User Manual
Diagnosis of ADAPT system

Version 1.0

Author: Erik Almqvist
Date: December 15, 2009

Status

Reviewed		09-11-29
Approved		09-12-01

Course name:	Control Project	E-mail:	diagnos2009@googlegroups.com
Project group:	FFF	Document responsible:	Erik Almqvist
Course code:	TSRT10	Author's E-mail:	daner963@student.liu.se
Project:	Diagnosis	Document name:	UserManual1_0.pdf

Project Identity

Group E-mail:	diagnos2009@googlegroups.com
Homepage:	http://www.isy.liu.se/edu/projekt/tsrt10/2009/
Orderer:	Erik Frisk, Linköping University
	Phone: +46 (0)13 28 2035 , **E-mail:** frisk@isy.liu.se
Customer:	The Division of Vehicular Systems, Linköping University
	Phone: +46 (0)13 28 1000 , **E-mail:** Vehicular.Systems@isy.liu.se
Course Responsible:	David Törnqvist, Linköping University
	Phone: +46 (0)13 28 1882, **E-mail:** tornqvist@isy.liu.se
Project Manager:	Niklas Wahlström
Advisors:	Mattias Krysander, Linköping University
	Phone: +46 (0)13 - 28 2198 , **E-mail:** matkr@isy.liu.se

Group Members

Name	Responsibility	Phone	E-mail
Niklas Wahlström	Project manager	0705-122349	nikwa148@student.liu.se
Daniel Eriksson	Document manager	073-4405730	daner963@student.liu.se
Erik Almqvist	Software manager	0705-149935	erija952@student.liu.se
Emil Nilsson	Test manager	073-6766558	emini550@student.liu.se
Andreas Lundberg	Design manager	0704-061227	andlu549@student.liu.se

Document History

Version	Date	Changes made	Sign	Reviewer
0.1	09-11-24	First draft.		Erik Almqvist
1.0	09-11-24	Second draft.		Erik Almqvist

Contents

1 Introduction

This document is a user manual for the diagnositis algorithm, called FFFDA, which has been developed during the CDIO-project "Diagnosis of ADAPT system", by project group **FFF**[1] at Linköping University 2009. The project was part of the course "Reglerteknisk Projektkurs TSRT10".

In this manual there are descriptions on how to setup the diagnosis algorithm, how to compile and run the algorithm together with the DxC framework and how to change algorithm parameters used within the diagnosis algorithm.

The diagnosis algorithm, FFFDA, is developed for use on a Linux system and requires the DxC framework and a GCC compiler to be able to compile correctly.

1.1 Background

NASA is interested in analyzing different ways to monitor whether or not systems that are sent into space are working properly, and also in finding out where the faults are when there are faults present in the system. It is of course beneficial to know exactly which faults that are present in e.g. a satellite before you send someone to repair it. It may also be the case that detecting a fault, and smoothly shutting down the system or limit its activities, can prevent other parts of the system to get damaged. The reasons above illustrates why NASA together with Palo Alto Research Center (PARC) have started an annual competition called the Diagnostic Challange Competition (DCC). The developed diagnois algorithm is intended to participate in the DCC'10, in the Industrial Track System Tier 2 challenge.

2 Installation and integration into the DXC

In order to use the diagnosis algorithm, a framework, DxC, developed by NASA for evaluating diagnosis algorithms must be installed. Instructions on how to install DxC can be found on the DCC competition homepage[dxp09]. The FFFDA diagnosis algorithm is developed in a Linux environment, and although it should be perfectly possible to compile the algorithm in a Windows environment, it has not been tested by the developing team.

Once the installation of the DxC framework is completed the contents of the provided zip file should be extracted to the catalogue `$DXC_HOME/Algs/fffda`, where `$DXC_HOME` is the home catalogue for the DxC Framework. This should produce a file structure like the one shown in figure 1. Once the algorithm is placed correctly it is possible to run the algorithm as described in section 4.

