Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling met

Titel: Interactieve visualisatie van user interface modellen
Richting: master in de informatica - Human Computer Interaction Jaar: 2008

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Ik ga akkoord,

SMOLDERS, Frederik

Datum: 5.11.2008

Interactieve visualisatie van user interface modellen

Frederik Smolders

promotor :
Prof. dr. Karin CONINX

co-promotor :
dr. Jan VAN DEN BERGH

Eindverhandeling voorgedragen tot het bekomen van de graad
master in de informatica Human Computer Interaction

universiteit hasselt

Abstract

Door onder andere de opkomst van Domain-Specific Modeling worden modellen in graafstructuur alsmaar belangrijker. Modellen hebben naast documentatiewaarde vaker en vaker ook betekenis voor de eigenlijke implementatie van het project.
Bijgevolg worden de grafische editors waarin modellen geconstrueerd worden ook steeds belangrijker. De aandacht hierbij gaat vaak naar de code generatie uit modellen of automatische layouts voor het te produceren model. Aan de usability van de grafische editors wordt echter zelden aandacht besteed.

Toch is de usability van deze editors geen te onderschatten aspect, voor zowel developers die modellen produceren om aan de hand daarvan code of applicaties te genereren, als voor gebruikers die modellen produceren enkel om hun documentatiewaarde.

Deze thesis bespreekt de basistaken die vaak terugkomen in het samenstellen van een model in graafstructuur, en probeert op basis van enkele scenario's de usability te verbeteren. Deze scenario's worden uitgediept aan de hand van de mogelijke methoden om het scenario te voltooien, en werkt zo progressief tot aan een methode die gemakkelijk bruikbaar en zo bondig mogelijk is. Het algemene principe in die voorgestelde methoden is het tijdig aanbieden van feedback aan de gebruiker, zodat deze altijd op de hoogte is van het effect van zijn acties op het model.
Dit principe en de bijhorende methoden werden als uitgangspunt gebruikt voor een nieuwe library, die het aanbieden van onder andere kant en klare acties aan de developer om zo eenvoudig mogelijk editors te kunnen implementeren, en gebruiksvriendelijke grafische editors voor de gebruiker in het algemeen, promoot. Voor beide gebruikersgroepen moet dit tijd en moeite besparen.

Na een evaluatie volgt de conclusie waarin de inhoud van deze thesis in het kort gerecapituleerd wordt, vergezeld van een kritische blik hierop.

Voorwoord

Deze thesis zou nooit tot stand gekomen kunnen zijn zonder de hulp en steun van enkele mensen. Ik wil mij in dit voorwoord even tot hen richten.

Eerst en vooral richt ik me tot mijn promotor Prof. dr. Karin Coninx en co-promotor/begeleider dr. Jan Van den Bergh, zonder wiens hulp deze thesis onmogelijk was geweest. In het bijzonder wil ik dr. Jan Van den Bergh enorm bedanken die mij in tal van afspraken en e-mails met zijn deskundig advies op de goede baan heeft geholpen, en mij met raad en daad gedurende de volledige thesis begeleid heeft.
Hier is een woordje van dank aan alle professoren en assistenten die ik gedurende de voorbije jaren aan de Universiteit Hasselt en het Expertise centre for Digital Media (EDM) heb mogen ontmoeten ook op zijn plaats.

Verder bedank ik natuurlijk mijn ouders voor de mogelijkheden die ze me gegeven hebben. Moest mijn vader dit nog meegemaakt mogen hebben, hoop ik dat hij fier zou zijn.

Ook bedank ik mijn vriendin Carola voor haar geduld, begrip en steun. Daarnaast bedank ik Davy, Tijl, en andere vrienden voor het telkens aanhoren van de laatste thesisperikelen.

Tot slot nog een extra ‘dank u’ voor Davy en mijn redster in nood, Carola, voor het opofferen van hun tijd voor een laatste spellingscontrole.

Inhoudstabel

Lijst van figuren

Lijst van tabellen

1 Inleiding

1.1 Thesis context

Domain-Specific Languages (DSL), en de bijhorende Domain-Specific Models (DSM), worden alsmaar vaker gebruikt [Bézi06]. Zoals de naam al doet vermoeden is een DSL een programmeertaal die speciaal ontwikkeld is voor een bepaald domein of voor een bepaalde taak.
Domain-Specific Modeling modelleert een applicatie aan de hand van domein specifieke concepten. Deze DSM is gebouwd op basis van een bijhorende DSL, er bestaat dus een mapping tussen DSM concepten en DSL code. Een developer kan dan modellen produceren volgens de Domain-Specific Modeling Language, dit is als het ware de grammatica van het model. Uit die geproduceerde modellen kan vervolgens code, of in sommige gevallen een volledige applicatie, gegenereerd worden [DSMF08].

Dit is slechts één voorbeeld van het hedendaagse gebruik van modellen, eentje dat alsmaar populairder wordt.
Daarnaast zijn er natuurlijk nog tal van andere toepassingen waarin graaf modellen gebruikt worden. Naar alle waarschijnlijkheid zal dit aantal mettertijd enkel nog uitbreiden en zullen modellen een nóg belangrijkere rol gaan spelen, zoals duidelijk wordt in onder andere [Léde01], [Choc03] en [Ehri05].

Duidelijk is dat modellen alsmaar essentiëler worden, zij het voor DSM, of zij het voor design doeleinden of documentatiewaarde. Deze modellen moeten het leven van de designer vaak vergemakkelijken, niet nog complexer maken. Daarom moet speciale aandacht gaan naar de grafische editors die gebruikt worden om die modellen samen te stellen. Immers, wanneer die editors niet gemakkelijk bruikbaar zijn, zal dit zijn effect hebben op de tijd die besteed wordt aan het maken van een model, en waarschijnlijk ook op de gemoedstoestand van de gebruiker. Dit zou hun voordeel deels teniet doen.

Deze thesis richt zich in het bijzonder tot de usability van de grafische model editors, en daarnaast op de ontwikkeling van die editors. Het is natuurlijk ook van belang dat deze grafische editors zo eenvoudig en snel mogelijk geïmplementeerd kunnen worden.
Hierbij worden code generatie, zoals bij DSM, en andere aspecten achterwege gelaten, om zich zo vooral toe te spitsen op hoe een gebruiker een model samenstelt in een grafische editor, en hoe we de gebruiker kunnen ondersteunen in deze activiteit.

1.2 Thesis opbouw

In hoofdstuk 2 worden enkele frameworks besproken die gericht zijn op het creëren van grafische model editors of visualisaties. Hierbij houden we ook rekening met hoe moeilijk en tijdsrovend het voor de developer is om zulk een editor te implementeren. Het hoofdstuk wordt afgesloten met een korte evaluatie en verantwoording voor het ontwikkelen van een nieuwe library.

Hoofdstuk 3 analyseert vaak voorkomende activiteiten tijdens het opbouwen van een model. Naast de taken die de basis vormen om een model te kunnen samenstellen, worden ook enkele scenario's voorgesteld. Een scenario kan op verschillende manieren voltooid worden. Deze methoden worden geanalyseerd en geëvalueerd, om zo progressief tot een methode te komen die zowel op vlak van uitvoeringstijd als gebruiksvriendelijkheid een verbetering moet betekenen ten opzichte van zijn voorgangers.

De methoden die voortvloeien uit hoofdstuk 3 worden in hoofdstuk 4 gecombineerd, waar zij de fundamenten zullen vormen voor een nieuwe library. De opbouw van deze library wordt kort besproken, en extra aandacht wordt besteed aan de eigenlijke implementatie van de besproken methoden in hoofdstuk 3. Als afsluiter van dit hoofdstuk worden twee voorbeelden bekeken die gebouwd zijn aan de hand van deze library.

Hoofdstuk 5 stelt een korte evaluatie voor van zowel het library gedeelte als van de usability van een typische grafische model editor gecreëerd met behulp van die library.

Tot slot volgt nog de conclusie van deze thesis, met een bondige samenvatting van de inhoud vergezeld van een kritische blik hierop.

2 Gerelateerd Werk

2.1 Inleiding

In dit hoofdstuk wordt gerelateerd werk besproken. We bekijken eerst bondig enkele frameworks die zich richten tot het creëren van grafische model editors. Hier wordt in het bijzonder aan Visual Library 2.0 extra aandacht geschonken, aangezien de library die in deze thesis voorgesteld wordt hierop gebaseerd is.

Er kunnen betreffende deze frameworks hoofdzakelijk twee richtingen onderscheiden worden:

- Frameworks waarin de developer gebruik makend van code zijn editor implementeert.
- CASE-tools waarin de developer vaak op basis van modellen die hij samenstelt een grafische editor kan genereren.

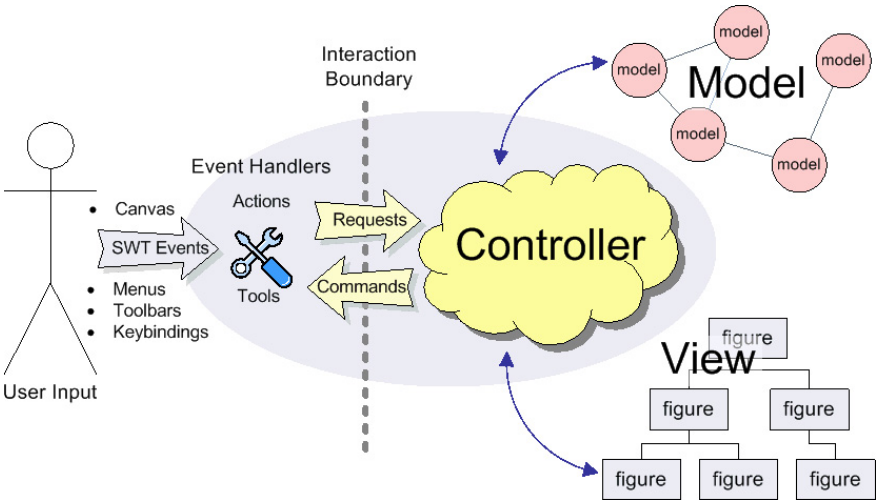
Deze thesis concentreert zich op de usability van de grafische editor waar de eindgebruiker mee in contact zal komen. Daarnaast wordt ook in het kort bekeken hoe snel en gemakkelijk het voor de developer gemaakt wordt om zulk een grafische editor te implementeren of te genereren.

Op het eind van dit hoofdstuk volgt een bondige algemene evaluatie van de bestaande frameworks. Hieruit volgt een verklaring voor de nood aan de ontwikkeling van een nieuwe library, Tlibrary.

2.2 GEF

Graphical Editing Framework [GEF08a] [GEF08b] [Ecli08a] is een framework, als onderdeel van het Eclipse project [Ecli08b], met als doel gemakkelijk een graphical editor in Eclipse te kunnen creëren. Dit framework is opgebouwd volgens de model-view-controller architectuur:

- Model: alle data die bewaard gaat worden.
- View: alles wat zichtbaar is voor de gebruiker, de grafische representatie van het model.
- Controller: dit is de link tussen het model en de view, en beheert alle events die gebeuren op het model. In GEF wordt de controller EditPart genoemd. Elk model object heeft gewoonlijk een controller. Deze EditPart is hetgeen waar de gebruiker mee interageert [GEF08b].



Figuur 1 - High-level view van GEF [Ecli08a]

Gebouwd bovenop Draw2d, dat zich concentreert op het tekenen van figuren en de layout ervan, moet GEF het mogelijk maken modellen te tonen aan de hand van Draw2d figuren, en ze gemakkelijk te kunnen editeren met behulp van muis en keyboard.

Door het gebruik van Draw2d kunnen aanwezige widgets als labels, borders, etc. gebruikt worden om de view te specificeren. Complexere figuren daarentegen moeten gecombineerd worden uit meerdere bestaande figuren, of eventueel zelf geïmplementeerd worden.

GEF biedt verder via zijn PaletteViewer een gemakkelijk te implementeren palette aan waarin de gebruiker van de gemaakte grafische editor tools kan selecteren of een object rechtstreeks vanuit de palette naar het diagram kan slepen, mogelijk gemaakt door de geïntegreerde support van drag-and-drop.
Hiernaast wordt tevens een Property Sheet door GEF aangeboden, waarin de eigenschappen van geselecteerde items getoond kunnen worden [GEF] [Ecli08a].

Basistaken zoals selectie, creatie, verwijdering en verplaatsen van objecten zijn door het framework ondersteund. Toch blijft het creëren van een model editor een relatief complexe opdracht die een grondige kennis van dit framework vereist.

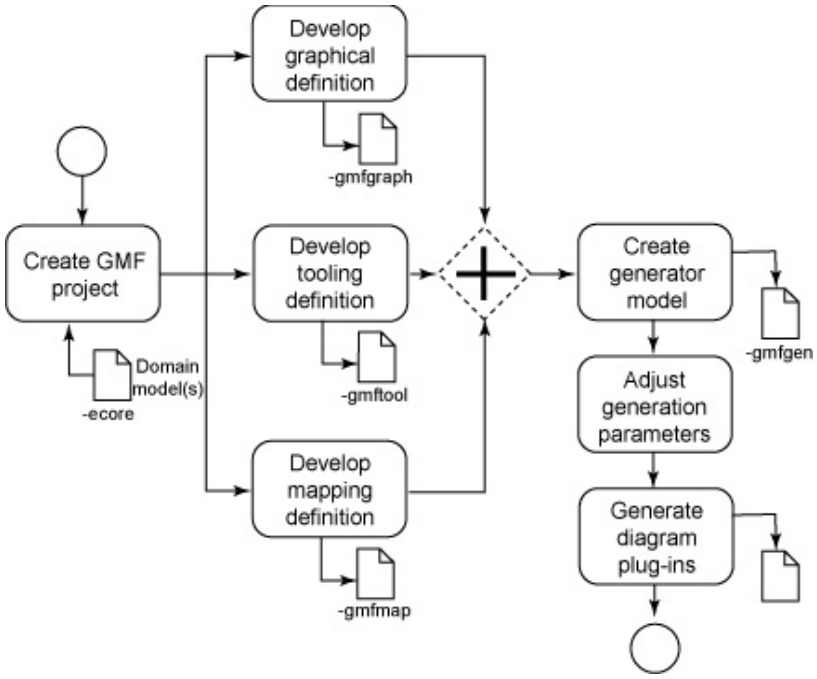
2.3 GMF

Graphical Modeling Framework [GMF08a] is ook een onderdeel van het Eclipse project en verenigt EMF en GEF, die ook beide deel uitmaken van het Eclipse platform [Ecli08b], in één project. GEF is hier boven reeds besproken, maar om GMF te begrijpen bekijken we ook EMF even.

EMF, oftewel Eclipse Modeling Framework, is een modeling framework en code generation facility om tools en andere applicaties te bouwen, gebaseerd op een gestructureerd data model [EMF08]. EMF kan dus op basis van een modelspecificatie in XMI (XML Metadata Interchange) bruikbare code en een basis editor voor het gespecificeerde model genereren.

Dit is vooral handig voor developers die al uitgebreid modellen opstellen voor hun projecten. Deze modellen zijn dan, naast hun documentatiewaarde, ook van betekenis voor de eigenlijke implementatie, waardoor het werk van de developer enigszins verlicht wordt.

Met behulp van GMF kunnen dus grafische editors voor Eclipse gecreëerd worden. Dit gebeurt aan de hand van enkele modellen: een Domain Model, een Graphical Definition Model, een Mapping Model, en eventueel een Tooling Definition Model [GMF08b].



Figuur 2 - GMF overzicht [GMF08c]

Een GMF project is dus minimum de combinatie van volgende drie modellen [GMF08b]:

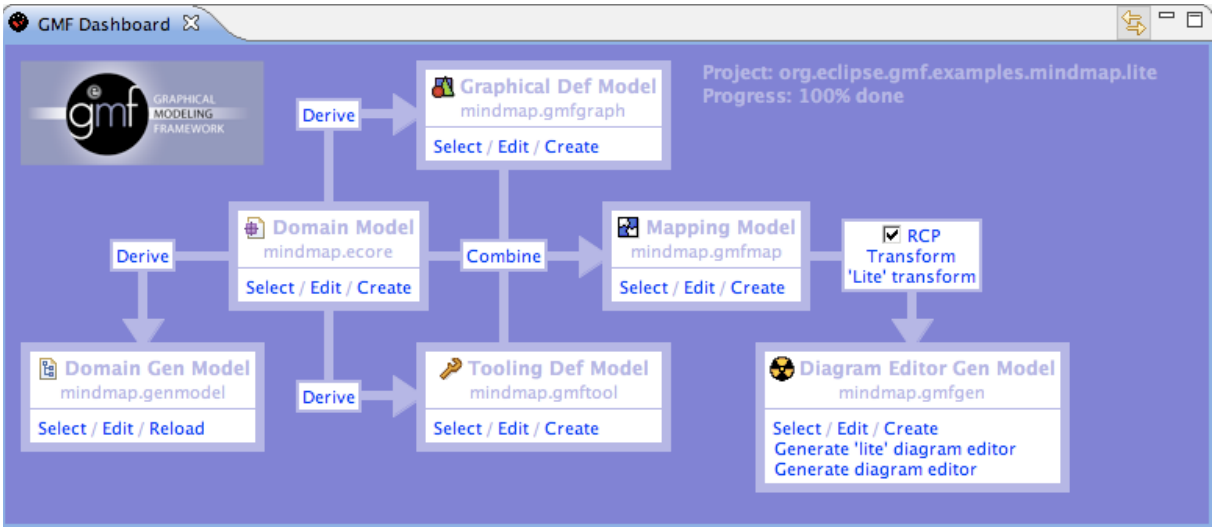
- Het Domain Model is als het ware een grafische omschrijving van alles wat het te maken model moet inhouden, dit definieert het model. Dit model kan geïmporteerd worden uit een EMF project.

- Het Graphical Definition Model bevat alle grafische objecten, zoals de nodes, links, figuren, etc., die samen de grafische kant van het model zullen bepalen.

- Het Mapping Model verzorgt de relatie tussen het Domain Model en het Graphical Definition Model.

Indien gewenst kunnen in het optionele Tooling Definition Model palettes en menu's voor de grafische editor gedefinieerd worden. Hiervoor is bovendien een handige wizard voorhanden.

Met behulp van het GMF dashboard, zoals in figuur 3, kan relatief eenvoudig het overzicht in de creatie van de modellen gehouden worden. Dit dashboard heeft dan ook een opvallende gelijkenis met figuur 2.



Figuur 3 - GMF dashboard [GMF08b]

Wanneer alle modellen compleet zijn, kan op basis van de combinatie hiervan een Generator Model gegenereerd worden, en kan de gegenereerde grafische editor geopend worden.
Eventueel kan dan naderhand de gegenereerde code nog aangepast worden om de grafische editor volledig naar eigen hand te zetten [GMF08c] [GMF08d].

Een grafische editor kan in GMF dus gemaakt worden op basis van enkele grafische modellen. Hiervoor is geen programmeerwerk met code voor nodig, tenzij voor meer geavanceerde features en enkele situaties waar wel wat manueel werk verricht moet worden. Vaak gebruikte objecten zoals onder andere tools, menu commando's, toolbars, common properties, zijn standaard beschikbaar in GMF.
Het visueel aanmaken van de benodigde modellen om de editor te genereren, verlicht de developer van de complexiteit van GEF, waar grondige kennis van het framework nodig is.

2.4 JGraph

JGraph [Jgra08] is een bekende graafvisualisatie library voor Java. In deze library worden nodes, of vertices, en edges ook vaak cells genoemd. JGraph is de naam van het eigenlijke Java component dat in java programma's gebruikt kan worden om grafen te visualiseren. Het is een extensie van JComponent en is grotendeels gebaseerd op JTree [Alde01].

