the DAQ side will be the Message Reporting System (MRS) [8]. The design of DDC-MT subsystem follows the main philosophy of MRS:

> *No subsystem sends its messages directly to any other subsystems. Instead of that, it sends them to a common distributor (Message Reporting System) making those messages available for any application, which needs them.*

3.1 DDC-MT Class Diagram

The class diagram of the DDC-DT subsystem is shown in figure 6.

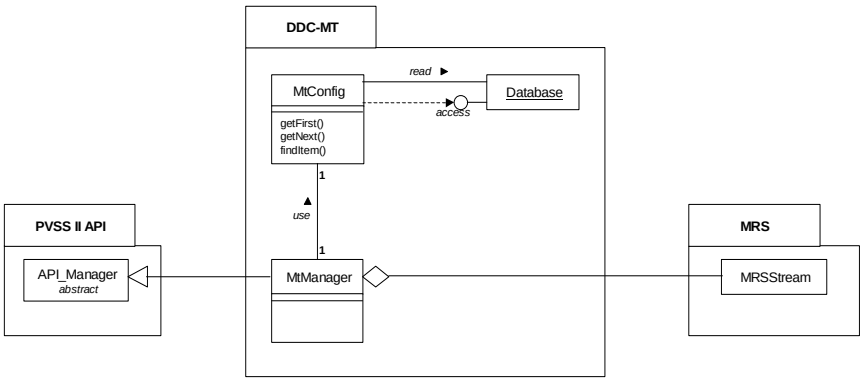

Figure 6. DDC-MT class diagram.

The ***MtManager*** class is the main class of the DDC-MT subsystem. It inherits the abstract ***API_Manager*** class of the SCADA's API, which provides the necessary functionality – read and subscribe - to the SCADA database. On the DAQ side it uses an instance of the ***MRSStream*** class to provide the means to inject messages to the MRS. A number of public member operators are available for the construction of messages. The ***DtConfig*** class interfaces the configuration database of the DDC-MT subsystem. The database contains a list of datapoints (variables and alarms) to subscribe and format attributes. The format attributes are used to generate the message, belonging to a specific datapoint. In terms of PVSS II, an *alarm* is a specialized datapoint. For the design they are regarded as common variables.

3.2 DDC-MT Dynamic Model

The collaboration diagram of the DDC-MT subsystem is shown in figure 7.

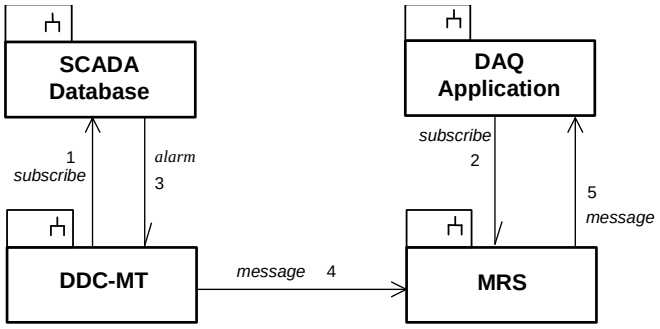

Figure 7. DDC-MT collaboration diagram.

The DDC-MT subsystem shall go through the following steps, which are visualized as sequence diagram in figure 8.

1. Read the configuration database, containing a list of alarms and variables, together with their MRS specific attributes like severity, qualifier, etc.
2. Subscribe the alarms and variables.
3. Catch the alarm or changed variable, generate a message and send it to MRS.
4. From this point any DAQ application is able to receive the messages.

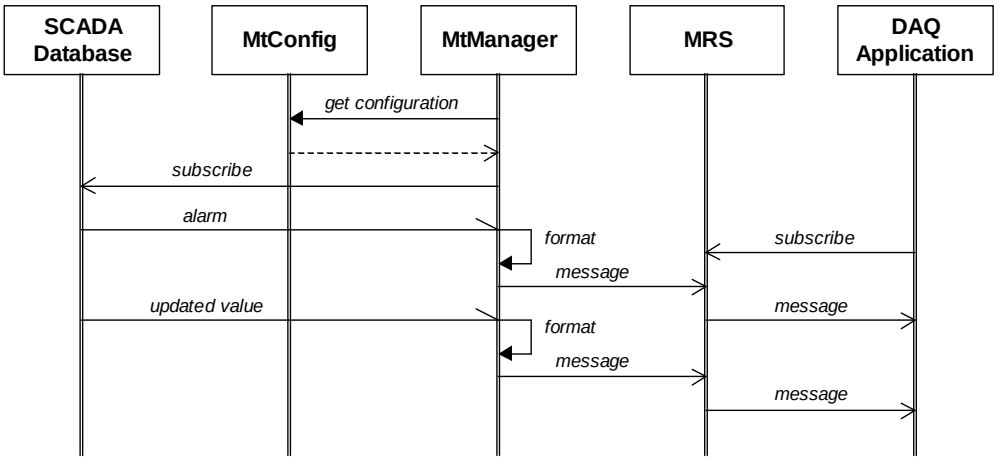

Figure 8. DDC-MT sequence diagram (*sending DCS messages to DAQ*).