3 Compiling the diagnosis algorithm

If parameters are changed in the algorithm, it will have to be recompiled. In order compile the FFFDA diagnosis algorithm the GNU GCC (G++) compilator and the DxC framework is needed. The G++ compilator can be downloaded at GCC-homepage[2].

Once the G++ compilator is installed the fffda algorithm can be compiled by running the command 'make' in the FFFDA algorithm's home catalogue as shown in 2.

[1] Finn Fem Fel
[2] http://gcc.gnu.org/

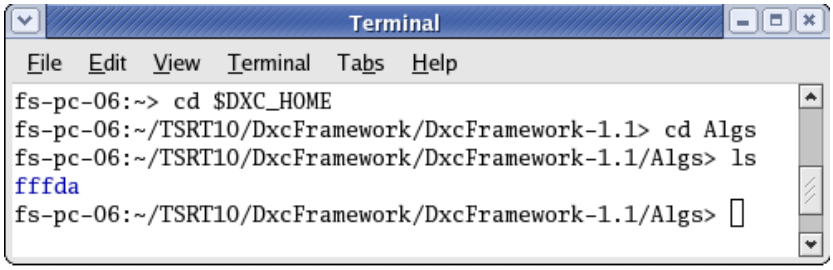

Figure 1: The installation directory once the algorithm is installed.

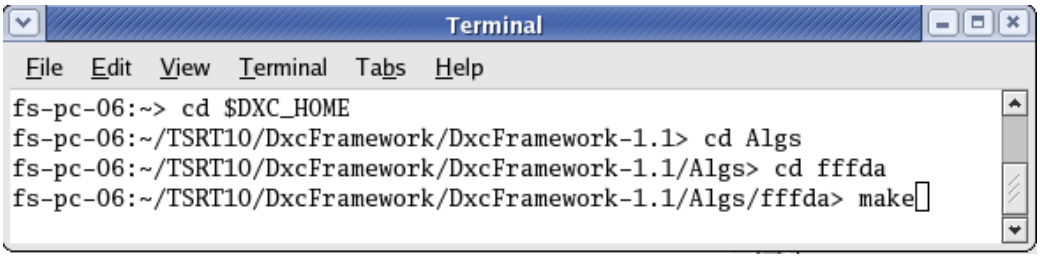

Figure 2: In order to compile the FFFDA, run the make command in the FFFDA home-catalogue.

4 Running the diagnosis algorithm

The FFFDA algorithm can be executed through the DxC framework in two different ways. Either by providing a specific scenario for the algorithm to run, or by running through all the available scenarios (in the `$DXC_HOME/Scenarios/ADAPT` catalogue).

In order to run all the available scenarios, use the ScenarioLoader executable provided by the DxC framework. This executable is usually placed in the `$DXC_HOME/Bin` catalogue, and can be runned by typing `./ScenarioLoader` (see Figure 3).

In order to run standalone scenarios two different terminals is needed. Place the first terminal in the `$DXC_HOME/Bin` catalouge and run the command `./StandaloneSDS [SCENARIO]` where `[SCENARIO]` is the path to the scenario that is to be run. Place the other terminal in the `FFFDA/Bin` directory and run the command `./fffda` to execute the FFFDA diagnosis algorithm (see Figure 4).

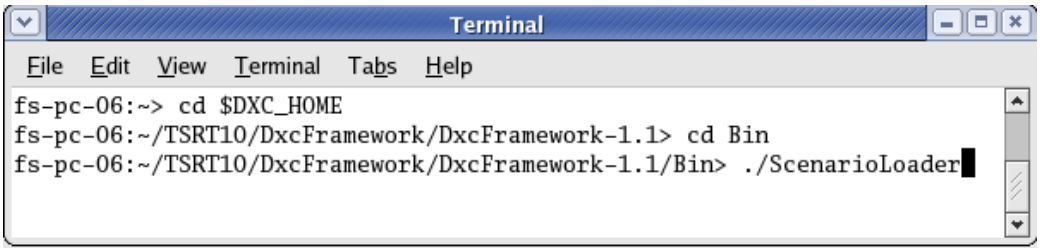

Figure 3: Run the FFFDA by running the DxC's ScenarioLoader...