Ook JGraph ondersteunt basis interacties zoals onder andere het aanmaken en verwijderen van verbindingen, bewegen van nodes, en zelfs cloning. Ook drag-and-drop van een object uit een andere source wordt ondersteund [Jgra08]. Voor het bouwen van een model editor is ook hier weer grondige kennis nodig van dit complexe framework.

2.5 Microsoft DSL Tools voor Visual Studio 2005

Met Microsoft DSL, oftewel Domain-Specific Language Tools [DSLT08], kan men domain-specific languages definiëren, grafische designers ervoor maken, en code generators definiëren [Pele06]. Via een wizard kan een DSL gecreëerd worden. Het resultaat is een grafische designer in Visual Studio die de gespecificeerde DSL gebruikt. Vanuit modellen die men in deze editor maakt, die op zich natuurlijk gebaseerd zijn op de gebruikte domain-specific language, kan men bruikbare code genereren.

De focus ligt hier duidelijk op domain-specific languages en de ontwikkeling daarvan, niet zozeer op de bijhorende grafische model editor en de usability daarvan. Laat nu dit laatste aspect juist van belang zijn in deze thesis.

2.6 Visual Library 2.0

2.6.1 Algemeen

Visual Library 2.0 [NetB08b] is de opvolger van Graph Library 1.0 en is een general-purpose en graaf georiënteerde visualisatie library in Netbeans [NetB08a]. Gebruik maken ervan kan zowel in Netbeans als in een stand-alone java applicatie.

2.6.2 Scene

De hoofdcomponent is de scene, dit is het eigenlijke model. De programmeerstijl is gelijkend op die van Swing. Aan de scene worden widgets toegevoegd die gestructureerd zijn in een boomhiërarchie, waarin de scene de root is die alle visuele data bijhoudt.
Om de scene te tonen in een JComponent moet er een view van deze scene aangemaakt worden.

LabelWidget en ImageWidget, om respectievelijk teskt of figuren te tonen, zijn gebaseerd op de hoofdklasse Widget, wat een primitief visueel element voorstelt. Door een compositie van een aantal van deze widgets zijn relatief complexe elementen op te stellen [NetB08c].

2.6.3 Acties

Een widget kent uit zichzelf geen enkele actie, hiervoor moeten WidgetActions op de widget gelegd worden. Een WidgetAction vereist vaak een decorator en/of provider als parameter.
Een decorator specificeert de grafische kant van hulpmiddelen die tijdens een actie nodig zijn, terwijl een provider het eigenlijke gedrag van de actie bepaalt.
Voor de meeste acties zijn kant en klare implementaties hiervoor beschikbaar. Voor meer complexe acties of om acties volledig naar eigen hand te zetten, zullen echter eigen providers of decorators geïmplementeerd moeten worden [NetB08c].

Bij het toevoegen van WidgetActions moet rekening gehouden worden met de volgorde daarvan. Als een widget bijvoorbeeld een MoveAction heeft, consumeert deze actie alle events vanaf dat de muisknop wordt ingedrukt op het widget in kwestie, tot de muisknop losgelaten wordt. Wanneer bijvoorbeeld een SelectAction toevoegd wordt na een MoveAction, zal de MoveAction altijd de mouse-events consumeren, en zal de SelectAction bijgevolg nooit bereikt worden. Voorzichtigheid is hier dus geboden [NetB08c].

2.6.4 Verbindingen

Om widgets met elkaar te kunnen verbinden, bestaat het ConnectionWidget. Deze widget heeft een source en target anchor om te definiëren welke de bron en doel widgets zijn. Het pad dat de verbinding volgt, wordt bepaald door zijn router. Standaard is dit de DirectRouter, die een rechte lijn trekt van de bron naar het doel, maar deze is ook aan te passen of te veranderen naar een OrthogonalSearchRouter, die zijn pad bepaalt aan de hand van andere widgets.

Net zoals aan gewone widgets kunnen ook aan ConnectionWidgets kinderen worden toegevoegd. Zo zijn eenvoudig labels aan verbindingen te plakken [NetB08c].

2.6.5 ObjectScene

Een boomhiërarchie van widgets is helaas te complex om mee te werken in een model, het overzicht hierin houden zou teveel tijd en moeite in beslag nemen. Het model voorstellen door objecten zou een abstractie toevoegen waardoor het model gemakkelijker te controleren is. Daarom biedt Visual Library 2.0 ook een ObjectScene aan.
Hierin wordt een model voorgesteld door objecten, die op zich kunnen bestaan uit meerdere widgets, in tegenstelling tot de gewone scene die alleen widgets kent. De ObjectScene verzorgt de mapping tussen de widgets en de objecten, maar de developer moet zelf de widgets aanmaken en als object registreren [NetB08c].

2.6.6 GraphScene

Naast de ObjectScene biedt Visual Library 2.0 ook specifieke ondersteuning voor modellen in graafstructuur, en dit door middel van de GraphScene en GraphPinScene die beide afgeleiden zijn van de ObjectScene.
In de GraphScene wordt een model voorgesteld door nodes en edges, terwijl de GraphPinScene dit door nodes, edges en pins doet. In de GraphScene worden edges verbonden met een bron en doel node, in de GraphPinScene worden deze verbonden met een bron en doel pin, die altijd bij een node horen [NetB08c].

Een node kan worden toegevoegd aan het model door addNode, een edge door addEdge. Deze methoden moeten zelf geïmplementeerd worden en handelen de grafische kant af. Hier maken we dus de gewenste widgets aan, met eventueel de nodige WidgetActions, en sturen deze terug via een return statement.
Eens de implementatie van de nodige methodes voor nodes en edges voltooid is, moet de developer zich niet meer bezig houden met de grafische kant van de nodes en edges [NetB08c].

Edges kunnen verbonden worden door de methoden setEdgeSource en setEdgeTarget. Door een node mee te geven met deze methoden wordt de edge aan deze node verbonden, zonder dat we rekening moeten houden met de anchors van de bijbehorende ConnectionWidget. Hiervoor moet echter deze methode eerst geïmplementeerd worden [NetB08c].

2.6.7 Custom development

Verschillende aspecten van Visual Library 2.0 kunnen uitgebreid worden door zelfgemaakte componenten. Zo kunnen we bijvoorbeeld zelf widgets, acties, routers, anchors etc. maken. Dit laat ons toe de library naar eigen hand te zetten [NetB08c].

2.7 Overige

Tom Sawyer Software's Graph Editor Toolkit, of kortweg GET, is een toolkit die het tonen van data in een graafstructuur vergemakkelijkt [Dogr02], door onder andere het aanbieden van zooming, drag and drop, etc. De toolkit bevat ook een sterke integratie van layout algoritmes, GET is dan ook gebaseerd op het graaf en teken model, en de layout technologie van Graph Layout Toolkit, evenzeer van Tom Sawyer Software [TomS08]. Verder omvat deze toolkit gespecialiseerde features zoals multiple windows en nesting van modellen.

GraphViz [Grap08] is nog zo een toolkit die zich specifiek op graafvisualisatie layouts toespitst. De layouts uit deze toolkit kunnen in verscheidene applicaties die nood hebben aan graaf layouts gebruikt worden [Ells03].

DiaGen is een diagram Generator die op basis van een modelspecificatie een volledige grafische editor voor dat model kan genereren, met automatische support voor direct manipulation [Mina95].

Ook in Piccolo [Picc08], bekend om zijn zoomable user interfaces, zijn grafische model editors denkbaar. Toch ontbreekt hier geïntegreerde support specifiek voor graaf modellen, zo heeft piccolo geen notie van nodes of edges, waardoor deze features zelf geïmplementeerd moeten worden, wat bijgevolg weer extra tijd vraagt.

2.8 Besluit

Algemeen kan gesteld worden dat frameworks zonder editor, waar de ontwikkelaar zijn grafische model editor via code moet implementeren, een grondige kennis van het gebruikte framework vereisen. Veelal worden er wel standaard implementaties voor de basistaken in het model aangeboden, maar deze aanpassen of volledig nieuwe acties aanmaken is vaak geen sinecure.
De ondersteuning van een palette en eventueel een properties panel kan de developer enigszins tijd besparen. Echter, de implementatie ervan vereist vaak zelfs dan nog teveel werk.

Frameworks die een editor aanbieden om de gewenste grafische editor met onderliggend gedefinieerd model te implementeren, zoals de DSL Tools van Microsoft, verminderen de nood aan parate kennis van het framework. [Pele06] toont echter aan dat de editors gebruikt in DSL Tools en GMF zelf moeilijk te gebruiken zijn, dit terwijl een editor juist het werk van de developer moet verlichten. Ook de gegenereerde grafische editors blijken onvolledig te zijn, en moeten dus vaak nog aangepast en uitgebreid worden.

Verder is er weinig literatuur of onderzoek te vinden waaruit blijkt dat bestaande modeling frameworks oog hebben voor het verbeteren van het gebruikersgemak van de grafische editors die eruit voort moeten vloeien. De meeste frameworks richten zich wel op automatische layout van de graaf, wat in zekere zin ook het gebruikersgemak van de editor zal verhogen. Anderen richten zich dan weer op het genereren van een grafische editor op basis van een domain model of meerdere modellen.
Maar wij zullen in deze thesis wél de focus leggen op de usability van die grafische editors die de gewone gebruiker onder handen zal krijgen, een vaak vergeten of onderontwikkeld aspect van huidige modeling frameworks.

In deze thesis wordt Tlibrary voorgesteld, een library die aan deze noden een invulling probeert te bieden. Het volgende hoofdstuk beschrijft de basistaken die in een grafische editor voor een model van toepassing zijn. Verder proberen we in dat hoofdstuk op basis van enkele scenario's nieuwe methoden voor te stellen om deze scenario's zo efficiënt en gebruiksvriendelijk mogelijk tot een goed einde te brengen. Dit doen we aan de hand van een grondige analyse van bestaande methoden, en een verantwoording voor de nieuwe methode. Op basis van die nieuwe methoden en de filosofie erachter werd de Tlibrary samengesteld. Deze library en zijn features worden in hoofdstuk 4 besproken.

Naast gebruikersvriendelijkheid van zijn grafische editors, die deze nieuwe methoden automatisch implementeren, probeert Tlibrary ook zo beknopt mogelijk te zijn voor de developer. Zo is bijvoorbeeld een drag-and-drop palette eenvoudig te implementeren, om zo de creatie van een grafische editor en het bijhorende model zo duidelijk en snel mogelijk te maken.

3 Usability en efficiëntie in grafische model editors

3.1 Inleiding

[Camp06] stelt dat bij het maken van een design slecht 25% van de tijd wordt besteed aan het effectief creëren ervan, en 75% aan het aanpassen van het gemaakte design. Willen we een goede diagram of model editor maken, is dit een gegeven waar rekening mee gehouden moet worden.
Bij het maken van een model doet de gebruiker constant al dan niet grote aanpassing aan zijn reeds gemaakt model. Die aanpassing kan volgen op een verandering van het mentaal model van de gebruiker, door een foutieve actie gemaakt in het model, of door een onvoorziene reactie van het programma op een actie van de gebruiker.

Vorig gegeven stelt dat een designer ruim 3-maal zoveel tijd besteedt aan het aanpassen van een design dan aan het creëren ervan. Het is dan ook een logisch gevolg dat in het aanpassen van een model de meeste tijdswinst behaald kan worden. Maar hoe bereiken we dit? Dit hoofdstuk biedt een antwoord op enkele veel voorkomende scenario's.

Eerst en vooral wordt een bondige uitleg gegeven over de gebruikte terminologie in dit hoofdstuk. Dit is immers parate kennis om dit hoofdstuk, en waarschijnlijk deze hele thesis, te begrijpen. Deze uitleg vinden we terug in 3.2.

Nadien bekijken we de basistaken die uitgevoerd kunnen worden met een grafische model editor. Dit onderdeel geeft een context aan de scenario's die erna zullen volgen. Immers, als we niet weten hoe een node wordt toegevoegd, hoe kunnen we dan bijvoorbeeld een node toevoegen aan een bestaande edge?
Voor we complexe taken als deze kunnen analyseren moeten we eerst een beeld hebben van hoe de basistaken verlopen bij het aanmaken en aanpassen van een model.
Daarom bekijken we volgende basistaken:

- Een node toevoegen.
- Een edge toevoegen.
- Een node verplaatsen.
- Een edge verplaatsen.
- Een node of edge verwijderen.

Per taak bekijken we wat de mogelijkheden en de alternatieven zijn. Deze taken kunnen beschouwd worden als de minimale functionaliteit van een model editor, en vinden we onder 3.3.

Naderhand wordt op basis van enkele scenario's bekeken hoe we de usability van deze editors kunnen verbeteren.
We bekijken verschillende punten waarop de huidige editors voor verbetering vatbaar zijn. Per punt analyseren we de bestaande methoden, om zo tot een aangepaste methode te komen die de gebruiker een verhoogde usability biedt.
De verzameling van deze nieuwe methoden moet komen tot een drastische verbetering in zowel uitvoeringstijd als in gebruiksvriendelijkheid.

Dit onderdeel is de essentie waar dit hoofdstuk rond draait. De punten die hier bekeken en geanalyseerd worden zijn:

- Het toevoegen van een node in een bestaande edge.
- Het toevoegen van een item aan een node.
- Het verwijderen van een item uit een node.

Deze zijn respectievelijk te vinden onder punt 3.4, 3.5 en 3.6.

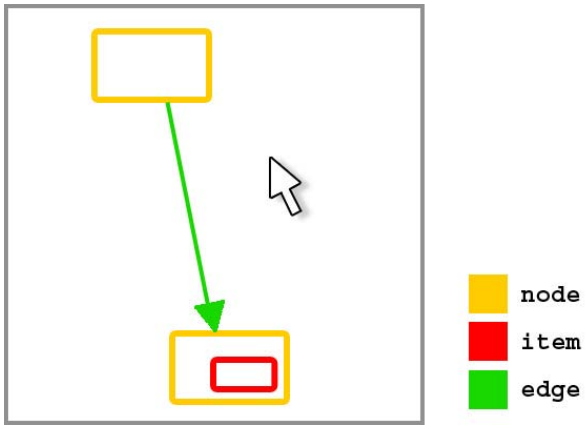
Uiteindelijk combineren we de nieuwe methoden tot een meer algemene visie, waarin we als het ware een preview krijgen van hoe een model eruit zal zien na een aanpassing. Dit gebeurt in punt 3.7.

3.2 Terminologie

Wanneer we het over een model hebben, vallen vaak de termen ‘node’ en ‘edge’. Een node is een knoop of knooppunt in een graaf (oftewel, ons model), een edge is een verbinding tussen nodes. Samen vormen ze een graafstructuur en dus de hoofdstructuur van het model.

Om een betekenis mee te geven aan een node, kan deze aangepast worden. Bijvoorbeeld door het toevoegen van objecten. Zulke objecten die aan een node toegevoegd kunnen worden zullen in deze thesis ‘items’ genoemd worden.

Figuur 4 toont twee nodes, een edge, en een item, met elk een andere kleur om het onderscheid te verduidelijken. In de figuren die in dit hoofdstuk volgen zijn deze kleuren niet meer van toepassing.



Figuur 4 - Terminologie

3.3 Basistaken

Bij het opstellen van een model komen vaak dezelfde basistaken kijken, waaronder de volgende als het absolute minimum kunnen beschouwd worden:

- Een node toevoegen.
- Een edge toevoegen.
- Een node verplaatsen.

- Een edge verplaatsen.
- Een node of edge verwijderen.

3.3.1 Een node toevoegen

Een nieuwe node toevoegen kan doorgaans op 2 manieren, afhankelijk van welk soort interactie er gebruikt wordt:

- Drag-and-drop: een node uit een palette slepen naar de gewenste plaats op het model en deze daar droppen.
- Point-and-click: we klikken in een palette de node aan om vervolgens op de plaats in het model te klikken waar we deze node willen plaatsen.

Volgens [Camp06] hebben gebruikers eerder een afkeer van een tool palette, die een point-and-click interactie hanteert, waarna de developers een drag-and-drop palette hebben geïmplementeerd in hun TaskSketch tool. Dit vonden gebruikers dan weer veel bruikbaarder.

De drag-and-drop methode krijgt dus de voorkeur op een point-and-click methode. Een mogelijke verklaring hiervoor is dat de gebruiker meer het gevoel heeft effectief een node uit de palette te nemen, uit een oneindige verzameling nodes, en deze zelf naar het model te verplaatsen.
We veronderstellen dan ook dat in de rest van deze thesis telkens onze voorkeur zal uitgaan naar een drag-and-drop variatie van een bepaalde methode, als deze bestaat natuurlijk.

Bij de drag-and-drop methode kunnen we verder nog een onderscheid maken op basis van wanneer de node zichtbaar wordt. Gebeurt dit bij het droppen van de node op het model, waarna de node op die plaats aangemaakt wordt? Of wordt de node letterlijk uit de palette gesleept, wat wil zeggen dat de node al zichtbaar is en aan het model toegevoegd wordt op het moment dat de sleepactie de palette verlaat en het model binnenkomt?
Beiden geven hetzelfde resultaat, namelijk een node die na de drop op de gewenste plaats staat, enkel krijgt de gebruiker tijdens de uitvoering van de sleepactie een andere feedback. In het tweede geval krijgt de gebruiker immers onmiddellijk te zien hoe de eigenlijke node eruit zal zien in het model. De naam en het icoon in de palette moeten vanzelfsprekend zo duidelijk en ondubbelzinnig mogelijk zijn, maar toch zal dit nooit tonen hoe de node eruit zal zien indien deze toegevoegd wordt aan het model. Verdere uitleg of een figuur in een tooltip kan hier meer duidelijkheid brengen.
In het eerste geval krijgen we het nieuwe beeld van het model, inclusief de nieuwe node, pas wanneer men de drop uitvoert. In het tweede geval krijgen we deze feedback tijdens de sleepactie al, de drop bevestigt enkel op het einde van de sleepactie de gewenste locatie van de nieuwe node.
Bovendien geven beide gevallen dezelfde uitvoeringstijd.

Bij de point-and-click methode krijgen we de feedback pas bij de klik in het model, nadat de nieuwe node wordt toegevoegd.

Indien het model een zeer rigide structuur heeft, kan ook geopteerd worden om de nieuwe node automatisch op de juiste plaats te zetten. Zo kan door bijvoorbeeld een enkele klik in de palette een node op de correcte plaats worden toegevoegd. Deze variatie laten we hier echter buiten beschouwing aangezien dit meer aanleunt bij layout algoritmes.
We concentreren ons enkel op algemeen toepasbare methoden.

3.3.2 Een edge toevoegen

Deze taak kan op tal van manieren uitgevoerd worden, waarvan de meest voorkomenden hier opgesomd worden.

Een omslachtige manier is via de palette een edge op het model plaatsen, om daarna beide uiteinden handmatig te verplaatsen naar de gewenste nodes. Het voordeel in deze methode is dat het plaatsen van een edge op een gelijkaardige manier gebeurt als het plaatsen van een node. Het verplaatsen van de twee uiteinden is echter te omslachtig en neemt teveel tijd in beslag.

Een andere manier lijkt op het toevoegen van een node gebruikmakend van een point-and-click palette. Om een edge toe te voegen klikken we op het correcte icoon in de palette, waarna we van de start naar de eind node slepen, of de start en eind node aanklikken, om zo een edge tussen deze twee nodes aan te maken.
Deze manier van werken lijkt op die van tekenprogramma's waar de gewenste tool moet worden aangeduid in een palette om ze te kunnen gebruiken. De tool zou na het maken van één edge automatisch uitgeschakeld kunnen worden, of actief blijven tot de gebruiker een andere taak uitvoert, zoals het toevoegen van een nieuwe node.