4 Design of DDC command transfer

This chapter presents the basic software architecture to provide DAQ the ability to execute commands on DCS. This subsystem of the DDC software is called the ***DDC Command Transfer (DDC-CT)***. The Run Control (RC) component [9] will be the interface point on the DAQ side, which implies that a *command* is regarded as a *controller transition.* Examples of such transitions are: *load, configure, start* and *stop.* These high-level commands are defined for each controller, which represents a DCS partition. Not any DAQ application should be able to execute such commands. Only dedicated applications, called DAQ *supervisors*, are enabled to use the DDC-CT. The commands have to be implemented on the DCS side and are both semantically and syntactically beyond the responsibility of the DDC software.

4.1 DDC-CT Implementation Issues

Some implementation issues, which are inevitable, have a large influence on the design. The PVSS II system provides the ability to write control procedures by means of *scripts*. They are interpreted by the *Control Manager* and are triggered by setting a dedicated *datapoint*. The scripts have a syntax similar to C/C++ and execute the desired commands. More information about the scripting language and control manager can be found in [5]. Setting a datapoint will be implemented, using SCADA's API and is not more than writing a variable inside SCADA's database as described in chapter 2. An important detail of the script is to write the result of the command back into another dedicated datapoint, which is subscribed and sent back to the caller.

The configuration database of the DDC-CT subsystem contains a list of commands, together with the trigger-datapoint, result-datapoint and a list of auxiliary datapoints for possible parameters. Furthermore it contains a timeout value, indicating the amount of time a command has to finish.

The next chapters contain several diagrams in which the control manager (the PVSS application responsible for executing the commands) is not shown. The DDC subsystems have only access to the SCADA's database.

4.2 DDC-CT Class Diagram

The class diagram of the DDC-CT subsystem is shown in figure 9.

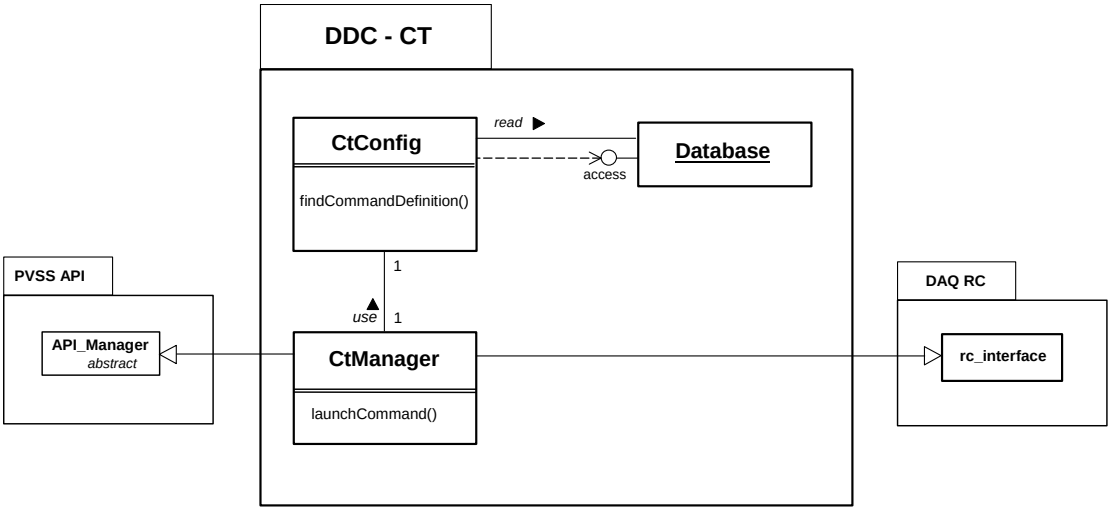

Figure 9. DDC-CT class diagram

The ***CtManager*** is the central class of the DDC-CT subsystem. It inherits the abstract ***API_Manager*** class of the SCADA's API, which provides the necessary functionality – write and subscribe – to the SCADA database. On the DAQ side it inherits the ***rc-interface*** class from the RC component. The necessary configuration information is retrieved by means of the ***CtConfig*** class.

4.3 DDC-CT Dynamic Model

The collaboration diagram of the DDC-CT subsystem is shown in figure 10.

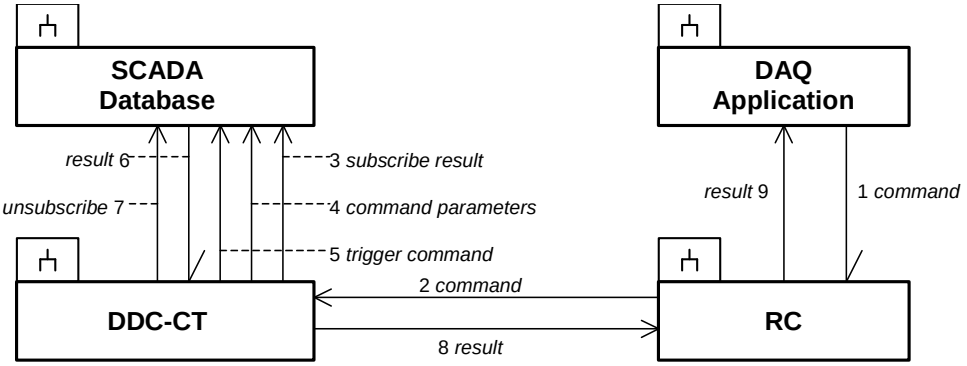

Figure 10. DDC-CT Collaboration diagram

A typical use case for the DDC-CT subsystem, from the ***CtManager*** point of view, goes through the following steps. It is visualized in the sequence diagram of figure 11.

