1 Bestand

Deze sectie bespreekt hoe een bestand wordt opgeslaan en hoe problemen bij het inladen van een semantisch incorrect programma worden afgehandeld. Daarna wordt ook besproken hoe ondersteuning voor meerdere talen mogelijk werd gemaakt in de appplicatie.

1.1 Keuze XML

Voor het opslaan van een programma moest een leesbaar formaat worden gekozen. Eerder in dit verslag werd al aangehaald dat een programma kan beschouwd worden als een boom structuur. De combinatie van deze twee aandachtspunten en het bestaan van betrouwbare verwerkings bibliotheken voor XML hebben ervoor gezorgd dat er gekozen werd voor XML.

In het analyseVerslag werd er gesteld dat data wordt opgeslagen naar meerdere files. Dit is echter een vervelende implementatie omdat de IDE dan meerdere files aanmaakt die aanpasbaar moeten zijn door de gebruiker. Voor de gebruiker is het simpelder indien er slechts èèn XML file was om aan te passen. De beschrijving van de XML voor alle geïmplementeerde blokken en XML DTD is achteraan beschreven in bijlage **??**.

Het volgende deel van deze sectie beschrijft de eigenlijke implementatie van het opslaan en inlezen van een programma. Als ook het afhandelen van een semantisch incorrect programma bij inladen.

1.2 Opslaan en inladen van een programma

Deze sectie beschrijft kort hoe een programma wordt opgeslaan en hoe het wordt ingeladen. Voor het inladen worden ook complexere zaken zoals het inladen van variabelen en functies besproken. Alsook de gevolgen van het inladen van een semantisch foutief programma.

1.2.1 Opslaan

Het opslaan van een progamma gebeurt in XML. Dit zal gebeuren zoals beschreven in Bijlage **??**.

Voor het opslaan wordt gebruik gemaakt van de boomstructuur gecreërd door de **modellen** (Sectie **??**) deze zijn centraal verzameld in de ModelCollection (Sectie **??**). Er is dus enkel nood aan een module die deze boom structuur afloopt en opslaat naar een bestand. Echter hebben we ervoor gekozen om het opslaan van een model los te koppelen van hun klasse. Dit om de klasse niet te vervuilen en de mogelijkheid te hebben om gemakkelijk een ander bestandsformaat te kiezen voor het opslaan van een programma. Er werd een Interface DataSaver geëimplementeerd die deze uitbreidbaarheid verzekerd.

Hierdoor duikt er in Java het probleem op waarbij er niet automatisch de functie wordt genomen met de meest specifieke klasse van een instantie. Om dit op te lossen werd er ook hier gebruik gemaakt van het Visitor patroon zoals beschreven in Sectie **??**

Voor het opslaan van **kostuums** van een klasse hebben we gekozen om de ingeladen afbeeldigen te kopiëren naar de folder waar de gebruiker het programma wenst op te slaan. Elke afbeelding wordt hernoemt naar een combinatie van de gekozen kostuums naam en de klasse waarbij deze behoort. De paden van de afbeeldingen worden relatief opgeslaan t.o.v de gekozen folder.

1.2.2 Inladen

Voor het inladen van het programma wordt geëist dat de XML correct gedefinieerd is. Bijlage **??** beschrijft deze definitie. De volgende paragrafen beschrijven het gedrag van de applicatie bij gebruik van incorrecte semantiek in verschillende delen van het bestand.

Events Als eerste worden de Events ingeladen. Aangezien deze overal in het programma kunnen gebruikt worden. Een Event moet uniek zijn. Bij het inlezen van een duplicaat Event zal dit Event worden overgeslaan. De members van een Event moeten uniek zijn. Bij het inlezen van een duplicate member zal deze member worden overgeslaan.

Bij het inlezen van blokken die gebruik maken van Event zal het bestaan van het Event gecontrolleerd worden.