Een afgeleide methode van de vorige is om de 'edge tool' niet met een knop in het menu of in de toolbox te activeren, maar door het inhouden van een zogenaamde modifier key op het toetsenbord. Houden we bijvoorbeeld de de shift-toets ingedrukt en beginnen we te slepen vanuit een node (hierna kan de modifier key meestal al losgelaten worden), wordt er een edge aangemaakt. Sleep nu verder tot de gewenste doel node, laat de muisknop los, en de edge is aangemaakt.

Het voordeel van deze laatste methode is dat hij sneller werkt, en zo gemakkelijker meer edges aangemaakt kunnen worden. Er moet immers geen tool geselecteerd worden in het menu of in de toolbox.
Een nadeel kan zijn dat de gebruiker van het bestaan van de juiste modifier key op de hoogte moet zijn. Een knop in een menu of toolbox is zichtbaar en geeft een indicatie dat met die knop een edge aangemaakt kan worden. Bij het gebruik van een modifier key moet de gebruiker kennis hiervan hebben gemaakt, eventueel via een korte opleiding, een tutorial, of tips aan het begin van het programma.

Naast vorige voor de hand liggende methoden zijn er tal van geavanceerde methoden denkbaar. Zo kan bijvoorbeeld afhankelijk van waar een nieuwe node geplaatst wordt, beslist worden om automatisch een edge tegelijk met een node aan te maken. Om dit te kunnen moet het programma op de hoogte zijn van de structuur van het model, anders kan deze geen juiste beslissing maken. In boomstructuren lijkt dit gemakkelijker toepasbaar te zijn dan in een model zonder een vaste structuur, waar dynamisch op basis van tal van parameters bepaald moet worden of er al dan niet automatisch een edge moet aangemaakt worden, en met welke andere node deze in verbinding moet staan. Hierbij moet vooral aandacht besteed worden aan het feit dat indien deze methode niet goed en transparant werkt, dit waarschijnlijk eerder als een last dan een hulp zal werken voor de gebruiker.

Een afgeleid geval van het vorige is wanneer de gebruiker een nieuwe node tussen twee bestaande nodes wil plaatsen die verbonden zijn met een edge. Dit geval wordt verder in dit hoofdstuk, in punt 3.4, uitgebreid besproken en geanalyseerd.

3.3.3 Een node verplaatsen

Bij het aanmaken en aanpassen van het model zal het natuurlijk meermaals voorkomen dat één of meerdere nodes verplaats moeten worden.

Het verplaatsen van één enkele node gebeurt meestal door deze node eenvoudigweg te slepen naar zijn nieuwe locatie. Meestal zal vermeden willen worden dat nodes elkaar kunnen overlappen. Daarom kan bij het verplaatsen van een node best gecontroleerd worden of de nieuwe locatie zou clippen met een bestaande node, en indien dit het geval is de node niet naar die locatie te bewegen.

Als we deze denkpiste verder volgen, kunnen we er ook voor kiezen in plaats van de node niet te verplaatsen wanneer deze met een andere node clipt, de node automatisch een andere locatie toe te wijzen in de buurt van de aangeduide locatie. Deze locatie wordt zodanig gekozen dat er geen conflicten ontstaan met andere nodes. Zo kan bij het clippen van de verplaatste node met een andere node er bijvoorbeeld voor gekozen worden de node automatisch naast of onder de geclipte node te plaatsen.

Wil men meerdere nodes tegelijk verplaatsen ligt dit enigszins moeilijker. Eerst en vooral moet er een mogelijkheid gegeven worden om meerdere nodes te selecteren. Een vaak gebruikte tool hiervoor is de ‘rectangle selection tool’, waarmee de gebruiker een kader kan tekenen op het model. Alle nodes die zich binnen dit kader bevinden zullen geselecteerd worden. Dit is een eenvoudige doch effectieve manier om snel een onderdeel van het model te selecteren.
Om tot het verplaatsen van de volledige sectie over te gaan, moet de gebruiker meestal één node uit die selectie verplaatsen waardoor de rest van de selectie dezelfde verplaatsing, elke node relatief ten opzichte van hun originele locatie, doormaakt.

Ook bij het verplaatsen van meerdere nodes tegelijk moet weer rekening gehouden worden met de locatie van de andere nodes die niet in de selectie zitten. In dit geval is het echter niet aangeraden op één of meerdere nodes uit de selectie automatisch een andere locatie toe te wijzen. Wanneer men een gedeelte van een model wil verplaatsen, wil de gebruiker naar alle waarschijnlijkheid geen aanpassing doen in dit gedeelte. De verplaatsing gebeurt als aanpassing in het model in zijn groter geheel.

Aan de interne structuur van het geselecteerde gedeelte van het model wordt bij voorkeur dus geen verandering aangebracht tijdens en na het verplaatsen. Indien er wel veranderingen aangebracht worden, blijven deze best zo klein en duidelijk mogelijk. Zo wordt verricht werk, namelijk het opbouwen van de structuur van het geselecteerde gedeelte, niet teniet gedaan door automatische aanpassingen, en kan de gebruiker vrij een model herschrikken zonder onverwachte resultaten te vrezen.

3.3.4 Een edge verplaatsen

Hieronder kunnen we twee zaken verstaan.

Als eerste kan een edge verplaatst moeten worden tengevolge van een verplaatsing van een node die zich aan een eindpunt van de edge in kwestie bevindt. Dit wordt meestal opgelost doordat de eindpunten van de edge als het ware verankerd liggen in de nodes aan die eindpunten. Een verplaatsing van een node zorgt dus automatisch voor de verplaatsing van alle edges waar deze node op aangesloten is.

Soms wordt ervoor gekozen de eindpunten van een edge niet te verankeren in de node aan dat aanpunt maar in de locatie waar dat eindpunt staat, of wordt de mogelijkheid gegeven om te kiezen tussen beide methoden.

Als tweede kunnen we onder het verplaatsen van een edge verstaan dat de we eindpunten ervan willen verplaatsen. Dit stelt ons in staat de edge te herverbinden met andere nodes. Deze mogelijkheid is belangrijk omdat men zo voorkomt dat een foutief geplaatste edge volledig verwijderd moet worden, om daarna een nieuwe aan te maken op de correcte plaats.
Het verplaatsen van de eindpunten verloopt meestal analoog met het verplaatsen van een node, het eindpunt verslepen naar de juiste locatie, in dit geval de gewenste node

Dit is dus een belangrijk hulpmiddel in het aanpassen van een model.

3.3.5 Een node of edge verwijderen

Het aanpassen van een model houdt vaak ook in dat er naast elementen toegevoegd of verplaatst worden, er ook elementen verwijderd moeten worden.

Het verwijderen van een node of edge verloopt meestal gelijkaardig. Door het selecteren van de node of edge, en deze door een knop in het menu of de menubalk te verwijderen. Een alternatief hiervoor is om dit via een popup-menu te doen.

Bij het verwijderen van een node kunnen we ons afvragen of de edges die op de node in kwestie aangesloten zijn al dan niet mee verwijderd moeten worden.
Als deze automatisch verwijderd worden, maar de gebruiker deze actie niet voorzien had, moet de gebruiker handmatig nieuwe edges toevoegen op de gewenste plaats.
Het tegendeel is echter even waar. Worden de edges niet automatisch mee verwijderd terwijl de gebruiker alles wat met de te verwijderen node te maken heeft, wil verwijderen, kan ook dit leiden tot extra werk. De gebruiker moet nu immers handmatig deze edges gaan verwijderen.

Het verwijderen van meerdere nodes tegelijk is vaak een mogelijkheid, en gebeurt op een gelijkaardige manier. Na het selecteren van meerdere nodes, met bijvoorbeeld een rectangle selection tool, kunnen de geselecteerde nodes in hun geheel verwijderd worden. Als een groot deel van een model verwijderd moet worden, zou het omslachtig zijn om elke node apart te verwijderen.

3.4 Het toevoegen van een node in een bestaande edge

Een vaak voorkomende aanpassing die in een model kan gebeuren, is het toevoegen van een nieuwe node tussen twee bestaande nodes die reeds verbonden zijn door een edge. Hoe kunnen we de uitvoeringstijd van deze taak inkorten?

Er bestaan verscheidene methoden om een nieuwe node toe te voegen aan een bestaande edge tussen twee nodes. Hier volgt een overzicht van mogelijke denkpistes. Elke methode wordt eerst omschreven met behulp van een figuur. Nadien wordt deze uitgebreid geanalyseerd en geëvalueerd. In de evaluatie wordt ook bekeken hoe de methode verbeterd kan worden. De erop volgende methode zal telkens een oplossing proberen te bieden aan de tekortkomingen van de vorige.

Uiteindelijk wordt de methode uit Tlibrary uitgebreid bekeken en verklaard, dit is de laatst besproken methode.

Om de analyse van de methoden en de vergelijking hiertussen te structureren, wordt gebruikt gemaakt van het GOMS Keystroke-Level Model [Kier01], met telkens hetzelfde te testen doel, maar met een verschillend scenario.
Gebruikmakend van KLM kunnen we de uitvoeringstijd van de methode voorspellen. De bekomen waarden stellen ons beter in staat de verschillen tussen de besproken methoden te zien, en te ontdekken waar de grootste tijdswinst behaald kan worden.
Verder brengt dit een vaste structuur om de methoden uit te leggen aan de hand van een sequentie van acties.

3.4.1 KLM

Waarom gebruiken we juist KLM [Card80] uit alle GOMS modellen [Davi08]? In het GOMS concept wordt een taak geanalyseerd aan de hand van Goals, Objects, Methods en Selection rules, waarvan GOMS logisch de afkorting is. KLM is de meest eenvoudige methode uit de GOMS familie van analyse technieken. De te omschrijven taken in dit onderdeel zijn weinig tot niet abstract en vereisen dus geen hiërarchische structuur in de vorm van subacties. Tevens is er geen nood aan selection rules, er is maar één manier om tot het doel, de Goal, te geraken. De omschreven taak is rechttoe rechtaan, en telkens te vervullen door een duidelijke onambigue methode. Verder wordt er geen rekening gehouden met de learning time. Voor onze doeleinden volstaat de KLM, en is een andere, complexere, GOMS methode overbodig [John96] [Kier01].

Het KLM kan niet gebruikt worden wanneer een taak niet te omschrijven is door een serie van opeenvolgende acties, of wanneer er parallelle activiteiten of onderbrekingen plaatsvinden. Dit is hier echter niet het geval, waardoor deze techniek aan onze eisen voldoet [John96].

KLM kan mogelijk een minder accurate schatting van de uitvoeringstijd geven dan andere GOMS modellen, maar we gebruiken deze analyse hoofdzakelijk om structuur te brengen tussen de verschillende omschreven methoden, en om de orde van grootte van de totale uitvoeringstijd tussen de methoden te vergelijken.
Ook is het interessant te kijken welke onderdelen van de methode juist de langste uitvoeringstijd vergen, en waarom.

Om de uitvoeringstijd van een taak te berekenen, sommen we de keystroke-level acties samen met hun operators op.
Een keystroke-level actie is een discrete actie die niet verder op te splitsen is in andere acties, bijvoorbeeld “beweeg de muis naar locatie x” of “klik op de muisknop”. Dit zijn telkens acties waarin de gebruiker één enkele simpele actie moet uitvoeren. Aan elk zulk een actie wordt een operator toegekend, deze worden zo dadelijk beschreven.
Eens de opsomming van acties compleet is, tellen we per operator hoeveel keer deze voorkomt en vermenigvuldigen we dit met de geschatte uitvoeringstijd van deze operator. Wanneer we voor elke operator zo hun totale uitvoeringstijd in de desbetreffende taak hebben berekend, rest ons alleen nog deze totalen op te tellen. Zo bekomen we een uiteindelijk een geschatte uitvoeringstijd voor de volledige taak [John96] [Kier01].

Als uitvoeringstijd voor de KLM operators gebruiken we deze uit [Kier01], zijnde:

Operator	Betekenis	Geschatte tijd
K	Keystroke, een toets op het keyboard indrukken.	0,28 sec.
T(n)	Typen van een sequentie van toetsen op het keyboard.	n maal K sec.
P	Point, het bewegen van de muis naar de gewenste locatie op het scherm.	1,1 sec.
B	Button, een muisknop indrukken of loslaten.	0,1 sec.
BB	Een muisknop klikken, dus indrukken en onmiddellijk loslaten.	0,2 sec.
H	Home, de handen bewegen naar het keyboard of de muis.	0,4 sec.
M	Mental act, een denkproces door de gebruiker.	1,2 sec.
W	Waiting, wachten tot het systeem klaar is en de gebruiker verder kan.	moet vastgesteld worden

Tabel 1 - KLM operators

Telkens is dus gekozen voor de gemiddelde waarde. De M-operator kan bijvoorbeeld een uitvoeringstijd hebben variërend van 0,6 tot 1,35 seconden, we gebruiken echter het in [Kier01] voorgestelde gemiddelde.
Belangrijk is om telkens dezelfde, in dit geval bovenstaande, waarden te gebruiken voor de verschillende operators, om zo tot een consistente vergelijking te komen en variatie tussen de geschatte tijden van de operators uit te sluiten.

3.4.2 Aannames

Om tot een geldige vergelijking te komen moeten we eerst enkele regels opstellen hiervoor. Zo moeten de gebruikte methoden om een node toe te voegen, te verwijderen etc. in elke hieronder te bespreken methode dezelfde zijn.

Een node toevoegen:
Sommige methoden zijn enkel toepasbaar bij gebruik van point-and-click of drag-and-drop, dit zal telkens vermeld worden.

Een edge toevoegen:
Om een edge te creëren gaan we ervan uit dat we enkel van de ene node naar de andere moeten slepen. Dit is natuurlijk niet volledig in overeenstemming met de realiteit. Veelal moeten we eerst in een menu of toolbox aanduiden dat we een verbinding willen tekenen. Dat we simpelweg van de ene node naar de andere moeten slepen met de muis is dan ook een meestal onrealistische aanname, en zal er in de realiteit vaak een toets op het toetsenbord moeten ingehouden worden tijdens het slepen, of moet men eerst een knop op de werkbalk aanklikken om het tekenen van edges te activeren. Maar om tot de essentie van de gebruikte methoden te komen laten we deze details in onze verdere analyse achterwege.

Een edge herverbinden:
Voor het herverbinden van een edge moet deze eerst geselecteerd worden, waarna het begin- of eindpunt gesleept kan worden naar de nieuwe node.

Een node of edge verwijderen:
Verder veronderstellen we dat er onmiddellijk een knop beschikbaar is om een edge of node te verwijderen, zonder te moeten zoeken in een menu. Een node of edge kan dus verwijderd worden door het selecteren van dat object, om nadien op de deleteknop te klikken.

Deze aannames gelden voor elke methode, dit om een zo duidelijk mogelijke vergelijking te maken. Dit zorgt ervoor dat we elke methode als het ware zouden testen in hetzelfde programma, met telkens dezelfde procedures om een node toe te voegen enz. Zonder deze consistentie zou de vergelijking tussen de methoden in 3.4.5 een incorrect beeld geven van de onderlinge verschillen door variatie van andere factoren.

3.4.3 Doel

We willen tot een efficiënte methode komen om een node toe te voegen in een bestaande edge. We streven ernaar een methode te vinden waardoor deze taak voltooid kan worden in een uitvoeringstijd die deze van het normaal toevoegen van een node in het model benadert. Idealiter zouden de uitvoeringstijden zelfs gelijk zijn.

Om dit streefdoel te verduidelijken bekijken we eerst de KLM voor het toevoegen van een node:
- Met drag-and-drop interactie:

Nr.	Omschrijving	Operator	Stap
1.	voeg node toe aan het model	M	stap 1: voeg node toe
2.	lokaliseer het node icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	druk op de muisknop en houd deze in	B	
5.	lokaliseer de gewenste locatie in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	laat de muisknop los	B	
8.	verifieer of de nieuwe node correct staat	M	

Tabel 2 - KLM, toevoegen van een node, met drag-and-drop

Totale tijd = 2P + 2B + 4M = 7,2 sec.

- Met point-and-click interactie:

Nr.	Omschrijving	Operator	Stap
1.	voeg node toe aan het model	M	stap 1: voeg node toe
2.	lokaliseer het node icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	klik met de muisknop	BB	
5.	lokaliseer de gewenste locatie in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	klik met de muisknop	BB	
8.	verifieer of de nieuwe node correct staat	M	

Tabel 3 - KLM, toevoegen van een node, met point-and-click

Totale tijd = 2P + 2BB + 4M = 7,4 sec.

We nemen 7,2 seconden als ons ultiem doel, hiernaar streven we, al zullen we er eerst vanuit gaan om zo dicht mogelijk tegen deze uitvoeringstijd te komen.

3.4.4 Werkwijze

Na een tekstuele omschrijving van de methode volgt een opsplitsing hiervan in stappen. Elke stap is een duidelijk afgebakend geheel van een aantal acties, zoals “voeg een node toe” of “verwijder de edge”. Nadien wordt een figuur getoond die telkens de status van het model laat zien tussen elke stap.

In het KLM worden de stappen verder opgesplitst in discrete acties. Deze zijn vergezeld van een operator om zo tot de totale uitvoeringstijd van die methode te komen.

In de evaluatie wordt het KLM uitvoerig besproken. Het bekijken van de uitvoeringstijden van de stappen onderling maakt vaak duidelijk welke stap de meeste tijd vergt en waarom. Dit stelt ons in staat te concluderen of de methode in kwestie al dan niet efficiënt omgaat met de uitvoeringstijd, en waar we ons best op kunnen concentreren om een tijdswinst te behalen. Hierbij hoort dus ook het zoeken naar een manier om de uitvoeringstijd eventueel nog verder terug te dringen.

Het is de bedoeling om in elke methode een stap dichter tegen ons streefdoel te komen.

3.4.5 Methoden

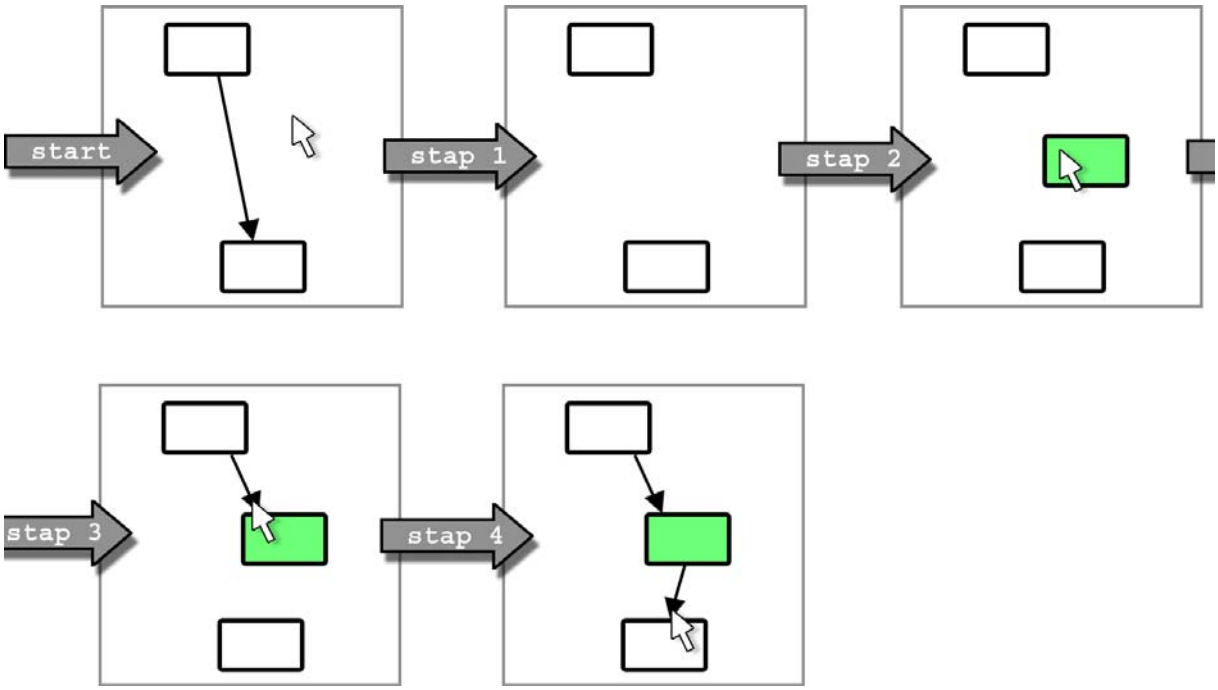
Methode 1

Omschrijving:

De meest voor de hand liggende methode is natuurlijk de bestaande edge te verwijderen, om nadien de nieuwe node toe te voegen aan het model en deze vervolgens op een correcte manier te verbinden met de twee voorheen verbonden nodes.

