Ontwerp & Analyse

Lectoraat Mobiliteitstechniek

Versie 1.1

Arnhem, 04-02-2014

Student …..... Bastiaan A. Stück
Student nr. …..... 496631
Stagedocent ..... Joost Kraaijeveld
Stagebegeleider …..... Jason v. Kolfschoten
Opdrachtgever …..... Jan Benders
Organisatie …... Lectoraat Mobiliteitstechniek

Inhoudsopgave

1. Inleiding

In dit document beschrijf ik mijn analyse en ontwerp voor de drie uitgevoerde iteraties in mijn stageproject. Per iteratie geef ik een korte beschrijving, waarna ik de analyse en het ontwerp kort en bondig weergeef.

Aangezien de derde iteratie de grootste in mijn project was, geef ik daar een iets langere beschrijving.

2. Iteratie 1 – Precompileren van C code

2.1. Korte beschrijving

De eerste iteratie besloeg het precompilerend maken van enkele C broncode bestanden. Het lectoraat ontwikkeld een plug-in bibliotheek voor Simulink, die het mogelijk maakt visueel te programmeren voor enkele van de door het lectoraat en partners gebruikte ECU's.

Plug-ins voor Simulink bestaan uit C broncode, die gebruikte wordt om het in Simulink geprogrammeerde visuele model te compileren naar machinecode, die vervolgens op een ECU geflasht kan worden.

Het lectoraat wilde de code in enkele van de bestanden graag gecompileerd uitgeven, zodat deze niet bekeken of bewerkt kon worden. Hiervoor heb ik een deel van de bibliotheek precompilerend gemaakt.

2.2. Analyse

Het deel van de bibliotheek dat precompilerend moest worden, bestond uit vier bestanden: Twee headers en twee C bestanden. Deze vier bestanden refereerden meerdere andere bestanden die ook niet precompilerend waren, enzovoort.

Er werden meerdere functies en classes uit andere bestanden gebruikte, alsmede defines uit een header die door Simulink werd gegenereerd.

2.3. Ontwerp

Voor het grootste deel kon het precompileren bereikt worden met simpel knip en plakwerk. Sommige functies konden gekopieerd of verplaatst worden zodat hier geen externe bestanden meer voor nodig waren. Voor externe classes kon gebruik worden gemaakt van *forward declaration* zodat deze bij de compiler bekend waren, en er bij het compileren geen afhankelijkheid was van de header waar ze vandaan kwamen.

De door Simulink gegenereerde header kon niet gewijzigd worden, aangezien deze elke compilatie opnieuw gegenereerd werd. In plaats daarvan geef ik de benodigde waarden als argumenten mee aan de functies waarin deze gebruikt worden.

3. Iteratie 2 – Debuggen van DAQlists

3.1. Korte beschrijving

De tweede iteratie was een inwerkings-iteratie in HANtune. Het betrof een geval van undefined behavior in HANtune's implementatie van DAQlists. Ik ging dit proberen te debuggen.

3.2. Analyse

Als eerste heb ik gekeken of ik de bugs kon reproduceren, dit bleek zo te zijn. Wanneer er meer dan 1 DAQlist was ingeladen en er een verbinding met een ECU actief was, zorgde het in of uitladen van DAQlists soms voor een exception, een crash, het verbreken van de verbinding of ander ongewenst gedrag.

3.3. Ontwerp

Ik heb met de debugger van Netbeans geprobeerd om de oorzaak van de onstabiliteit op te sporen. Al snel viel mij op dat het ongewenste gedrag minder vaak leek op te treden wanneer HANtune in debug modus werd uitgevoerd. Na drie keer proberen gebeurde het alsnog en na enige tijd door de code stappen leek het vast te blijven zitten in de loop die XCP berichten afhandelt. Dit zorgt naar mijn mening voor het verbreken van de verbinding omdat HANtune's XCP thread vastloopt en niet meer op de communicatie van de ECU reageert, wat een timeout tot gevolg heeft.