1) Read the configuration database, containing for this purpose a list of DCS command definitions.
2) Validate an incoming command and return an error code if the command is not valid or not authenticated.
3) If valid, subscribe the response datapoint, write the command parameters and update the trigger-datapoint.
4) The Control Manager activates the appropriate script.
5) Catch the command result, send it via the RC to the DAQ application and unsubscribe the response datapoint. (If result does not return within timeout, an error code is returned.)

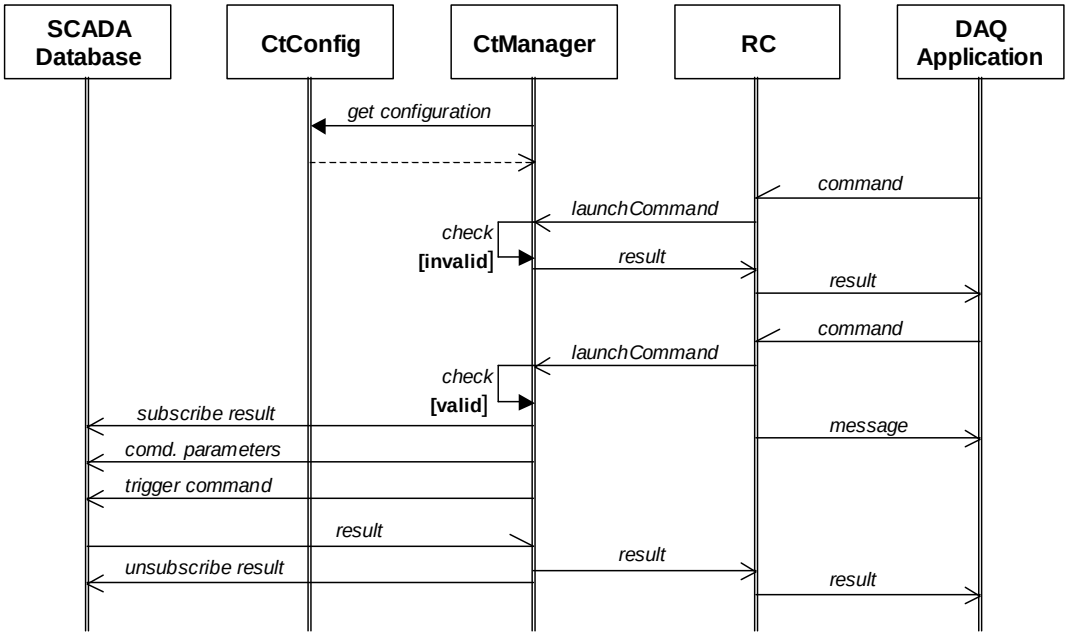

Figure 11. DDC-CT sequence diagram.

5 References

[1] ATLAS high level Triggers, DAQ and DCS Technical Proposal, CERN/LHCC/2000-17, http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/SG/TP/draft_tp.html.

[2] H.Burckhart, M.Caprini, R.Jones "Connection DCS – DAQ in ATLAS", ATLAS DCS IWN8, Nov 1999, http://atlasinfo.cern.ch/ATLAS/GROUPS/DAQTRIG/DCS/dcs_daq_0.6.pdf.

[3] R.Hart, V.Khomoutnikov "ATLAS DAQ–DCS Communication Software: User Requirement Document", DAQ Note-164, Nov. 2000, http://atddoc.cern.ch/Atlas/Notes/164/Note164-1.html.

[4] I.Alexandrov et al, "Performance and Scalability of the Back-End in the ATLAS/DAQ EF Prototype –1", RT99 Santa Fe, http:.//atddoc.cern.ch/Atlas/Conferences/RT99/Rt175.ps.

[5] PVSS II von ETM, http://www.pvss.com

[6] I.Soloviev, "Configuration Databases User's Guide", DAQ Note-135, Sep. 2000, http://atddoc.cern.ch/Atlas/Notes/135/Note135-1.html.

[7] S. Kolos "Implementation of the Information Service: User guide" DAQ Note-037, Aug. 1999, http://atddoc.cern.ch/Atlas/Notes/037/Note037-1.html.

[8] D.Burckhart, M.Caprini, R.Jones, S.Kolos, A.Radu " Message Reporting System. User Guide and Implementation", DAQ Note-059, Feb. 1999, http://atddoc.cern.ch/Atlas/Notes/059/Note059-1.html.

[9] P.-Y.Duval et al, "Run Control User's guide", DAQ Note-107, July 2000, http://atddoc.cern.ch/Atlas/Notes107/Note107-1.html