Om één node in een bestaande edge te plaatsen moet men dus maarliefst 4 stappen uitvoeren:

- stap 1: verwijder bestaande edge;
- stap 2: voeg node toe;
- stap 3: maak edge 1;
- stap 4: maak edge 2.



Figuur 5 - Nieuwe node (groen) toevoegen in edge, methode 1

Figuur 5 laat telkens de staat van het model zien na het uitvoeren van elke stap. Let hierbij ook op de positie van de muispointer. Na stap 1 staat deze nog op de deleteknop en is dus niet zichtbaar in het model.

De groen gekleurde node stelt telkens de node die juist toegevoegd is voor.

Deze methode is voor zowel point-and-click als drag-and-drop mogelijk. We veronderstellen het gebruik van drag-and-drop.

KLM:

Nr.	Omschrijving	Operator	Stap
1.	voeg node in bestaande edge toe	M	stap 1: verwijder bestaande edge
2.	beweeg de muis naar de edge	P	
3.	klik met de muisknop	BB	
4.	beweeg de muis naar de deleteknop	P	
5.	klik met de muisknop	BB	
6.	lokaliseer het node icoon in de palette	M	stap 2: voeg node toe
7.	beweeg de muis naar het icoon	P	
8.	druk op de muisknop en houd deze in	B	
9.	lokaliseer de gewenste locatie in het model	M	
10.	beweeg de muis naar deze locatie	P	
11.	laat de muisknop los	B	
12.	maak een verbinding aan	M	stap 3: maak edge 1
13.	beweeg de muis naar de source-node	P	
14.	druk op de muisknop en houd deze in	B	

15.	beweeg de muis naar de nieuwe node	P	
16.	laat de muisknop los	B	
17.	maak nog een verbinding aan	M	stap 4: maak edge 2
18.	druk op de muisknop (we zitten nog op de nieuwe node)	B	
19.	beweeg de muis naar de target-node	P	
20.	laat de muisknop los	B	
21.	verifieer of de nieuwe node correct staat en verbonden is	M	

Tabel 4 - KLM, toevoegen van een node in een bestaande edge, methode 1, met drag-and-drop

Totale tijd = 7P + 6B + 2BB + 6M = 15,9 sec.

Bij point-and-click geeft dit:

Nr.	Omschrijving	Operator	Stap
1.	voeg node in bestaande edge toe	M	stap 1: verwijder bestaande edge
2.	beweeg de muis naar de edge	P	
3.	klik met de muisknop	BB	
4.	beweeg de muis naar de deleteknop	P	
5.	klik met de muisknop	BB	
6.	lokaliseer het node icoon in de palette	M	stap 2: voeg node toe
7.	beweeg de muis naar het icoon	P	
8.	klik met de muisknop	BB	
9.	lokaliseer de gewenste locatie in het model	M	
10.	beweeg de muis naar deze locatie	P	
11.	klik met de muisknop	BB	
12.	maak een verbinding aan	M	stap 3: maak edge 1
13.	beweeg de muis naar de source-node	P	
14.	druk op de muisknop en houd deze in	B	
15.	beweeg de muis naar de nieuwe node	P	
16.	laat de muisknop los	B	
17.	maak nog een verbinding aan	M	stap 4: maak edge 2
18.	druk op de muisknop (we zitten nog op de nieuwe node)	B	
19.	beweeg de muis naar de target-node	P	
20.	laat de muisknop los	B	
21.	verifieer of de nieuwe node correct staat en verbonden is	M	

Tabel 5 - KLM, toevoegen van een node in een bestaande edge, methode 1, met point-and-click

Totale tijd = 7P + 4B + 4BB + 6M = 16,1 sec.
Als resultaat geeft dit een lichte stijging van 0,2 sec.

We zien weinig verschil in de KLM's tussen point-and-click en drag-and-drop, omdat er immers maar één node moet worden toegevoegd, de rest van het KLM blijft hetzelfde. De muisknop indrukken en de muisknop terug loslaten wordt bij point-and-click vervangen door 2 keer klikken.

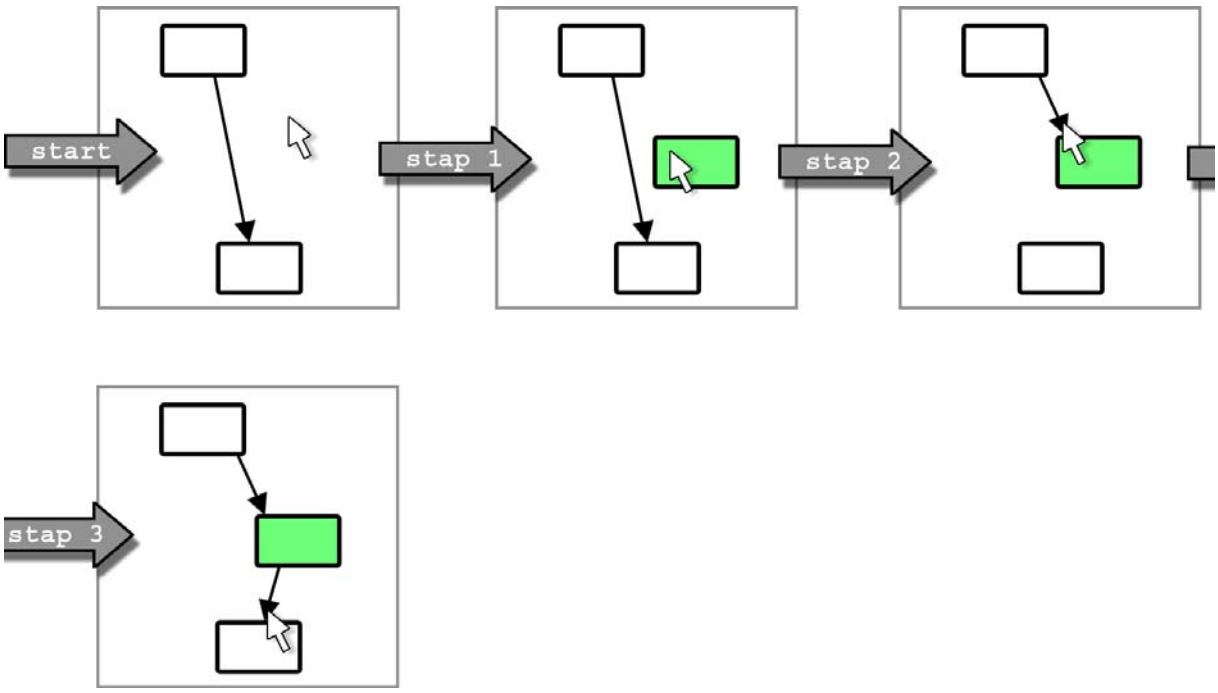
In de volgende methoden zullen we daarom enkel het resultaat geven van de alternatieve KLM in plaats van de volledige berekening.

Evaluatie:
Zoals we kunnen zien, telt het aanmaken van de 2 nieuwe edges voor bijna de helft van de acties, en natuurlijk ook voor een groot deel van de tijd. Dit terwijl er origineel al een edge stond die we dan verwijderd hadden, wat natuurlijk ook weer de nodige uitvoeringstijd vroeg.
Door het grote aantal acties en dito seconden kan deze methode als inefficiënt bestempeld worden. Het toevoegen van een nieuwe node tussen een bestaande edge is met deze methode zeer omslachtig en te tijdsintensief.

Als we de originele edge zouden kunnen gebruiken bij het tussenvoegen van een nieuwe node zouden we misschien tot een vermindering in uitvoeringstijd kunnen komen.

<u>Methode 2</u>
Omschrijving:
Indien de mogelijkheid gegeven wordt om verbindingen te herverbinden, kan methode 1 licht aangepast worden tot deze methode.
We voegen de nieuwe node toe en we verbinden de originele edge aan de nieuwe node. Nu moet nog maar één nieuwe edge toegevoegd worden. Net zoals methode 1 is deze methode voor zowel point-and-click als drag-and-drop mogelijk.
Door deze aanpassing kan men de 4 stappen van methode 1 terugbrengen tot 3 stappen:

- stap 1: voeg node toe;
- stap 2: verbind originele edge aan deze node;
- stap 3: maak één nieuwe edge.



Figuur 6 - Nieuwe node (groen) toevoegen in edge, methode 2

KLM:

Nr.	Omschrijving	Operator	Stap
1.	voeg node in bestaande edge toe	M	stap 1: voeg node toe
2.	lokaliseer het node icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	druk op de muisknop en houd deze in	B	
5.	lokaliseer de gewenste locatie in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	laat de muisknop los	B	
8.	lokaliseer de edge	M	stap 2: verbind originele edge aan deze node
9.	beweeg de muis naar de edge	P	
10.	klik met de muisknop	BB	
11.	beweeg de muis naar gewenst eindpunt	P	
12.	druk op de muisknop en houd deze in	B	
13.	beweeg de muis naar de nieuwe node	P	
14.	laat de muisknop los	B	
15.	maak nog een verbinding aan	M	stap 3: maak een nieuwe edge
16.	druk op de muisknop (we zitten nog op de nieuwe node)	B	
17.	beweeg de muis naar de target-node	P	
18.	laat de muisknop los	B	
19.	verifieer of de nieuwe node correct staat en verbonden is	M	

Tabel 6 - KLM, toevoegen van een node in een bestaande edge, methode 2

Totale tijd = 6P + 6B + 1BB + 6M = 14,6 sec.

Door de sleepactie die bij point-and-click vervangen wordt door tweemaal klikken, komt ook hier weer 0,2 seconden bij voor deze manier van nodes toevoegen, en komen we zo op 14,8 sec.

Evaluatie:
Naast methode 1 is dit ook een redelijk voor de hand liggende en vaak gebruikte methode die toch een bescheiden tijdswinst oplevert ten opzichte van de vorige.

Deze tijdswinst kan echter groter zijn dan dit cijfer doet uitschijnen. Kijkt men vooral naar stap 2 waar 3 P-acties in voorkomen. Deze zullen naar alle waarschijnlijkheid sneller kunnen dan de gemiddelde geschatte tijd van 1,1 sec., aangezien de beweging van de muis in actie 11 en het slepen van het eindpunt naar de nieuwe node in actie 13 meestal over een korte afstand zal zijn.
We merken verder op dat er geen M-actie wordt uitgevoerd voor het lokaliseren van het eindpunt van de bestaande edge en de nieuw geplaatste node. We veronderstellen hierbij dat de gebruiker, die enkele seconden daarvoor die node geplaatst heeft, deze locatie nog weet, en dat tijdens M-actie 8 de gebruiker ook ziet waar de eindpunten van die edge zich bevinden.

De tijdswinst ten opzichte van methode 1 bekomen we vooral doordat we de originele edge in het begin niet meer moeten verwijderen.
Maar deze originele edge herverbinden doet de bekomen tijdswinst voor een groot deel teniet, het herverbinden duurt hier immers langer dan een nieuwe edge maken.

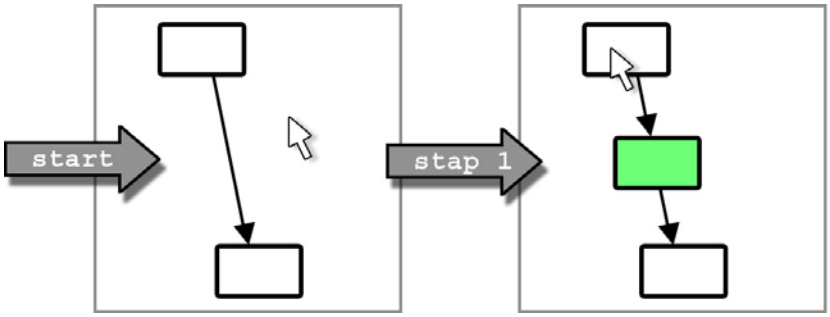
Het maken van een nieuwe edge duurt volgens deze KLM 3,6 sec., terwijl het herverbinden van een edge 4,9 sec. duurt. Zoals zojuist uitgelegd, is dit cijfer hoger geschat dan het in de realiteit zal zijn. Dit alles natuurlijk nog steeds onder de gestelde aannames, zo kan het natuurlijk evenzeer zijn dat het maken van een nieuwe edge méér tijd zal kosten dan deze inschatting veronderstelt.
We stellen dan ook dat het herverbinden en het aanmaken van een edge niet zoveel van elkaar zullen verschillen. Maar hierna moet natuurlijk nóg een edge aangemaakt worden, zodat de nieuwe node zich tussen de twee originele nodes zal bevinden.

Nu we de originele edge niet meer moeten verwijderen, is het duidelijk dat stap 2 en 3, zijnde het herverbinden en het aanmaken van een edge, een substantieel deel uitmaken van de uitvoeringstijd van deze methode. Stap 2 en 3 samen duren maarliefst 8,6 sec., oftewel meer dan de helft van de totale tijd.

Willen we de uitvoeringstijd voor het toevoegen van een node in een bestaande edge drastisch terugbrengen, dan valt bij het correct aansluiten van de nieuwe node een leuke tijdswinst te halen.

<u>Methode 3</u>
Omschrijving:
Deze methode verschilt wezenlijk van de voorgaande 2 methoden door het tussenvoegen van de nieuwe node in een bestaande edge te automatiseren. Tevens verschilt deze methode erin dat er een point-and-click manier gebruikt wordt om nieuwe nodes toe te voegen.
We klikken op de node in de palette die we in het model willen plaatsen. Normaliter zouden we nu op een plaats in het model klikken om deze node daar te plaatsen. Voor het tussenvoegen van deze node in een edge zouden we bijvoorbeeld op een bestaande node kunnen klikken waardoor de nieuwe node achter de geselecteerde node geplaatst wordt. Dit zou enkel werken in een model met gerichte edges. Anders zou men immers niets kunnen verstaan onder het ‘achter’ plaatsen van de nieuwe node.
We omschrijven dit onder één stap, immers om een gewone node met het point-and-click systeem te plaatsen zou men ergens in het model moeten klikken, wat in dit geval op een node is:

- stap 1: voeg node toe.

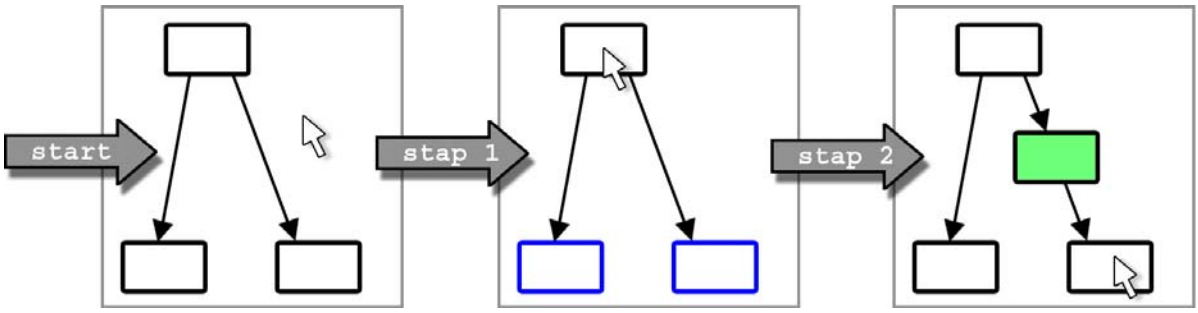


Figuur 7 - Nieuwe node (groen) toevoegen in edge, methode 3

Maar ook in een model met gerichte edges kan dit voor ambiguïteit zorgen. Wat als er vanuit de geselecteerde node meerdere edges vertrekken? Waar willen we de nieuwe node dan toevoegen? Een oplossing voor deze meerdere mogelijkheden zou kunnen zijn dat we de doel

nodes, die aan de edges hangen die uit de geselecteerde node vertrekken, laten oplichten. Dan moet de gebruiker nogmaals een gewenste node aanduiden waardoor het systeem weet tussen welke edge de nieuwe node toegevoegd moet worden. Deze werkwijze zou ook werken in een model met ongerichte edges, men specificeert de gewenste edge door het aanduiden van de 2 nodes aan het uiteinde ervan.
In dit geval onderscheiden we twee stappen:

- stap 1: voeg node toe;
- stap 2: selecteer de tweede node.



Figuur 8 - Nieuwe node (groen) toevoegen in edge, methode 3, tweede geval. (Opgelichte nodes zijn blauw gekleurd.)

Het herverbinden van de originele edge en het aanmaken van één nieuwe wordt door deze methode vervangen door het aanklikken van maximum 2 nodes. Het elimineren van deze 2 stappen, zoals beschreven in de evaluatie van methode 2, is hierbij vervuld. Laten we nu evalueren of dit wel het gewenste effect heeft in de uitvoeringstijd van deze taak.

KLM:
In het eerste geval gaan we ervan uit dat de edges gericht zijn, en dat vanuit de geselecteerde node slechts één edge vertrekt.

Nr.	Omschrijving	Operator	Stap
1.	voeg node in bestaande edge toe	M	stap 1: voeg node toe
2.	lokaliseer het node icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	klik met de muisknop	BB	
5.	lokaliseer de gewenste node in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	klik met de muisknop	BB	
8.	verifieer of de nieuwe node correct staat en verbonden is	M	

Tabel 7 - KLM, toevoegen van een node in een bestaande edge, methode 3

Totale tijd = 2P + 2BB + 4M = 7,4 sec.

In het volgende geval gaan we ervan uit dat de edges ongericht zijn, of dat de edges gericht zijn maar er uit de geselecteerde node meerdere edges vertrekken. Er moeten dus twee nodes geselecteerd worden om zo de gewenste edge te specificeren.

Nr.	Omschrijving	Operator	Stap
1.	voeg node in bestaande edge toe	M	stap 1: voeg node toe
2.	lokaliseer het node icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	klik met de muisknop	BB	
5.	lokaliseer de gewenste node in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	klik met de muisknop	BB	
8.	lokaliseer de volgende gewenste node in het model	M	stap 2: selecteer de tweede node
9.	beweeg de muis naar deze locatie	P	
10.	klik met de muisknop	BB	
11.	verifieer of de nieuwe node correct staat en verbonden is	M	

Tabel 8 - KLM, toevoegen van een node in een bestaande edge, methode 3, tweede geval

Totale tijd = 3P + 3BB + 5M = 9,9 sec.

Het selecteren van een tweede node zorgt voor een lichte stijging van 2,5 seconden.

Evaluatie:
In het eerste geval wordt de uitvoeringstijd van methode 2 nagenoeg gehalveerd. Het automatisch aansluiten van de nieuwe node aan de edge zorgt voor een drastische verlaging in de uitvoeringstijd van de taak. Dit resultaat is volgens de evaluatie van methode 2 dan ook niet geheel onverwacht, daar duurde het aansluiten van de nieuwe node immers ruim 8 seconden.