De exception trad volgens de stacktrace op in de XCP class, in de *processDAQ* methode. Deze methode gaf mij vlak voor het einde van het project meer problemen, vandaar dat ik vermoed dat deze twee bugs aan elkaar verwant zijn.

Ik heb binnen deze iteratie de precieze bug niet kunnen vinden, wel heb ik redelijk nauwkeurig kunnen vaststellen waar hij zich bevind. Ook heb ik een testdocument opgesteld voor het in de toekomst nauwkeuriger kunnen testen van deze functionaliteit.

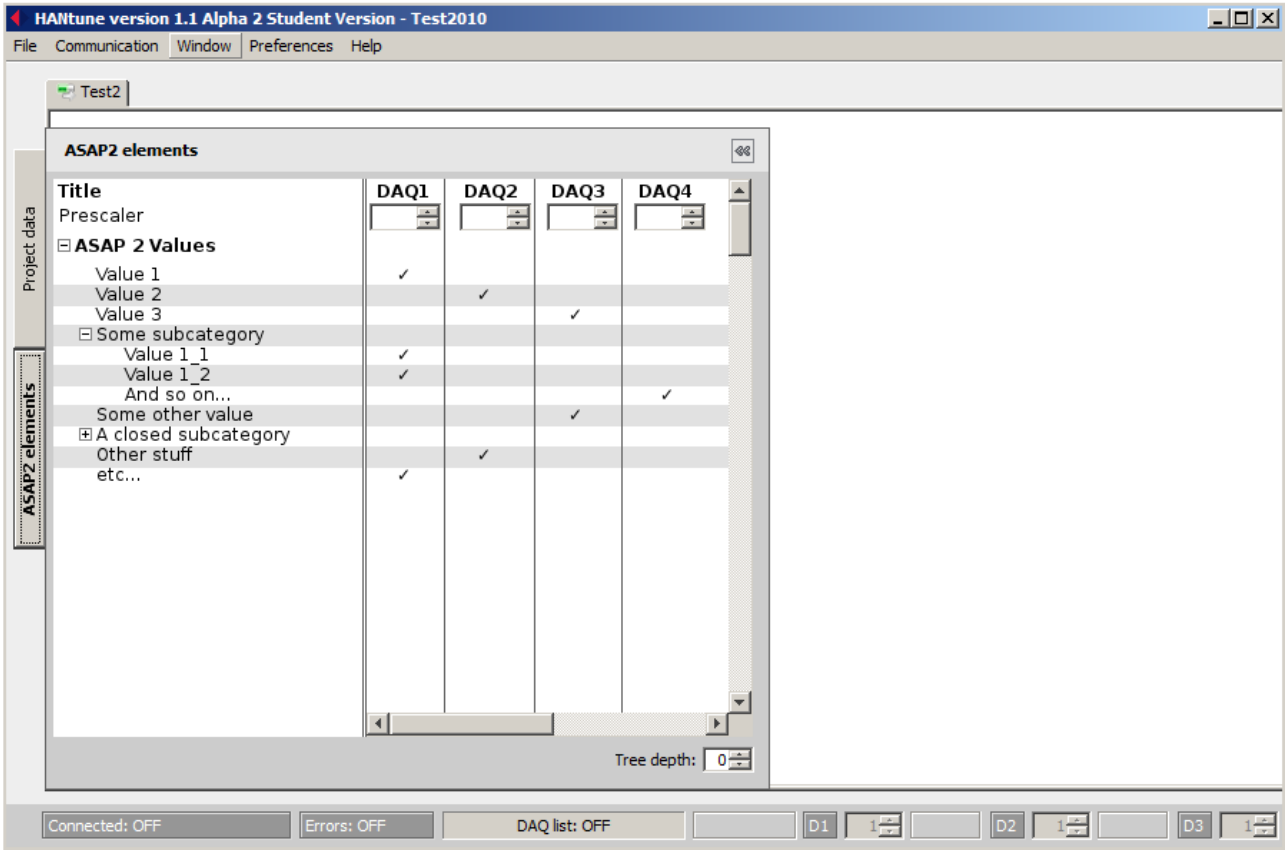
4. Iteratie 3 – GUI

4.1. Korte beschrijving

De derde iteratie was het hoofdonderdeel van mijn project, de nieuwe GUI voor HANtune. Deze GUI omvatte een betere weergave voor de DAQlist onderdelen, enkele updates aan de statusbalk en sidebar, en enkele toegevoegde functionaliteit en bugfixes in de rest van de GUI.

4.2. Analyse

Allereerst ben ik met de opdrachtgever in gesprek gegaan over de in de nieuwe iteratie te bereiken doelen. Hier kwam naar voren dat het vooral ging om een betere manier voor het weergeven van DAQlist elementen. Deze werden in een treeview per DAQlist weergegeven, wat zorgde voor 3 tabs met dezelfde treeview. Wij werden het al snel eens dat dit veel gebruikersvriendelijker kon. Ik kwam met het volgende ontwerp:



De eerste mockup voor de nieuwe gebruikersinterface.

De opdrachtgever stemde hier mee in, waarna ik overging met het verder ontwerpen van de nieuwe GUI.

4.3. Ontwerp

Voor het ontwerp maakte ik eerst een visueel prototype in Java. Dit had twee grote voordelen: Het gaf mij de mogelijkheid te experimenteren met de implementatie, en de opdrachtgever kon de nieuwe GUI in HANtune zien en direct aanpassingen en suggesties geven.

Met enkele kleine aanpassingen ging ik vervolgens de werkelijke GUI opzetten. Hieronder volgt een korte samenvatting van elke class die ik heb toegevoegd, op de volgende pagina vindt u een Class Diagram dat deze classes visualiseert.

ASAPGrid
Het hoofdonderdeel van de nieuwe GUI is de *ASAPGrid* class. Deze class doet dienst als een swing component dat het visuele deel rendert. Intern gebruikt de ASAPGrid class meerdere Jtable classes om de data weer te geven.

TreeGridNode
Voor elk DAQlist element wordt er een bijbehorende *TreeGridNode* instantie aangemaakt. Deze TreeGridNode bevat de benodigde data om dit element weer te geven in de ASAPGrid structuur.