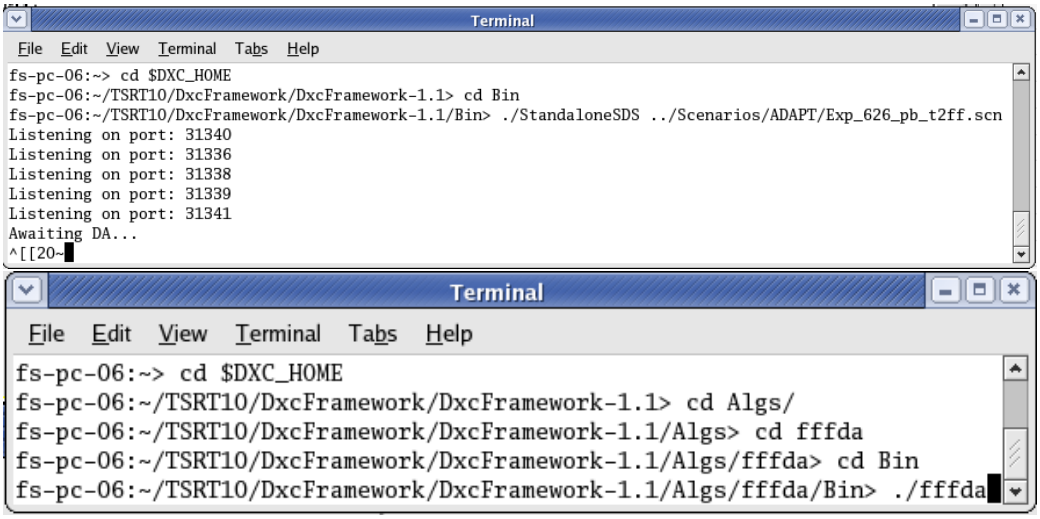

Figure 4: ... or run Scenarios by the DxC's standalone Scenario Data Source.

5 Changing algorithm parameters

The `algorithmparameters.h` (placed in the `fffda/Src` directory) contains various parameters used by the FFFDA to weight contradicting parameters and other common parameters used withing the algorithm. Contradictional parameters may be things such as thresholds for when to trigger a test quantity or when a load is thought to be in a stationary state. Changing these parameters will change the outcome from running scenarios in different ways. If a threshold is reduced one will get more detections of faults while running a scenario. But one will also experience an increased amount of false negative rates and less detection accuracy as more test quantities will detect false faults due to their altered thresholds.

If one tries to increase a competition score without adding any more test quantities, the algorithm parameters may be trimmed for increased performance. The FFFDA have been tuned to try to minimize false positive rates. But as false positive rates only is one of the metrics used to determine the score the final score may be increased by altered parameters.

The different parameters used in the FFFDA algorithm is described and explained within the `algorithmparameters.h` file.

6 Implementing a new test quantity

It's possible to implement new test quantities to increasing the detectability or isolability of the diagnosis algorithm. In order to implement a new test quantity some basic understanding of the FFFDA algorithm is provided below.