Wat opvalt, is dat de uitvoeringstijd in dit geval dezelfde is als het gewoon toevoegen van een node in het model middels een point-and-click interactie. Het gewoon toevoegen duurt ook 7,4 sec., met als enige verschil dat men klikt op een lege locatie in het model, in plaats van op een node zoals beschreven in deze methode.
Dit is het resultaat dat we voor ogen hadden en naar streefden. Het toevoegen van een node in een model duurt nu altijd even lang, eender of deze nu op een lege locatie of op een edge geplaatst moet worden. Op het eerste zicht lijkt er dan ook geen verbetering meer mogelijk in het toevoegen van een node in een bestaande edge.

Maar wanneer we naar het tweede geval van deze methode kijken, blijkt dit toch niet helemaal correct te zijn.
Vaak zullen uit een node meerdere gerichte edges vertrekken, of zijn de edges ongericht. Hierdoor moeten 2 nodes aangeduid worden om een edge te specificeren. De 9,9 sec. die hierdoor gebruikt worden zijn nog steeds een grote verbetering ten opzichte van methode 2, een winst van maarliefst 4,7 seconden, maar meer dan de uitvoeringstijd om een node gewoon toe te voegen in het model.

We zullen daarom nog een andere methode ontwikkelen die dit obstakel probeert te overkomen en volledig analoog probeert te zijn met het gewoon toevoegen van een node.

Merk op dat in het eerste geval ook een drag-and-drop implementatie denkbaar is, waar de nieuwe node gedropt wordt op de gewenste node, in plaats van die node te selecteren, en zo automatisch in de juiste verbinding wordt gezet. In het tweede geval is dit niet mogelijk, de nieuwe node kan immers niet op 2 verschillende nodes gedropt worden.

Methode 4

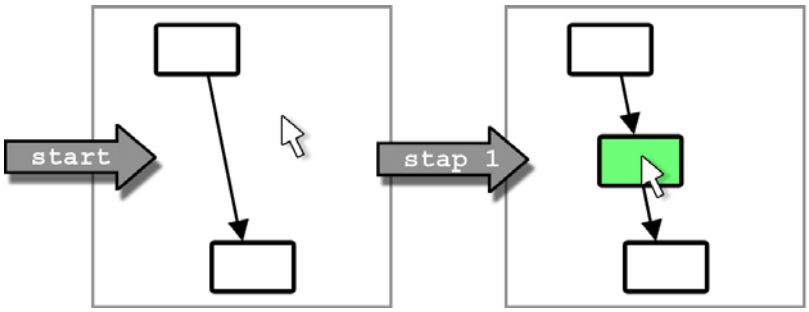
Omschrijving:

In deze laatste methode zal geen onderscheid gemaakt worden tussen gerichte en ongerichte edges. Hier gaan we nog een stap verder dan methode 3, waar het aanduiden van een tweede node om een edge te specificeren ervoor zorgde dat die methode niet altijd dezelfde uitvoeringstijd zou hebben als het gewoon toevoegen van een node. Die stap zal in deze methode dan ook vermeden worden.

Bij het toevoegen van een node gebruik makende van drag-and-drop interactie sleept men een node uit de palette naar het model, waar deze gedropt wordt. Deze methode bestaat erin dat het droppen van een nieuwe node ook mogelijk wordt gemaakt op een edge. Bij het herkennen van deze actie wordt de edge automatisch opgesplitst in twee edges en wordt de nieuwe node ertussen geplaatst.
Een point-and-click variant hiervan is ook denkbaar, waar men in de plaats van op een lege plaats in het model te klikken om de node te plaatsen, men op een edge klikt om de node daar toe te voegen. We prefereren echter de drag-and-drop versie, zoals reeds aangehaald in 3.3.1.

Deze methode omschrijven we in één enkele stap:

- stap 1: voeg node toe.



Figuur 9 - Nieuwe node (groen) toevoegen in edge, methode 4

KLM:

Eerst bekijken we deze methode gebruik makend van drag-and-drop interactie:

Nr.	Omschrijving	Operator	Stap
1.	voeg node in bestaande edge toe	M	stap 1: voeg node toe
2.	lokaliseer het node icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	druk op de muisknop en houd deze in	B	
5.	lokaliseer de gewenste edge in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	laat de muisknop los	B	
8.	verifieer of de nieuwe node correct staat en verbonden is	M	

Tabel 9 - KLM, toevoegen van een node in een bestaande edge, methode 4, met drag-and-drop

Totale tijd = 2P + 2B + 4M = 7,2 sec.

Bij point-and-click geeft dit:

Nr.	Omschrijving	Operator	Stap
1.	voeg node in bestaande edge toe	M	stap 1: voeg node toe
2.	lokaliseer het node icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	klik met de muisknop	BB	
5.	lokaliseer de gewenste edge in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	klik met de muisknop	BB	
8.	verifieer of de nieuwe node correct staat en verbonden is	M	

Tabel 10 - KLM, toevoegen van een node in een bestaande edge, methode 4, met point-and-click

Totale tijd = 2P + 2BB + 4M = 7,4 sec.

Evaluatie:
Zoals de omschrijving al deed vermoeden, leunt deze methode zeer dicht aan tegen het toevoegen van een node in het model, zowel bij drag-and-drop als bij point-and-click interactie. Men kan aannemen dat het vinden van de juiste edge in het model even lang zou duren als het vinden van de juiste locatie in het model. Hetzelfde geldt voor het droppen (of klikken) op deze edge, ook deze actie kan in dezelfde uitvoeringstijd als het droppen (of klikken) op de gekozen lege plaats in het model.

Door het rechtstreeks aanduiden van de edge waarin men de nieuwe node wil integreren, omzeilt men het aanduiden van de twee nodes om zo deze edge te specificeren. Deze methode geeft daarom hetzelfde resultaat ongeacht of de edges gericht zijn of niet, en ongeacht hoeveel edges er uit een bestaande node vertrekken.

Opmerkelijk is dat deze methode daarom ook met drag-and-drop interactie kan gebruikt worden, wat bij methode 3 niet altijd het geval was.

3.4.6 Besluit

We begonnen met methode 1 die het meest voor de hand liggend was. Snel was duidelijk dat deze inefficiënt en omslachtig omging met zowel de uitvoeringstijd als het aantal acties dat uitgevoerd moest worden.

Methode 2, waarin de originele edge herverbonden werd, was hier een verbeterde versie van, maar kwam nog niet in de buurt van ons streefdoel.

Daarom had methode 3 een aanpak die het tussenvoegen van een node als specifiek probleem probeerde op te lossen. Daar was hij grotendeels succesvol in. Toch kon de methode in enkele gevallen voor meerdere extra acties zorgen, waardoor ook de uitvoeringstijd licht opliep.

Tevens was deze methode niet zo geschikt voor drag-and-drop interactie, die onze voorkeur genoot.

Als laatste werd methode 4 besproken, die zelfs in de enkele gevallen waar methode 3 niet zoveel succes boekte altijd een consistente werkwijze hanteert en altijd dezelfde uitvoeringstijd behaalt.

De uitvoeringstijd van methode 4 is dezelfde als voor het toevoegen van een node in het model. Dit zowel met drag-and-drop als met point-and-click interactie, en ongeacht welk type verbinding er wordt gebruikt.

We kunnen besluiten dat het doel bij deze niet alleen benaderd is, maar dat ons streefdoel van 7,2 seconden ook effectief behaald is.

3.5 Het toevoegen van een item aan een node

Naast nodes die aan een model toegevoegd worden en edges die deze nodes verbinden om zo de hoofdstructuur van het model te bepalen, kunnen er ook vaak objecten toegevoegd worden aan een node, afhankelijk van het gebruikte model. Zulke objecten die in nodes geplaatst kunnen worden, zullen we ‘items’ noemen, zoals te lezen was in 3.2.

Nodes zorgen dus voor de opbouw van het model zelf, terwijl items de opbouw binnen een node verzorgen. Een duidelijk voorbeeld van deze werkwijze is het ‘scene script’ uit [Vand08]. Een ander voorbeeld zou een prototyping tool kunnen zijn, waar de dialoogvensters en de verbindingen ertussen de hoofdstructuur bepalen, en knoppen, textboxes, labels etc. de items zijn waarmee een dialoogvenster, dus de node, gevuld kan worden.

We bekijken enkele methoden om een item aan een node toe te voegen. Hiervoor gebruiken we geen specifieke aannames zoals in 3.4.2, aangezien hier geen nodes of edges toegevoegd of verwijderd moeten worden. We bespreken dus elke methode apart.

3.5.1 Doel

We gaan op zoek naar een methode om items toe te voegen aan een node, in een zo kort mogelijke uitvoeringstijd, in een zo klein mogelijk aantal acties, en bij voorkeur gebruik makend van drag-and-drop interactie. Een specifieker streefdoel stellen we hier niet.

3.5.2 Methoden

Methode 1

Omschrijving:

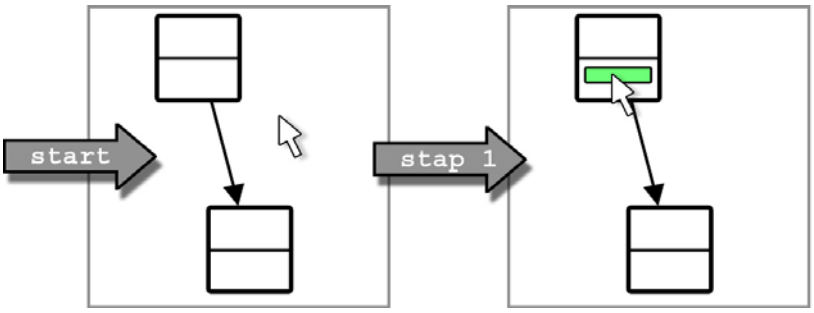
Deze methode kan gebruikt worden bij zowel een point-and-click als een drag-and-drop palette.

Gebruik makend van point-and-click interactie ziet deze methode er als volgt uit. We klikken in een palette het item aan dat aan een node toegevoegd moet worden, om daarna op de gewenste node, of het juiste onderdeel van die node, te klikken. Het item wordt nu op de geklikte plaats ingevoegd in die node.

De drag-and-drop versie is zeer gelijkend hierop, met als enige verschil dat het item naar de node gesleept wordt en daar gedropt wordt.

Er kan duidelijk maar één stap onderscheiden worden:

- stap 1: voeg item toe.



Figuur 10 - Item (groen) toevoegen, methode 1

KLM:

- met drag-and-drop:

Nr.	Omschrijving	Operator	Stap
1.	voeg item toe aan node	M	stap 1: voeg item toe
2.	lokaliseer het item icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	druk op de muisknop en houd deze in	B	
5.	lokaliseer de gewenste node in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	laat de muisknop los	B	
8.	verifieer of het nieuwe item correct staat	M	

Tabel 11 - KLM, toevoegen van een item aan een node, methode 1, met drag-and-drop

Totale tijd = 2P + 2B + 4M = 7,2 sec.

- met point-and-click:

Nr.	Omschrijving	Operator	Stap
1.	voeg item toe aan node	M	stap 1: voeg item toe
2.	lokaliseer het item icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	klik met de muisknop	BB	
5.	lokaliseer de gewenste node in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	klik met de muisknop	BB	
8.	verifieer of het nieuwe item correct staat	M	

Tabel 12 - KLM, toevoegen van een item aan een node, methode 1, met point-and-click

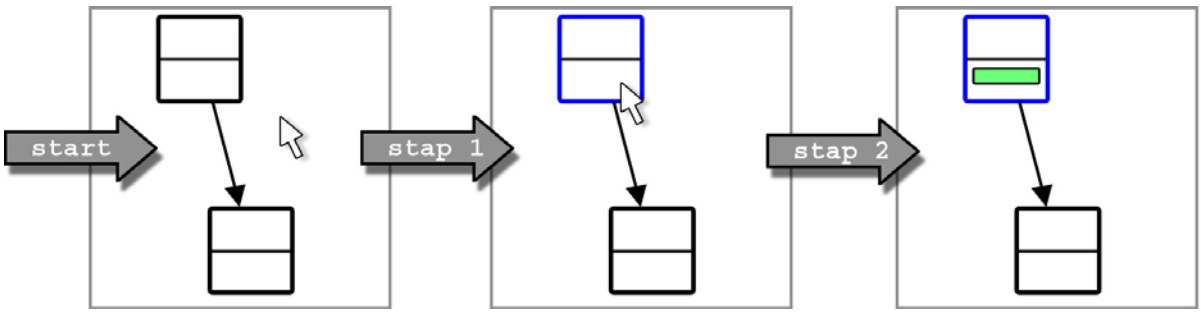
Totale tijd = 2P + 2BB + 4M = 7,4 sec.

Evaluatie:
Wat meteen opvalt, en uit de omschrijving en de figuur ook al duidelijk was, is dat de uitvoeringstijd hetzelfde is als deze voor het toevoegen van een node aan het model.
Hier is dus weinig tot niets aan uitvoeringstijd in te korten.

Toch bekijken we nog een alternatieve methode.

Methode 2
Omschrijving:
Deze methode is enkel toepasbaar in samenwerking met een point-and-click palette, en gaat omgekeerd te werk ten opzicht van methode 1.
We selecteren de node waarin we een nieuw item willen plaatsen, en voegen naderhand door een klik op het item in de palette het item hieraan toe. Dit kan enkel indien het model weet op welke plaats dit item moet terechtkomen.

Hier kunnen we 2 stappen onderscheiden:
- stap 1: selecteer node;
- stap 2: voeg item toe.



Figuur 11 - Item (groen) toevoegen, methode 2 (Opgelichte nodes zijn blauw gekleurd)

KLM:

Nr.	Omschrijving	Operator	Stap
1.	voeg item toe aan node	M	stap 1: selecteer node
2.	lokaliseer de gewenste node in het model	M	
3.	beweeg de muis naar deze locatie	P	
4.	klik met de muisknop	BB	
5.	lokaliseer het item icoon in de palette	M	stap 2: voeg item toe
6.	beweeg de muis naar het icoon	P	
7.	klik met de muisknop	BB	
8.	verifieer of het nieuwe item correct staat	M	

Tabel 13 - KLM, toevoegen van een item aan een node, methode 2

Totale tijd = 2P + 2BB + 4M = 7,4 sec.

Evaluatie:
Zoals de omschrijving weer deed vermoeden, is de uitvoeringstijd van deze methode dezelfde als methode 1 met point-and-click interactie. Deze methode is dan ook niets meer dan het omgekeerde hiervan. In plaats van eerst naar de palette te gaan en daarna op de node te klikken, klikken we nu eerst op de node en daarna in de palette. Hierdoor kunnen we de stap 'selecteer node' onderscheiden als aparte stap, maar dit heeft verder geen enkele invloed op de uitvoeringstijd.

Let hierbij op dat het gekozen item automatisch in de geselecteerde node geplaatst wordt, wat wil zeggen dat die node moet weten waar het toe te voegen item terecht kan. Als we de node in figuur 11 als voorbeeld nemen, moeten we in dit geval vaststellen dat het item toegevoegd wordt in het onderste gedeelte van die node. Stel dat zowel het bovenste als onderste gedeelte hetzelfde soort item kan bevatten, dan is deze methode niet toepasbaar, het systeem zou dan zelf een beslissing moeten maken waar het item terecht komt. Dit euvel is eventueel op te lossen door het laten oplichten van de mogelijke onderdelen van de geselecteerde node waar het gekozen item in geplaatst kan worden, waarna de gebruiker een keuze moet maken, of door bijvoorbeeld de keuze via een popup-menu in de palette aan de gebruiker te bieden.
In ieder geval zou een uitbreiding van deze methode enkel voor meer acties, en bijgevolg een langere uitvoeringstijd, zorgen.

<u>Methode 3: preview methode</u>
Omschrijving:
Tijdswinst is er ten opzichte van methode 1 niet te boeken, toch is de gebruiksvriendelijkheid nog een discussiepunt. Deze methode probeert de drag-and-drop versie van methode 1 te verbeteren.

Net zoals bij het toevoegen van een nieuwe node aan het model, moeten we ook hier ons afvragen wanneer we feedback krijgen over hoe het model, en in het bijzonder de node waar een item aan toegevoegd moet worden, eruit zal zien.

Waarom is deze feedback zo belangrijk? De beste aanpassing is een aanpassing die niet gemaakt moet worden. Als we op voorhand exact zouden weten wat het effect is van het toevoegen van een item, kan voorkomen worden dat een foutief item wordt toegevoegd, of op een verkeerde locatie, kortom, dat het item niet het gewenste effect heeft.

Bij methode 2 en de point-and-click versie van methode 1 wordt het item na de laatste muisklik aan de node toegevoegd. De gebruiker kan nu pas verifiëren of dit item wel de juiste keuze was, en of deze op de correcte plaats staat. Indien de gebruiker tevreden is met het resultaat van die voltooide taak is er geen enkel probleem.
Maar als het toevoegen van het item niet het gewenste effect heeft, zit er voor de gebruiker vaak niets anders op dan het item terug te verwijderen, of in het beste geval een undo functie te gebruiken. Beide taken zouden waarschijnlijk ongeveer dezelfde uitvoeringstijd in beslag nemen. Het verwijderen van een item wordt later uitvoeriger besproken in punt 3.6, de undo functie is buiten beschouwing van deze thesis.

Bij de drag-and-drop versie van methode 1 krijgen we de feedback pas na het droppen van het item in de node. Dus ook hier krijgen we pas feedback na het voltooien van de taak en zijn we

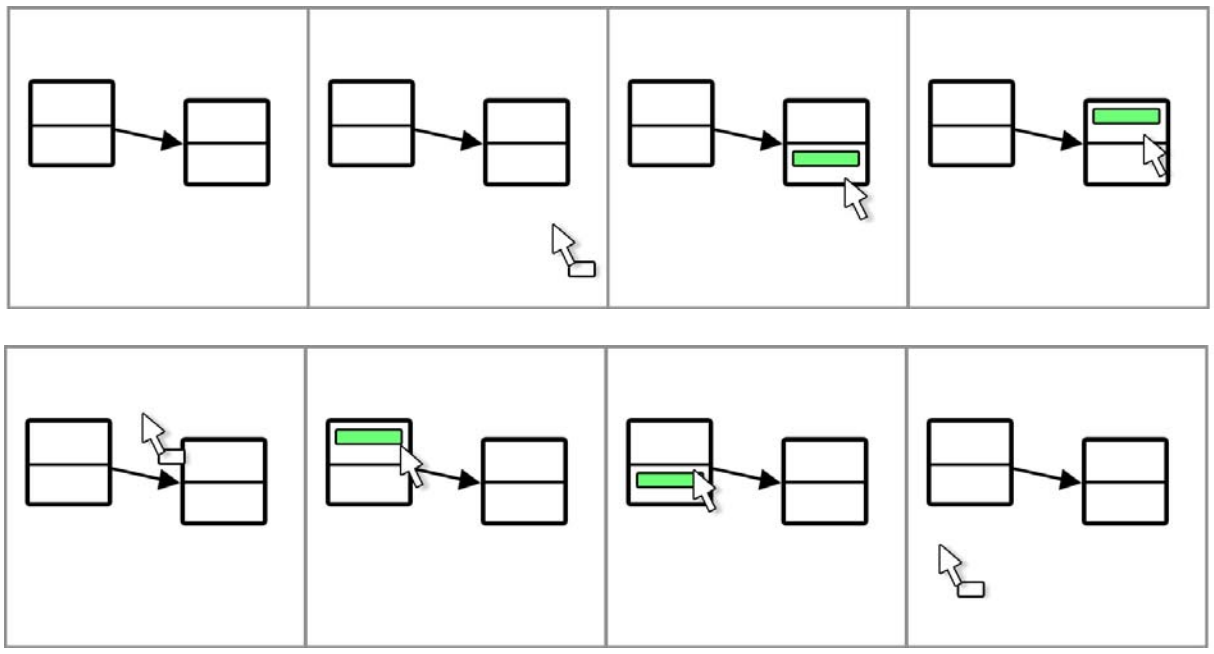
bij een ongewenst resultaat aangewezen op het terug verwijderen van het zopas toegevoegde item.