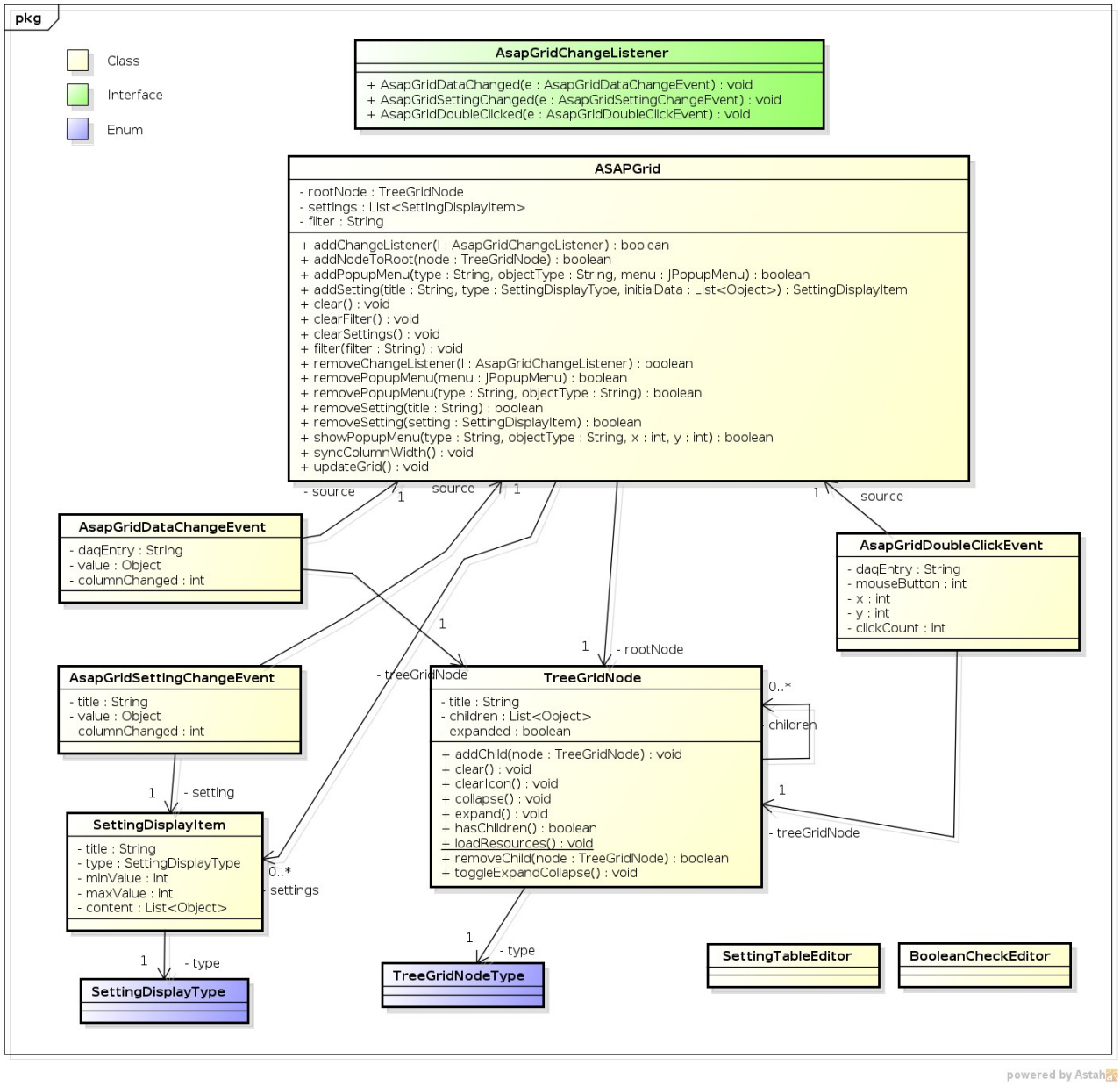
SettingDisplayItem
In het bovenste deel van het ASAPGrid wordt een header weergegeven, hierin kunnen settings worden opgenomen die per DAQlist kunnen worden gewijzigd. Net zoals de DAQlist elementen elk een achterliggende TreeGridNode hebben, heeft elke setting een achterliggende instantie van de *SettingDisplayItem* classe.

TreeGridNodeType & SettingDisplayType
Deze twee Enums duiden het type van een TreeGridNode of SettingDisplayType aan. Dit wordt zowel gebruikt om het desbetreffende item te renderen als om te bepalen wat voor gebruikersinteractie er mogelijk is met dit item.

Events & Listeners
De classes *AsapGridDataChangeEvent*, *AsapGridSettingChangeEvent* en *AsapGridDoubleClickEvent* worden meegegeven als argument aan functies van de *AsapGridChangeListener* interface.

De AsapGridChangeListener interface kan gebruikt worden om een listener op ASAPGrid te registreren, zodat deze listener op de hoogte wordt gebracht van wijzigingen in het ASAPGrid.

Editors
De *SettingTableEditor* en *BooleanCheckEditor* worden intern door ASAPGrid gebruikt om de gebruiker toe te staan data in de ASAPGrid te wijzigen. SettingTableEditor zorgt dat settings een editor krijgen toegewezen afhankelijk van hun type, en BooleanCheckEditor is verantwoordelijk voor de checkbox-achtige cellen van het DAQlist selectie gedeelte.



Het Class Diagram van mijn derde iteratie.