Variabelen Het programma wordt per klasse ingeladen. Klasse naam moet uniek zijn. Duplicate namen worden overgeslaan. Voor een klasse worden eerst de membervariablen ingeladen ook deze moeten uniek zijn. Membervariabelen worden gestokeerd in een lijst. Bij het inlezen van handler of functie zal deze lijst worden aangevuld als er een definitie van een lokale variable wordt ingelezen.

Bij het inlezen van een referentie naar een variable wordt nagekeken als deze variablen gedefinieerd is. Zoja, dan wordt er een referentie naar deze variable gecreeërd. Anders zal de referentie niet worden ingeladen. Maar het programma wordt wel verder ingeladen. Voor elke functie of handler wordt er vertrokken vanuit de lijst met member variabelen.

Functies Functies moeten een unieke naam hebben. Bij een duplicate functie wordt deze overgeslaan.

Functie aanroepen Bij het tegenkomen van een functie oproep wordt deze ingeladen en ingevuld met de gewenste parameters en return. Echter worden alle

functie oproepen apart opgeslaan zodat deze later gekoppeld kunnen worden aan hun gewenste functie. Als na het inlezen van alle functies, de functie waarnaar de functie oproep refereert niet gevonden is, zal het programma niet worden ingeladen.

Handlers Als bij het inladen van een Handler een onbekend Event wordt gegeven zal deze handler geen Event worden toegekend. De handler en zijn body worden wel ingeladen.

Kostuums Bij ongeldig pad naar een afbeelding of het beschreven bestand is geen afbeelding zal de er geen afbeelding worden getoond in het programma maar zal het inladen gewoon verder gaan.

Instanties De naam van een instantie moet uniek zijn. Bij het inladen van een duplicate instantie zal deze worden overgeslaan.

Wires Duplicate wires worden overgeslaan. Bij het inladen van een wire tussen onbekende instanties of met een onbekend Event. Zorgt ervoor dat vanaf dat punt geen instanties of wires worden ingeladen.

1.3 Multilanguage IDE

De visuele programmeer IDE moet multilanguage zijn. De gebruiker moet kunnen wisselen tussen verschillende talen. Aangezien we Java gebruiken zijn we gaan kijken naar standaard klassen om multilanguage applicaties te maken. De oplossing is de ResourceBundle Class die Java aanbiedt [**?**].

Een ResourceBundle laad automatisch de nodige file gegeven een bepaalde filenaam (en eventueel een locale). Vervolgens kan via een getString("label") de tekst in de gevraagde taal opgevraagd worden. Een locale beschrijft een bepaalde taal, zo kan er een UK engels, en een US engels gemaakt worden [**?**]. Hieronder staat beschreven hoe de files voor verschillende talen genoemd moeten worden. De inhoud van deze files volgt een simpel formaat nl, key = tekst [**?**]. Voorbeeldcode: [**?**] language=Java public class Main public static void main(String[] args) // De constructor heeft 2 parameters: // een taal en een land Locale locale = new Locale("en", "UK"); // language.properties is een file ResourceBundle language = ResourceBundle.getBundle("language", locale); System.out.println(language.getString("label")); De verschillende files: language=XML language.properties language$_e$*n.properties* Een mogelijke inhoud van `language.properties`: language=XML label1 = een bepaalde tekst label2 = een andere tekst Een mogelijke inhoud van `language_en.properties`: language=XML label1 = a certain text label2 = another text

1.4 Omzetting van Modellen naar Views

Bij het gebruik van GUI zal elk view eerst bij reset de bestaande modellen opvragen en hiervoor views creeëren. Voor het inladen van een klasse zullen

de blokken ingeladen worden en gekoppeld worden aan views. Dit is mogelijk aangezien een View kan werken door views aan elkaar toe te voegen maar ook door enkel modellen te koppelen.

Het aanmaken van de Views wordt gedaan door de LoadClassViewFromModel klasse. Deze zal de blokken van een ClassModel koppelen aan hun views en op het IDEPanel plaatsen. Hier werdt er geen gebruik gemaakt van het Visitor patroon zodat een model niet vervuild wordt met de functionaliteit van view creatie.