Hoe kan voorkomen worden dat de gebruiker het verkeerde item toevoegt? Hij moet feedback krijgen van het systeem nog voordat de taak voltooid is. Concreet betekent dit dat de gebruiker al het effect van zijn actie moet zien nog voor hij de muisknop heeft losgelaten bij de sleepactie. Zo kan hij beslissen om de actie door te voeren of niet.
Deze feedback nog voor een taak volledig voltooid is, wordt vanaf nu ‘preview’ genoemd, omdat er een preview versie van het model getoond wordt.

Om deze preview te bereiken moet het item dus tijdens de drag al getoond worden, niet na de drop van het item.
We slepen dus een nieuw item uit de palette, en vanaf wanneer het item op de plaats van de cursor kan ingevoegd worden, wordt dit item daar geplaatst. De plaatsing is echter niet definitief zoals bij het droppen van het item. Als de gebruiker niet tevreden is met de getoonde preview, sleept hij dit item verder tot de gewenste plaats. Hierdoor wordt het item terug verwijderd waar hij voorheen geplaatst was, en terug aan de cursor geplakt, tot de cursor weer een plaats tegenkomt waar het item terecht kan.
Het droppen van het item heeft als enig effect dat de status van de preview geaccepteerd wordt door de gebruiker, en de uitgevoerde taak definitief wordt.

Indien de gebruiker merkt dat het gesleepte item niet het item was dat hij nodig heeft, kan hij het gewoon droppen op een plaats waar het niet geaccepteerd kan worden, bijvoorbeeld in het model zelf, waardoor het item verwijderd wordt. De preview die op dat moment getoond wordt, is immers een item dat aan de muis plakt en nog aan geen enkele node toegevoegd is. Door het droppen wordt de preview versie van het model definitief, en wordt het item dus nergens toegevoegd.

In figuur 12 wordt de preview eigenschap grafisch voorgesteld. We zien twee nodes, met elk 2 onderdelen. Stel nu dat we een item dat in elk onderdeel van de nodes geplaatst kan worden uit de palette verslepen



Figuur 12 - Item (groen) toevoegen, methode 3, preview methode

Dit alles gebeurt in één enkele drag, zonder ergens een drop uit te voeren. De gebruiker kan natuurlijk op elk moment de status van het model bevestigen en een drop uitvoeren.
Eerst wordt het item over de rechtse node gesleept, de gebruiker krijgt onmiddellijk te zien dat dit item daar geplaatst kan worden en hoe dit eruit zal zien. Indien niet tevreden sleept hij het item verder naar een ander onderdeel van die node, of zelfs helemaal uit die node om verder te gaan naar een andere node.

Om het KLM te bespreken en te evalueren nemen we als voorbeeld het gewoon toevoegen van een item aan een node.
Dit kan weer in één stap, zijnde:

- stap1: voeg item toe.

De figuur hiervoor is dezelfde als methode 1, namelijk figuur 10.

KLM:

Nr.	Omschrijving	Operator	Stap
1.	voeg item toe aan node	M	stap 1: voeg item toe
2.	lokaliseer het item icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	druk op de muisknop en houd deze in	B	
5.	lokaliseer de gewenste node in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	verifieer of het nieuwe item correct staat	M	
8.	laat de muisknop los	B	

Tabel 14 - KLM, toevoegen van een item aan een node, methode 3, preview methode

Totale tijd = 2P + 2B + 4M = 7,2 sec.

Evaluatie:
Zoals eerder gezegd, is er geen tijdswinst ten opzichte van methode 1.

Wat opvalt, is dat de actie ‘verifieer of het nieuwe item correct staat’ uitgevoerd wordt terwijl de muisknop nog ingedrukt is, en niet meer als laatste actie komt. Deze verwisseling van acties ten opzichte van methode 1 is het essentiële deel van de preview methode. Mocht de gebruiker namelijk beslissen om het item toch ergens anders te zetten, kan hij deze verder slepen naar de gewenste node. Hierin zit de kracht van deze methode. Om dit te verduidelijken analyseren we zulk een scenario even.
We kunnen hier 2 stappen onderscheiden:

- stap 1: voeg item toe;
- stap 2: verplaats item naar andere node.

Het KLM ziet er dan als volgt uit:

Nr.	Omschrijving	Operator	Stap
1.	voeg item toe aan node	M	stap 1: voeg item toe
2.	lokaliseer het item icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	druk op de muisknop en houd deze in	B	
5.	lokaliseer de gewenste node in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	verifieer dat het item niet juist staat	M	
8.	lokaliseer de gewenste node in het model	M	stap 2: verplaats item naar andere node
9.	beweeg de muis naar deze locatie	P	
10.	verifieer of het nieuwe item correct staat	M	
11.	laat de muisknop los	B	

Tabel 15 - KLM, toevoegen van een item aan een node, methode 3, preview methode, tweede geval

Totale tijd = 3P + 2B + 6M = 10,7 sec.

Als we dit volgens methode 1 moeten doen, hebben we een uitvoeringstijd van 14,4 seconden voor het tweemaal toevoegen van een item, en hierbij moet nog de uitvoeringstijd van het verwijderen van het eerste item bijgeteld worden. Het verwijderen van een bestaand item uit een node wordt in het volgende onderdeel, 3.6, pas behandeld, dus gaan we hier nu nog niet verder op in.
Het is echter nu al duidelijk dat deze methode inderdaad een mooie tijdswinst oplevert, en daarbij tevens een verhoogd gebruikersgemak aanbiedt omdat er veel minder acties uitgevoerd moeten worden.

Het kan ook voorvallen dat de gebruiker merkt dat het gesleepte item niet hetgeen is dat hij verwachtte, of toch niet het gewenste effect had op de node. Hij wil het item nergens toevoegen.
Ook hier onderscheiden we weer 2 stappen:

- stap 1: voeg item toe;
- stap 2: verwijder item.

Het KLM van dit scenario geeft:

Nr.	Omschrijving	Operator	Stap
1.	voeg item toe aan node	M	stap 1: voeg item toe
2.	lokaliseer het item icoon in de palette	M	
3.	beweeg de muis naar het icoon	P	
4.	druk op de muisknop en houd deze in	B	
5.	lokaliseer de gewenste node in het model	M	
6.	beweeg de muis naar deze locatie	P	
7.	beslis om het item nergens toe te voegen	M	
8.	beweeg de muis buiten de node	P	stap 2: verwijder item
9.	laat de muisknop los	B	

Tabel 16 - KLM, toevoegen van een item aan een node, methode 3, preview methode, derde geval

Totale tijd = 3P + 2B + 4M = 8,3 sec.

Wanneer dit scenario met methode 1 behandeld moet worden, zou eerst het item gedropt moeten worden in de node om zo het effect vast te stellen, waarna het item verwijderd moet worden. Als het verwijderen ervan minder lang duurt dan 1,2 seconden zou dit sneller zijn dan de hierboven beschreven KLM. Het verwijderen van een item wordt in volgend onderdeel, namelijk 3.6, in detail besproken. Het lijkt echter wat té optimistisch om te denken dat we een item kunnen verwijderen in een uitvoeringstijd van minder dan 1,2 seconden, maar hierover meer in 3.6.

Met deze methode kan onmiddellijk gereageerd worden op de gegeven feedback, door het item naar een andere node of buiten de node te slepen. Wanneer de preview van het model bevestigd mag worden, moet de gebruiker niets meer doen dan de muisknop loslaten. Verkeerde acties worden zo in de kiem gesmoord, zonder dat er andere acties aan te pas komen.

3.5.3 Besluit

We gingen op zoek naar een methode om items toe te voegen aan een node, in een zo kort mogelijke uitvoeringstijd, met een zo klein mogelijk aantal acties, en bij voorkeur gebruik makend van drag-and-drop interactie.

Methode 1 liet ons zien dat het toevoegen van een item aan een node in een korte tijdsspanne kan gebeuren, en dat hier weinig tot zelfs geen tijdswinst te behalen valt.

Methode 2 is een variatie van methode 1, waar eerst een node geselecteerd wordt, om dan het item eraan toe te voegen. De uitvoeringstijd van deze methode bleef naar verwachting dezelfde als deze van methode 1.

In methode 3 werd er een denkwijze voorgesteld waardoor de gebruiker het effect van zijn uit te voeren taak ziet nog voordat de taak volledig ten einde is. We bekeken 2 scenario’s om dit te illustreren.

Door het gebruik van methode 3 komt de gebruiker niet voor onverwachte resultaten te staan, hij ziet hoe het model eruit zal zien, hoe het item eruit ziet en op welke plaats het zal staan. Op basis van deze feedback kan hij sneller beslissen of het toevoegen van het item al dan niet het beoogde resultaat geeft. Indien nodig kan hij op een snelle en gemakkelijke manier bijsturen door simpelweg de muis verder te bewegen naar een andere locatie.

We kunnen besluiten dat er inderdaad geen effectieve tijdswinst is geboekt in de taak van het toevoegen van een item aan een node, maar dat methode 3 het de gebruiker gemakkelijker maakt het effect van de taak in te schatten op het model, en dat deze gebruiker onmiddellijk kan bijsturen waar nodig. Dit bijsturen gebeurt in een uitvoeringstijd die korter is dan het alternatief, en tevens minder acties vereist.

3.6 Het verwijderen van een item uit een node

Desondanks het gebruik van de derde methode uit vorig onderdeel, kan het nog steeds voorvallen dat er een item uit een node verwijderd moet worden.

In dit onderdeel bekijken we enkele methoden die elk op hun manier het verwijderen van een item realiseren.

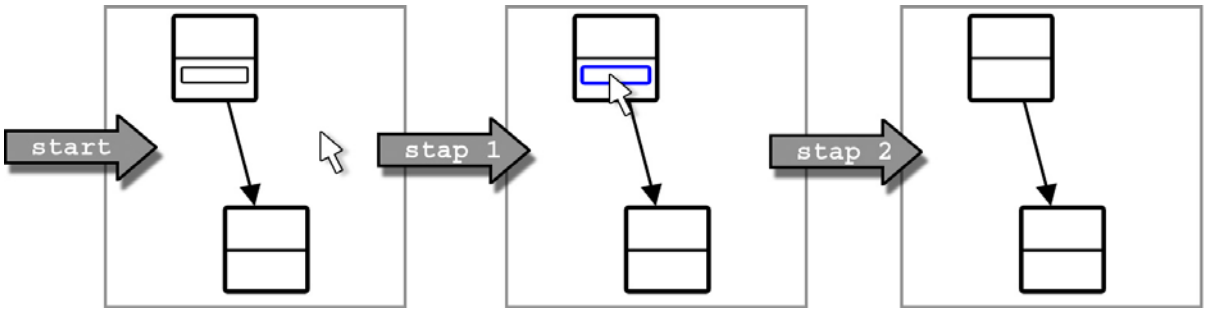
3.6.1 Doel

We gaan op zoek naar een methode om items te verwijderen uit een node, in een zo kort mogelijke uitvoeringstijd, met een zo klein mogelijk aantal acties, en bij voorkeur gebruik makend van drag-and-drop interactie.

3.6.2 Methoden

Methode 1
Omschrijving:
Analoog met het verwijderen van een node zouden we een item kunnen selecteren, en deze verwijderen met een klik op een knop in de menubalk.

Hier onderscheiden we 2 stappen:
- stap 1: selecteer item;
- stap 2: verwijder item.



Figuur 13 - Item verwijderen, methode 1 (Geselecteerde items zijns blauw weergegeven)

KLM:

Nr.	Omschrijving	Operator	Stap
1.	verwijder item	M	stap 1: selecteer item
2.	lokaliseer het item	M	
3.	beweeg de muis naar deze locatie	P	
4.	klik met de muisknop	BB	
5.	lokaliseer de deleteknop	M	stap 2: verwijder item
6.	beweeg de muis naar deze locatie	P	
7.	klik met de muisknop	BB	
8.	verifieer of het item verwijderd is	M	

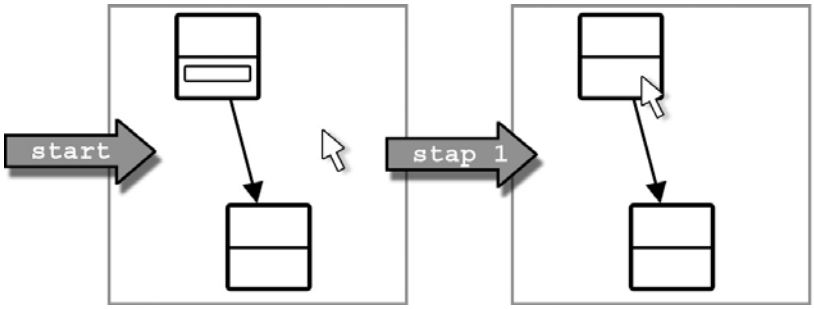
Tabel 17 - KLM, verwijderen van een item uit een node, methode 1

Totale tijd = 2P + 2BB + 4M = 7,4 sec.

Evaluatie:
Deze uitvoeringstijd is dezelfde als het toevoegen van een item volgens methode 1 met point-and-click interactie. Ook hier is er waarschijnlijk geen tijdswinst te behalen, tenzij misschien wanneer we gebruik gaan maken van popup-menu's.

Methode 2
Omschrijving:
In deze methode krijgen we de optie om via een popup-menu het item te verwijderen. Dit menu is op te roepen door een klik met de muisknop op het item.

Dit kan dus in een enkele stap:
- stap 1: verwijder item.



Figuur 14 - Item verwijderen, methode 2

KLM:

Nr.	Omschrijving	Operator	Stap
1.	verwijder item	M	stap 1: verwijder item
2.	lokaliseer het item	M	
3.	beweeg de muis naar deze locatie	P	
4.	klik met de muisknop (rechterknop)	BB	
5.	lokaliseer de delete optie	M	
6.	beweeg de muis naar deze locatie	P	
7.	klik met de muisknop	BB	
8.	verifieer of het item verwijderd is	M	

Tabel 18 - KLM, verwijderen van een item uit een node, methode 2

Totale tijd = 2P + 2BB + 4M = 7,4 sec.

Evaluatie:

Tegen de verwachting in is de uitvoeringstijd van deze methode dezelfde als die van methode 1.

De schattingen voor de operators kunnen een licht vertekend beeld geven, daarom plaatsen we hier enkele kanttekeningen bij.
Als het popup-menu slechts enkele opties bevat, zal de actie ‘lokaliseer de delete optie’ waarschijnlijk korter duren dan de ‘lokaliseer de deleteknop’ actie in methode 1, waar de knop gelokaliseerd moet worden in een menubalk.
Het compacte popup-menu in combinatie met het feit dat het menu op de locatie van de muis geopend wordt, zorgt ervoor dat actie 6 ‘beweeg de muis naar deze locatie’ korter zal zijn dan actie 6 in methode 1, waar de muis over een grotere afstand bewogen moet worden en dus langer onderweg zal zijn.

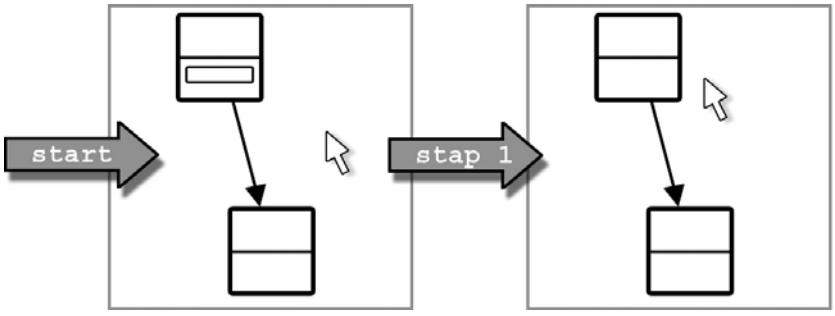
Merk verder op dat actie 4 ‘klik met de muisknop (rechterknop)’ dient om het popup-menu te tonen. Wanneer er bijvoorbeeld een fade-in effect zou gebruikt worden om dit menu tevoorschijn te halen, zou er een actie met de W-operator kunnen volgen, aangezien we dan moeten wachten tot het systeem het menu tevoorschijn heeft getoverd. Wij gaan er hier echter vanuit dat het menu onmiddellijk zichtbaar wordt.

Naar alle waarschijnlijkheid zal het gebruik van deze methode ten opzichte van methode 1 toch een kleine winst in uitvoeringstijd betekenen. Daarnaast kan deze methode eventueel ook in combinatie met methode 1 gebruikt worden, waardoor de gebruiker zelf de keuze krijgt welke optie hij zal gebruiken.

Zoals gezegd zullen we in deze taak geen tijdswinst meer kunnen behalen. Toch bekijken we verder nog een extra methode, die voor het verwijderen van items ook gebruik maakt van drag-and-drop interactie, in tegenstelling tot methode 1 en 2.

<u>Methode 3</u>
Omschrijving:
In tegenstelling tot vorige twee methoden die gebruik maken van point-and-click om een item te verwijderen, gebruikt deze methode drag-and-drop interactie. Zoals eerder gesteld in 3.3.1, prefereren gebruikers een drag-and-drop palette. Wanneer we een drag-and-drop manier zouden gebruiken om items toe te voegen, is de tegengestelde taak ook denkbaar om ze terug te verwijderen.
Door het item uit de node, of uit het onderdeel hiervan waar het item geplaatst is, te slepen en te laten vallen, wordt het item verwijderd.

Dit doen we in een enkele stap:
- verwijder item.



Figuur 15 - Item verwijderen, methode 3

KLM:

Nr.	Omschrijving	Operator	Stap
1.	verwijder item	M	
2.	lokaliseer het item	M	
3.	beweeg de muis naar deze locatie	P	stap 1: verwijder item
4.	druk op de muisknop en houd deze in	B	
5.	beweeg de muis uit de node	P	
6.	laat de muisknop los	B	
7.	verifieer of het item verwijderd is	M	

Tabel 19 - KLM, verwijderen van een item uit een node, methode 3

Totale tijd = 2P + 2B + 3M = 6 sec.

Evaluatie:
Onverwachts is de uitvoeringstijd van deze methode toch korter dan de vorige twee. Dit hadden we niet geanticipeerd, het was dan ook niet de drijfveer voor deze methode, maar het is een welgekomen verrassing.

De tijdswinst wordt behaald doordat de gebruiker niet meer moet zoeken naar de delete optie in het popup-menu zoals in methode 2, of in de menubalk zoals in methode 1. In deze methode moet die optie niet gezocht worden, de gebruiker moet enkel het item uit de node slepen om het te verwijderen. Dit verklaart het ontbreken van de mentale actie, zowel in methode 1 als methode 2 actie nr. 5, voor het lokaliseren van die optie voordat de gebruiker met de muis ernaartoe kan bewegen. Deze actie is in methode 3 overbodig.

Onze drijfveer voor deze methode was echter niet de uitvoeringstijd, maar de interactie gebruikt om het item te verwijderen. Dit kan in deze methode op een gelijkaardige manier als een item toevoegen met een drag-and-drop palette, gebruik makend van drag-and-drop interactie. Zo is er tussen de verschillende taken die een gebruiker moet uitvoeren op het model een grotere consistentie.

Bovendien is deze methode perfect combineerbaar met dezelfde preview denkwijze waarmee de derde methode voor het toevoegen van items, in 3.5.2, tot stand is gekomen. Hieruit krijgen we een afgeleide methode, methode 4.