An overview of how the FFFDA diagnosis algorithm can be seen in figure 5. In the FFFDA object structure there is an class `objTestQuantity` which is a parent class for all test quantities. It takes sensor and command data as inputs and delivers a sub-diagnosis, containing information about what parts that might have a fault. By inheriting the test quantity parent class into new test quantities it gives the user a large freedom how to design thier test quantities. Using this object oriented structure of test quantities also increases the flexibility of the diagnosis algorithm as more test quantities can be added

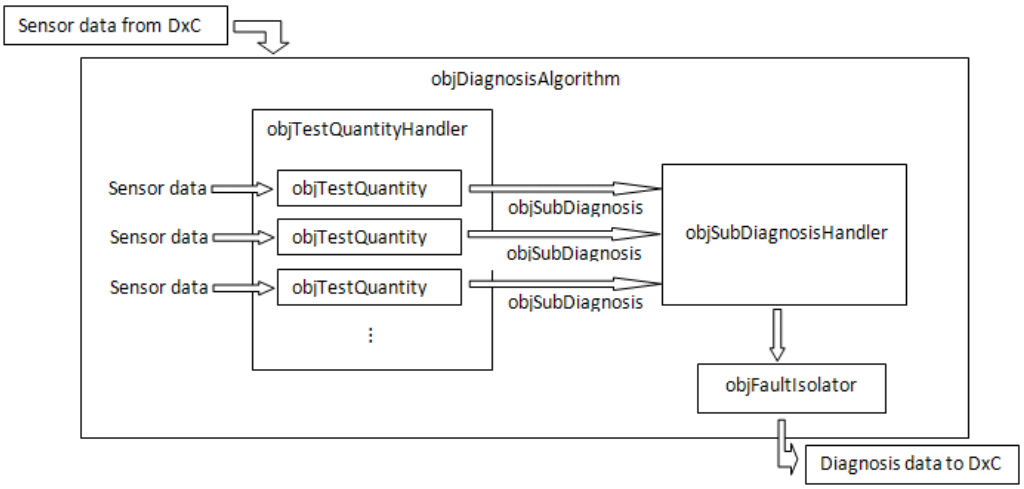

Figure 5: An overview of how the diagnosis algorithm will be implemented.

without changing the old ones.

Besides from the fact that every new test quantity has to be a subclass (by inheritance) of the parent class `objTestQuantity` there is a few other criterias that a new test quantity must fulfill. These criterias includes input and output data types and running of the test quantity.

Every test quantity has a function named `run()`. This function is executed by the FFFDA diagnosis algorithm each time new sensor data have been recieved. The `run()` function takes sensor and command data as input and delivers a sub diagnosis as output as described in figure 5.

As input test quantities uses the sensor and command data from a global sensormap and a global command map. Data is collected by invoking the corresponding function from the sensormap contained in a class called `objData`. For example, sensordata from the sensor "IT181" may be collected by invoking the function:

$sensorMap- > getDouble(string("IT181"));$

Some functions for filtering the data is provided withing the parent class `objTestQuantity`. What data is needed and if filtering is needed is based on the design of the test quantity.

Output from the new test quantity one will have to deliver a vector of pointers to `objSubDiagnosis` objects. This allows a test quantity to deliver more than one subdiagnosis if the class holds several different test quantities. The `objSubDiagnosis` objects is created in the test quantities and the pointer is sent to the algorithm which deletes them after they have been analysed. The test quantitiy does not need to take care itself of the objects it creates. It is also importent that each sent sub-diagnosis is created because if resuing a pointer to the same object then a fault will occur when the object is deleted.

A sub-diagnosis is a container for faults that is designed so that diagnosis candidates can be isolated inside the fault isolator (see 5). It contains multiple sub-diagnoses, each having a set of possible faults. The fault isolator returns possible candidates by using the isolation algorithm described in [Eri09].

Once a new test quantity have been created, it has to be instantiated in the diagnosis algorithm. The first step is to make sure that the new test quantity file is compiled by including it into the makefile of the system. The second step is to include the new test

quantity into the FFFDA diagnosis algorithm. This is done by including and instantiating the new test quantities in the test quantity handler (see 5). This is done in a special file called `initTestQuantities.cc`, as the number of test quantities can become quite large.

More information on how the different classes of the FFFDA diagnosis algorithm works can be found in the Technical Documentation[Eri09].

References

[dxp09] http://www.dx-competition.org/, september 2009.

[Eri09] Daniel Eriksson. Technical documentation, 2009.