<u>Methode 4: preview methode</u>
Omschrijving:
Deze methode, die een variatie is op methode 3, gaat er weer vanuit dat we een preview krijgen van het model, logischerwijs nog voor de taak is afgerond.

In concreto betekent dit dat we onze gemaakte acties in deze taak kunnen evalueren voordat we de muisknop loslaten om het item effectief te verwijderen.

Het aanstal stappen blijft ongewijzigd, net zoals de figuur, meer bepaald figuur 15. Deze zijn exact dezelfde als voor methode 3. Het KLM ondergaat een kleine maar belangrijke wijziging.

KLM:

Nr.	Omschrijving	Operator	Stap
1.	verwijder item	M	stap 1: verwijder item
2.	lokaliseer het item	M	
3.	beweeg de muis naar deze locatie	P	
4.	druk op de muisknop en houd deze in	B	
5.	beweeg de muis uit de node	P	
6.	verifieer of het item verwijderd is	M	
7.	laat de muisknop los	B	

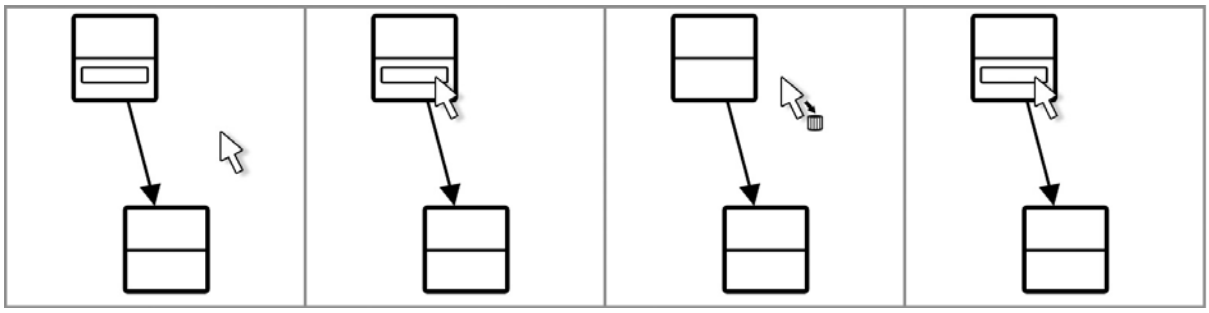
Tabel 20 - KLM, verwijderen van een item uit een node, methode 4, preview methode

Totale tijd = 2P + 2B + 3M = 6 sec.

Evaluatie:
Zoals de preview methode voor het toevoegen van een item in 3.5.2, zien we ook hier weer dat de laatste 2 acties omgewisseld zijn. Hierdoor krijgt de gebruiker feedback over zijn acties terwijl de taak van het verwijderen nog steeds bezig is, de muisknop is immers nog niet losgelaten.
Wanneer de gebruiker tevreden is met de nieuwe staat van het model, moet hij enkel nog de muisknop loslaten om de preview versie van het model te bevestigen.

Als de gebruiker niet tevreden is met het effect gecreëerd door het verwijderen van het item, dan kan hij het item terug op zijn plaats slepen, waardoor het item terug wordt toegevoegd, en de muisknop loslaten, waardoor het model terug in zijn originele staat zal zijn.

Voor de duidelijkheid bekijken we dit scenario grafisch voorgesteld in figuur 16.
Hier wordt het item uit de node gesleept. Een preview van het model, waarin het item uit de node verwijderd is, wordt als feedback getoond. De gebruiker beslist echter om het item toch te laten staan en sleept het item terug in de node, waarna hij de muisknop loslaat.



Figuur 16 - Item verwijderen, methode 4, preview methode

Dit alles gebeurt weer in één enkele drag. De gebruiker kan aan de hand van de staat van het model onmiddellijk de juiste beslissing nemen of het verwijderen van het item het gewenste effect heeft. Hij kan snel reageren op de getoonde informatie, en vermijdt hierdoor overbodige taken als bijvoorbeeld het terug toevoegen van het verwijderde item.

3.6.3 Besluit

Methode 1 beschrijft een voor de hand liggende methode waar het item eerst geselecteerd moet worden, waarna deze verwijderd kan worden door een druk op de deleteknop in de menubalk. Dit is analoog met de methode die vaak gebruikt wordt om nodes uit een model te verwijderen. Hier leek dan ook weinig tijdswinst te behalen.

Methode 2 probeerde door het gebruik van popup-menu's toch enigszins de uitvoeringstijd te beperken. Zijn KLM leek echter aan te tonen dat deze poging onsuccesvol was. Maar rekening gehouden met enkele kanttekeningen, zoals het feit dat het popup-menu op de locatie van de muis geopend wordt, is deze methode, in alle voorzichtigheid, toch een beetje sneller dan methode 1.

Verder gingen we op zoek naar een methode die een betere consistentie heeft met een drag-and-drop palette, en wouden we een methode vinden die ook gebruik maakt van drag-and-drop interactie.

De methode die deze wens kon vervullen was methode 3. Daarenboven gaf deze zelfs onverwachts een kleine winst in uitvoeringstijd. Door simpelweg het item uit de node te slepen en te laten vallen, kan het item verwijderd worden. De mentale stap om de delete optie in een popup-menu of menubalk te vinden, werd in deze methode overbodig, waardoor een hoger gebruikersgemak mogelijk gemaakt werd.

Methode 3 was bovendien perfect verenigbaar met de redenering die ons geleid had naar de preview methode voor het toevoegen van items in 3.5.2.
Hieruit ontstond methode 4, die dezelfde redenering volgde. Een item kan nog steeds verwijderd worden door deze uit de node te slepen, maar hier krijgt men tijdens het slepen onmiddellijk feedback over hoe het model eruit zal zien, de preview dus. Is de gebruiker tevreden met de staat van het model dan laat hij het item vallen, waardoor de status van de preview bevestigd en doorgevoerd wordt. De gebruiker kan, indien hij niet tevreden is met de verwijdering van het item, deze dus ook gemakkelijk terug in de node slepen. De preview wordt immers pas bevestigd bij het loslaten van de muisknop.

Doordat de gebruiker tijdens het uitvoeren van de taak kan zien welk effect het verwijderen van het item heeft, kan hij op basis hiervan tijdig en correct beslissen om het item al dan niet definitief te verwijderen.

Ook hier besluiten we dat er geen grote tijdswinst in deze taak behaald kon worden. Maar door het gebruik van methode 4 is de uitvoeringstijd toch licht ingekort, en is het aantal acties om deze taak te voltooien gereduceerd. Door de preview methode kan de gebruiker beter inschatten wat de consequenties zijn van de verwijdering van het item in kwestie, en kan hij eventueel tijdig de verwijdering ongedaan maken door simpelweg het item terug in de node te slepen.

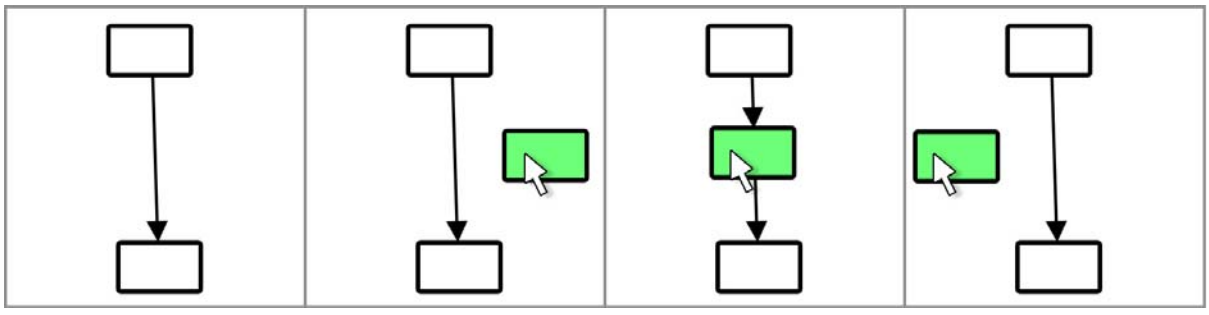
3.7 Preview

In enkele methoden hierboven is het woord ‘preview’ al gevallen. Hiermee wordt de staat van het model bedoeld nog voor de muisknop is losgelaten tijdens een drag, dus nog voor de huidige taak is afgerond. Door deze preview kan men als het ware in de toekomst zien, we zien immers het resultaat van onze taak alsof deze ook effectief doorgevoerd werd, terwijl de staat van het model toch niet definitief is. Zolang de muisknop nog ingedrukt is, is de gebruiker vrij om keuzes te maken en bij te sturen waar nodig, dit alles op basis van de informatie die hij via een preview versie van het model krijgt.

Voor het toevoegen en verwijderen van items is deze denkwijze al verwerkt in respectievelijk methode 3 in 3.5.2 en methode 4 in 3.6.2. Maar het vooruitkijken via een preview zou ook in andere gevallen handig zijn.

Zo zagen we in methode 4 voor het toevoegen van een node in een bestaande edge, in 3.4.5, dat een node tussen een edge gezet kon worden door een node uit de palette te slepen en deze op de gewenste edge te laten vallen.
Stel nu dat de node onmiddellijk zichtbaar is als we een nieuwe node uit de palette slepen en het model binnenkomen, zodat men vanaf dan als het ware de node versleept. Als we de node over een edge slepen, wordt de edge automatisch opgesplitst en wordt de gesleepte node ertussen gezet. Dit gebeurt dynamisch terwijl de muisknop nog ingedrukt is. We kunnen de nieuwe staat van het model evalueren en beslissen om de node al dan niet in zijn huidige staat te laten staan.
Indien niet tevreden kan de node dus ook uit die edge weggetrokken worden (we stellen ons hierbij de 2 nieuwe edges voor als de originele edge die gesplitst is), waardoor de originele edge zich herstelt, en de node dus niet meer verbonden is. Nu kan de node verder vrij bewogen worden, of eventueel in een andere edge gezet worden.

Dit scenario wordt in figuur 17 visueel voorgesteld. We kunnen hier telkens heel duidelijk zien welk effect het plaatsen van de node zal hebben, en zullen zo perfect kunnen inschatten, of zelfs weten, hoe het model er uiteindelijk uit zal zien bij het beëindigen van de huidige taak.



Figuur 17 - Node tussenvoegen in edge, preview methode

In figuur 17 zien we 2 nodes die verbonden zijn met een edge. Er wordt een nieuwe node uit de palette gesleept, en vervolgens van rechts naar links gesleept. Wanneer de node de edge raakt, wordt die edge in twee gesplitst met de nieuwe node ertussen. Hierna wordt de node

verder naar links bewogen en uit de edge gesleept, waardoor de edge terug in zijn originele staat verandert en de node verder verplaatst kan worden.

Zoals in alle andere preview methoden wordt de staat van het model ook hier pas definitief als de muisknop losgelaten wordt. De gebruiker komt dus nooit voor onverwachte resultaten te staan. Immers, het model zoals hij het ziet, wordt exact zo bevestigd als hij de muisknop loslaat.

Deze methode kan, naast bij het toevoegen van een nieuwe node aan het model, ook gebruikt worden tijdens het verplaatsen van een node.
Stel we hebben een node in het model geplaatst waarvan we later inzien dat deze tussen een bestaande edge thuishoort. Met deze methode kan de node gesleept worden over die edge, waardoor hij zeer eenvoudig daartussen gevoegd kan worden.

Voor het toevoegen en verwijderen van items zijn de preview methoden reeds besproken in 3.5.2 en 3.6.2. De combinatie van deze nieuwe methoden was de inspiratiebron voor de ontwikkeling van een library die de preview eigenschap implementeert en het creëren van gebruiksvriendelijke grafische model editors mogelijk maakt met een minimum aan inspanning en ontwikkelingstijd.

3.8 Besluit

Na een inleidende tekst in 3.1 en 3.2 die ons de vereiste kennis voor dit hoofdstuk verschafte, werden in 3.3 de basistaken van een grafische model editor onder de loep genomen. Deze basistaken konden op tal van manieren volbracht worden. 3.3 gaf ons een overzicht van de meest gangbare methoden hiervoor, en besprak hun voor- en nadelen.

3.4 ging aan de slag met een specifiek scenario, het toevoegen van een nieuwe node in een bestaande edge. Progressief werd tot een methode gewerkt die dezelfde uitvoeringstijd behaald als het gewoon toevoegen van een node op een lege plaats in het model, wat het vooropgesteld doel was.

Ook 3.5 besprak aan de hand van een scenario, namelijk het toevoegen van een item aan een node, enkele beschikbare methoden. Per methode werd weer progressief tot een betere methode gewerkt, om uiteindelijk tot een preview methode te komen. Deze preview methode laat de gebruiker als het ware in de toekomst kijken, door dynamisch het resultaat van zijn acties duidelijk te maken nog voor de taak is afgerond. Concreet betekent dit dat de staat van het model altijd laat zien hoe het model eruit zal zien als de gebruiker zijn acties doorvoert, in dit geval door het item te droppen. De preview moet onverwachte resultaten uitsluiten, waardoor vaak een tijdswinst te behalen valt ten opzichte van andere methoden. Onder 3.5.2 vindt men meer informatie hierover.

3.6 besprak het omgekeerde scenario van 3.5, namelijk het verwijderen van een item uit een node. Ook hier kwamen we progressief tot de beste methode, die ook een preview methode is, en dus dezelfde preview filosofie hanteert. Deze methode boekte zelfs onverwachts een kleine winst in uitvoeringstijd.

Om methode 4 uit 3.4 te verenigen met dezelfde preview werkwijze uit 3.5 en 3.6, werd hij in 3.7 uitgewerkt tot een volwaardige preview methode. Met deze methode kan men een node in

een edge slepen, en indien de preview niet het beoogde resultaat geeft, hem er ook terug uit slepen.

De combinatie van de preview methoden voorgesteld in 3.5, 3.6 en 3.7 zorgt voor een meer algemene visie op het construeren van een model. Het tijdig geven van de nodige feedback, de preview, kan in vele gevallen verkeerde acties voorkomen en bijgevolg een behoorlijke uitvoeringstijd besparen. Bovendien zijn deze methoden zeer eenvoudig bruikbaar.

4 Implementatie

4.1 Inleiding

De visie over previews tonen van een model, voorgesteld in hoofdstuk 3 en meerbepaald in 3.7, heeft geleid naar de ontwikkeling van een library waar deze voorgestelde methoden zeer eenvoudig beschikbaar worden gesteld.

Het resultaat is niet zozeer een volledige library waarin grafische editors voor modellen worden gemaakt waarin alle bekende features zoals undo/redo, het bewaren van modellen, etc. beschikbaar zijn. Maar eerder een implementatie waarin de preview methoden als een pakket worden samengesteld in een library gebouwd bovenop Visual Library 2.0.

Eerst worden de details van deze library besproken in 4.2, om daarna de implementaties van de voorgestelde methoden van hoofdstuk 3 te bekijken in 4.3. De analyse achter deze methoden vinden we terug in 3.5.2, 3.6.2 en 3.7. Tot slot van dit hoofdstuk bekijken we 2 voorbeelden van editors die op basis van deze library gerealiseerd zijn, in 4.4.

4.2 Tlibrary

De ontwikkelde library heeft de naam ‘Tlibrary’ meegekregen. De ‘T’ van Thesis uiteraard. De library steunt volledig op de voorgestelde methoden in hoofdstuk 3, om alzo deze gemakkelijk beschikbaar te stellen in meerdere grafische model editors. Wanneer we het over gebruiksvriendelijkheid hebben in Tlibrary is dit een tweevoudige stelling. Dit kan zowel voor de developer die de grafische editor maakt, als voor de gebruiker die de editor gebruikt, gelden.

De Tlibrary is gebaseerd op Visual Library 2.0, de terminologie hier steunt dan ook op die van 2.6. Per onderdeel wordt besproken hoe we de gebruiksvriendelijkheid, voor developer of gebruiker, proberen te verbeteren.

Nadien bekijken we het gebruik van de preview methoden, en hoe deze in Tlibrary centraal staan. De figuren in dit hoofdstuk zijn allen afkomstig uit een applicatie gebouwd op Tlibrary.

4.2.1 Palette

TLibrary stelt een palette beschikbaar die eenvoudig toe te voegen en te vullen is. Het toevoegen van objecten aan de palette gebeurt door de itemManager. Door de functie addPaletteChild kunnen objecten aan de palette toegevoegd worden.

Een object in de palette kan in categorieën onderverdeeld worden, en heeft telkens een icoon en een unieke naam. Op basis van deze naam gaat later in de scene en in de widgets beslist worden of het object in kwestie op die plaats toegevoegd kan worden.
De palette staat met ingebouwde drag-and-drop functionaliteit onmiddellijk in verbinding met de scene, en gebruikt de preview methode voor het toevoegen van nodes en items. Wanneer er een object uit de palette naar de scene gesleept wordt, zal elk widget op basis van de naam van het gesleepte object beslissen of het ‘acceptable’ is, en zo ja, wat er moet toegevoegd

worden. Acceptable betekent dus niets meer dan een object dat toegevoegd kan worden, oftewel accepteerbaar is.
Tevens kan per acceptable beslist worden hoeveel exemplaren van dat object toegelaten zijn.

4.2.2 TGraphScene

Zoals in 2.6.2 te lezen was, is de scene de hoofdcomponent in Visual Library 2.0. De scene in Tlibrary is een verdere ontwikkeling van de GraphScene, die gekozen is omwille zijn ondersteuning voor nodes en edges om zo verder te abstraheren van widgets.

Maar in tegenstelling tot Visual Library 2.0 moet in Tlibrary slechts één methode, addNode, geïmplementeerd worden om een bruikbare scene te maken met support voor nodes en edges. Herinneren we ons uit 2.6.6 dat de addNode en addEdge methodes instaan voor de grafische kant van de nodes en edges. Door de objectnaam die meegegeven wordt als parameter in addNode, kunnen we dus per soort node definiëren hoe deze eruit moet zien en welke acties erop worden ingesteld. Zo kan bijvoorbeeld eenvoudig een start en stop operator, en een gewone node worden gemaakt. Daarnaast kan ingesteld worden hoeveel edges er uit die nodes kunnen vertrekken en aankomen.
Zo is bijvoorbeeld een start operator eenvoudig aan te maken door maar één enkel exemplaar toe te laten op de scene, en aan te duiden dat er geen edges mogen eindigen in deze start operator.
De developer moet dus enkel op basis van de naam van de node beslissen hoe deze eruit ziet, en eventueel welke acties mogelijk zijn. Hierbij moeten alle grafische elementen in een TContainerWidget, uitgelegd in 4.2.3, steken. Zie het onderdeel over TWidgetActions, 4.2.5, welke kant en klare acties beschikbaar zijn voor de nodes.

Om te beslissen welke objecten er op de scene gezet kunnen worden, moeten deze als ‘acceptable’ aangeduid worden voor de scene. Een acceptable wil dus zeggen dat het gesleepte object uit de palette toegevoegd kan worden.

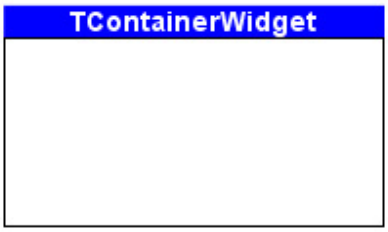
Om de gebruiker zo goed mogelijk op de hoogte te houden van zijn acties en de staat van het model, biedt de TGraphScene een infoMessage aan die automatisch geüpdate wordt met de huidige status van de TGraphScene. Hierin wordt bijvoorbeeld tijdens het maken van een edge aangegeven hoeveel edges er mogelijk zijn in een bepaalde node. De infoMessage is door de developer vrij te gebruiken, en zal meestal terechtkomen in de statusbalk.

Belangrijkste methods:

- addAcceptable: Geef de naam van de nieuwe acceptable, vanaf nu is het object met die naam acceptable in de scene.
- deleteSelectedNodes: Hierdoor worden de geselecteerde nodes of edges verwijderd.
- getInfoMessage: Geeft een string met de huidige status van het model terug.

4.2.3 TContainerWidget

Elke node in de TGraphScene is een TContainerWidget, waar naar believen TLabelWidgets of bestaande Visual Library 2.0 widgets als kind kunnen worden toegevoegd. Deze TLabelWidgets worden na dit onderdeel, in 4.2.4, besproken.



Figuur 18 - Leeg TContainerWidget

De TContainerWidget heeft enkel een titel, de inhoud is door de developer zelf te bepalen, net als de kleur van de titelbalk wanneer deze wel en niet geselecteerd is.

Belangrijkste methods:

- setTitle: Stel de titel van de TContainerWidget in.
- disableTitle: Hiermee wordt de titelbalk verborgen, bijvoorbeeld voor het maken van een start operator. Om bijvoorbeeld de start operator van figuur 19 te maken, moet enkel een imageWidget toegevoegd worden, en de titelbalk verborgen worden.

Figuur 19 - TContainerWidget met en zonder titelbalk

4.2.4 TLabelWidget

Een TContainerWidget kan dus ook met de bestaande VL 2.0 widgets gevuld worden. Wat doet een TLabelWidget dan anders?

Deze widget kan op zeer eenvoudige manier items uit de palette ontvangen. Slepen we bijvoorbeeld een item uit de palette over een correct ingestelde TLabelWidget, wordt dit item volgens de preview methode voorgesteld in 3.5.2 aan de TLabelWidget toegevoegd.

Voegen we nog meer items toe, worden deze standaard onder de bestaande items toegevoegd. Maar deze layout kan ook veranderd worden in een absolute layout zodat de items in het TLabelWidget vrij geplaatst kunnen worden, als ware het een kleine scene op zich, of in een layout waar de gebruiker zelf de volgorde van de items kan bepalen door het nieuwe item naar de correcte plaats te slepen.

Ook hier wordt via acceptables beslist welke items in het TLabelWidget gezet kunnen worden. Naast een naam, icoon en een palette categorie heeft een object in de palette ook meegekregen of het object een image of widget voorstelt. Wanneer het een image is, wordt automatisch de juiste figuur opgehaald en in het item geplaatst. Wanneer het echter een widget is, kunnen we een eigen implementatie van de TLabelwidget maken en in de acceptWidget method op basis van de naam van het palette object beslissen welk item, in dit geval welk widget, er wordt toegevoegd. We kunnen hier dus een volledig widget

samenstellen. Zo kunnen we bijvoorbeeld op het palette object “5 labels” reageren door 5 TLabelWidgets toe te voegen als item. Zelfs nog complexere composities zijn mogelijk.
Dit reageren op de naam van het gesleepte object is analoog met de addNode methode uit de TGraphScene.
Door het maken van verschillende implementaties van het TLabelWidget met telkens een andere acceptWidget method, kunnen we verschillende TLabelWidgets op verschillende manier laten reageren op hetzelfde palette object.

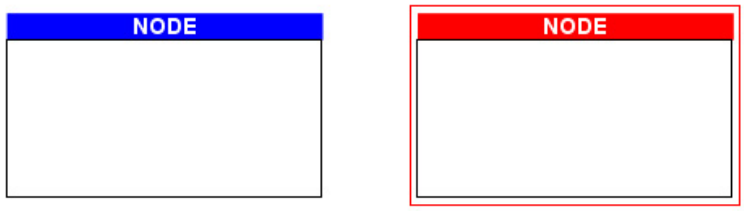
Belangrijkste methods:

- addAcceptable: geef de naam van de nieuwe acceptable aan, vanaf nu is het object in de palette met die naam acceptable in deze TLabelWidget.

4.2.5 TWidgetActions

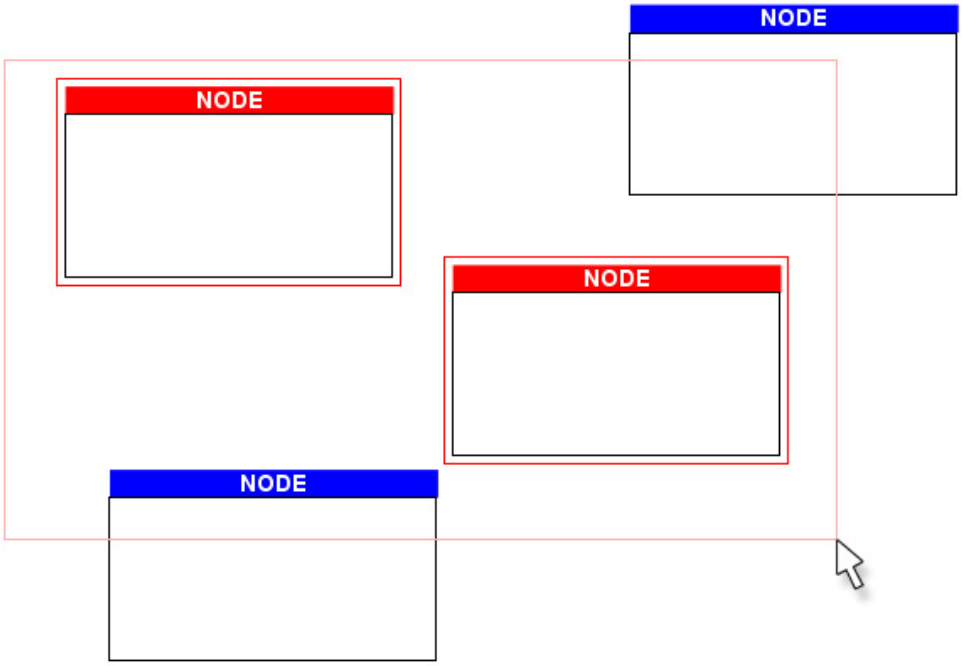
De Tlibrary biedt enkele kant en klare acties aan voor de nodes in de TGraphScene. Deze zijn zeer eenvoudig toe te voegen aan de nodes, door één regel code, en zijn onmiddellijk bruikbaar zonder extra programmeerwerk.
De belangrijkste worden in dit onderdeel besproken.

TNodeSelectAction:
De node kan geselecteerd worden. De ingestelde properties van de node worden in het properties panel getoond, en de ingestelde kleuren voor de border en de titelbalk worden actief. Indien er geen eigen kleurwaarden werden opgegeven, worden de standaard waardes gebruikt. Figuur 20 toont een voorbeeld hiervan.



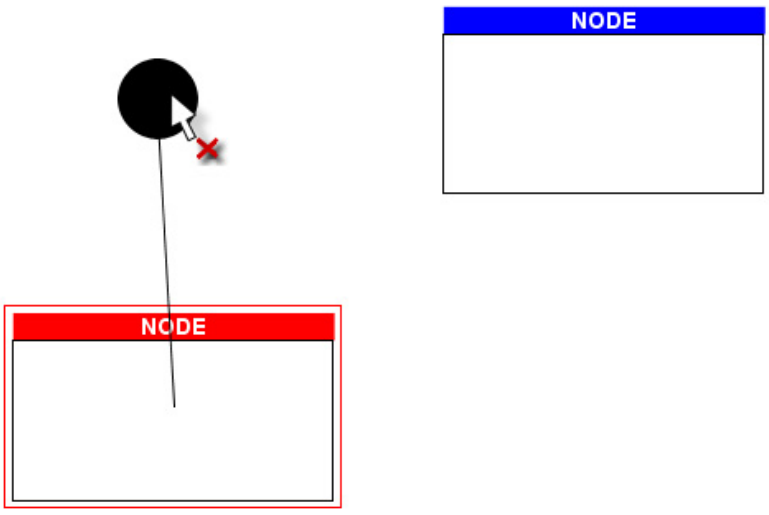
Figuur 20 - TContainerWidget niet en wel geselecteerd

Daarnaast is het ook mogelijk om meerdere nodes tegelijk te selecteren. Hiervoor is geen activatie van andere tool zoals een selection tool nodig, het volstaat op een lege plaats in de scene te beginnen slepen. Automatisch wordt de rectangle selection tool op het scherm getekend.
Nog tijdens het slepen van de selectierechthoek wordt feedback gegeven over welke nodes er in de selectie zitten, zoals we in figuur 21 kunnen zien. De selectie wordt pas bevestigd bij het loslaten van de muisknop.



Figuur 21 - TNodeSelectAction, multi select

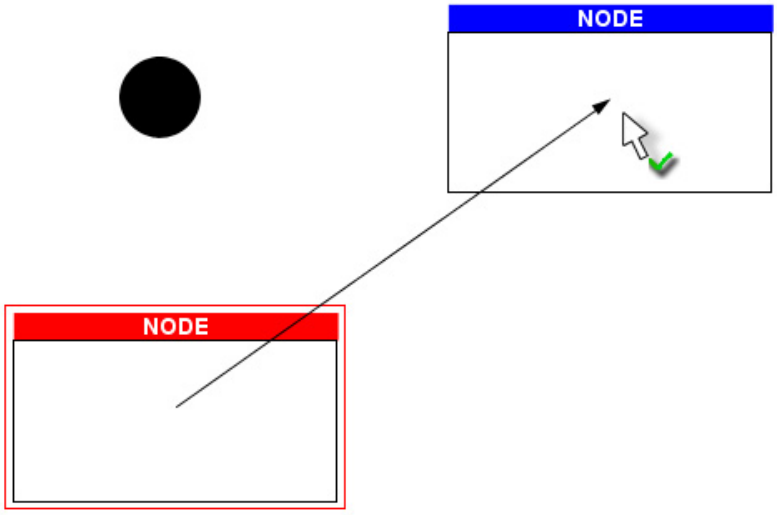
TNodeConnectAction:
Deze actie stelt de node in staat verbindingen te maken met andere nodes in het model. Hierbij wordt telkens rekening gehouden met het aantal edges dat uit die node kan vertrekken en hoeveel er hiervan nog beschikbaar zijn. Hetzelfde geldt voor de bestemming van de edge in aanmaak, hier wordt gecontroleerd of er al dan niet nog een edge op aangesloten kan worden.



Figuur 22 - TNodeConnectAction, feedback tijdens connecteren

Figuur 23 - Bijbehorende status bij figuur 22

Tijdens het maken van een edge wordt dynamisch aangegeven of die edge al dan niet gemaakt kan worden, zoals zichtbaar in figuur 22 en 24. In de infoMessage van de TGraphScene is ook te zien, afhankelijk van de huidige actie, hoeveel bron of doel connecties er mogelijk zijn in de node in kwestie. Een voorbeeld hiervan zien we in figuur 23 en 25. Deze boodschap wordt automatisch gegenereerd op basis van de beschikbare gegevens.



Figuur 24 - TNodeConnectAction, feedback tijdens connecteren

Figuur 25 - Bijbehorende status bij figuur 24

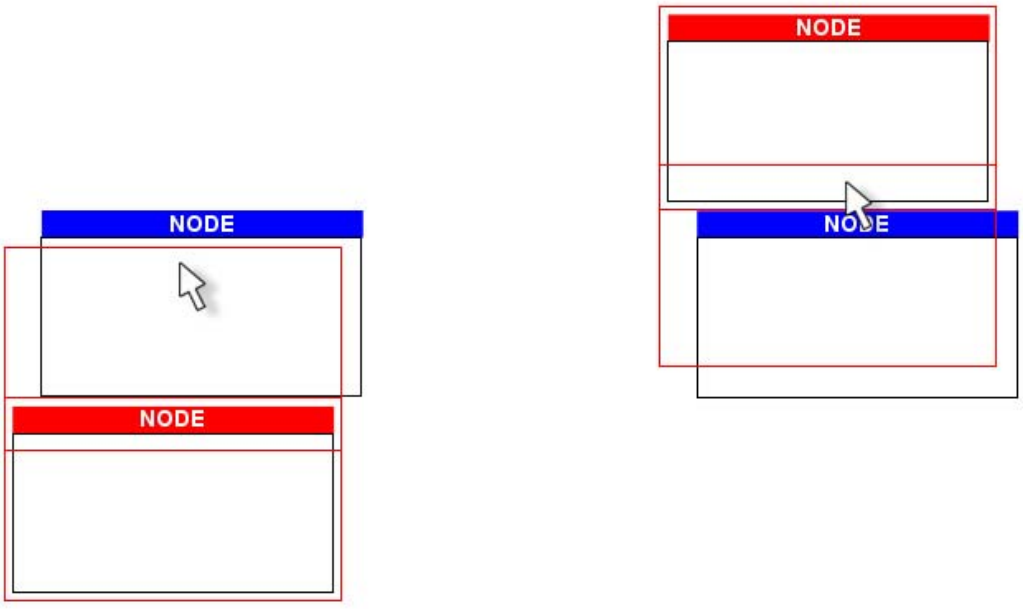
Ook bij het aanmaken van edges geldt weer het preview principe, de edge wordt immers niet aangemaakt totdat de muisknop losgelaten wordt. Zolang de muisknop ingedrukt is, kan de gebruiker over verschillende nodes bewegen om zo te zien of een edge naar die node mogelijk zou zijn.

TNodeMoveAction:
Wanneer deze actie aan een node wordt toegevoegd, is de gebruiker van de grafische editor in staat de node te verplaatsen in de scene. De nodes, die telkens TContainerWidgets zijn zoals 4.2.3 stelt, zijn enkel te bewegen door de node met de titelbalk te verslepen. Wanneer de titelbalk verborgen is, is de TContainerWidget met de volledige oppervlakte van het widget te slepen.

Zoals in 2.6.3 aangegeven, biedt Visual Library 2.0 standaard een kant en klare WidgetAction aan voor het bewegen van nodes. Toch volstond deze niet in Tlibrary, en dit om hoofdzakelijk twee redenen: het ontbreken van clipping detectie met andere nodes, en het ontbreken van een mogelijkheid om meerdere nodes tegelijk te verplaatsen. Tlibrary probeert deze gebreken in te vullen:

- Clipping detectie met andere nodes:
Bij het samenstellen van een model met nodes willen we niet dat bij het verplaatsen van nodes ze elkaar kunnen overlappen. Daarom moet tijdens het verplaatsen gecontroleerd worden of de nieuwe locatie wel beschikbaar is, dus of de gesleepte node geen volledige andere node, of een deel ervan, zou bedekken.

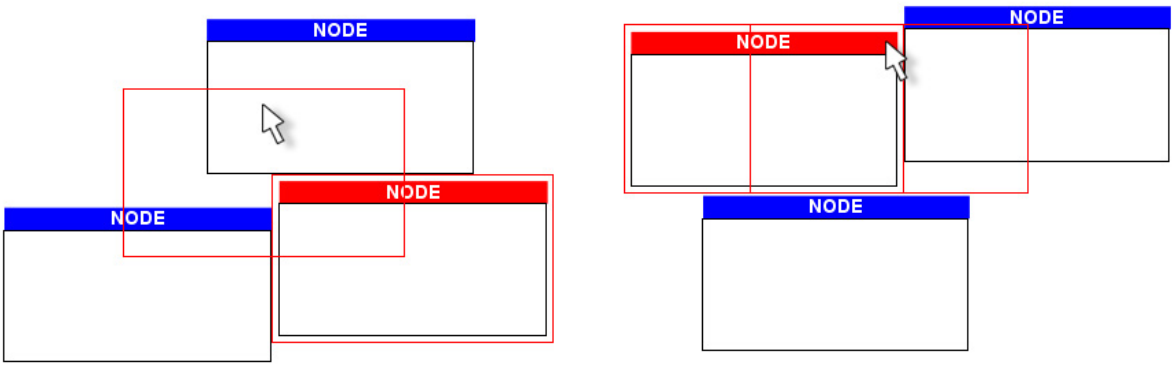
Wanneer bij het verplaatsen van een node deze zou clippen met een andere node, wordt de gesleepte node op basis van waar de gebruiker deze origineel wil zetten op een locatie gezet naast de node waarmee hij clipt. De locatie van de gesleepte node wordt dus automatisch aangepast wanneer hij zou clippen met een andere node. Die nieuwe locatie wordt dynamische bepaald afhankelijk van de plaats naar waar de gebruiker de node eigenlijk wou slepen. Die plaats is herkenbaar door een rechthoek die dezelfde border heeft als de gesleepte node. Deze rechthoek is als het ware een spookversie van de gesleepte node. Wanneer verder gesleept wordt en er geen sprake meer is van clipping, verdwijnt deze spookversie automatisch.
Figuur 26 laat ons bijvoorbeeld zien wat er gebeurt als we een node slepen van beneden naar boven, over een andere node. Ook de spookversie van de node is hier duidelijk geïllustreerd.



Figuur 26 - TNodeMoveAction, clipping detectie,automatische aanpassing van locatie

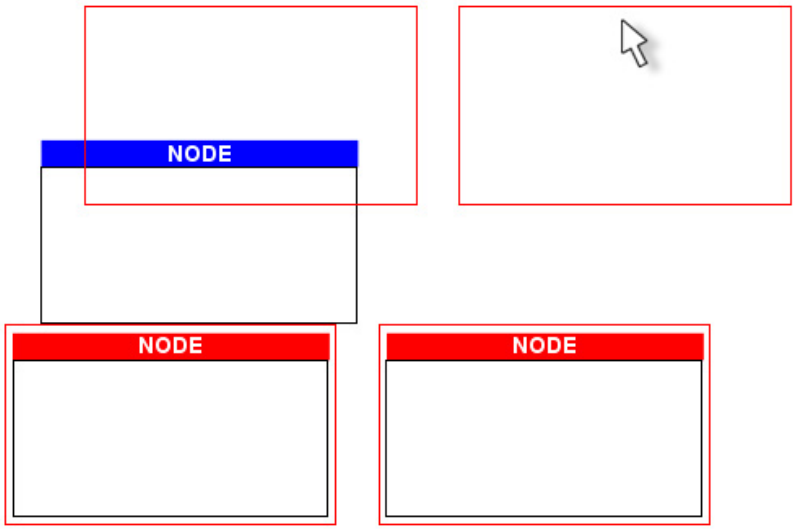
Bij het aanpassen van de locatie moet weeral gecontroleerd worden of de node op die nieuwe locatie niet zou clippen met nog een andere node. Wanneer dit gebeurt, zoals in figuur 27, wordt de locatie pas aangepast als de voorgestelde locatie niet meer in contact komt met een

andere node. In figuur 27 willen we de geselecteerde node van rechtsonder naar linksboven verplaatsen.



Figuur 27 - TNodeMoveAction, clipping detectie, controle bij aanpassing van locatie

- Meerdere nodes tegelijk verplaatsen:
Bij de TNodeSelectAction hebben we reeds gezien dat eenvoudig meerdere nodes geselecteerd kunnen worden. We willen echter ook de mogelijkheid hebben om de volledige selectie tegelijk te kunnen verplaatsen. Dit kan eenvoudig door één node uit de gemaakte selectie te slepen. Alle nodes die in de selectie zitten zullen dan, relatief ten opzichte van hun eigen locatie, dezelfde beweging maken.



Figuur 28 - TNodeMoveAction, meerdere nodes tegelijk te verplaatsen

Ook hier wordt bij het detecteren van clipping met andere nodes, die niet in de selectie zitten, de spookversies van de gesleepte nodes getoond. Dit effect kunnen we bekijken in figuur 28.
Er gebeurt echter geen automatische aanpassing van de locatie van de gesleepte nodes. Wanneer een node uit de selectie een andere locatie zou krijgen, zou de interne structuur van

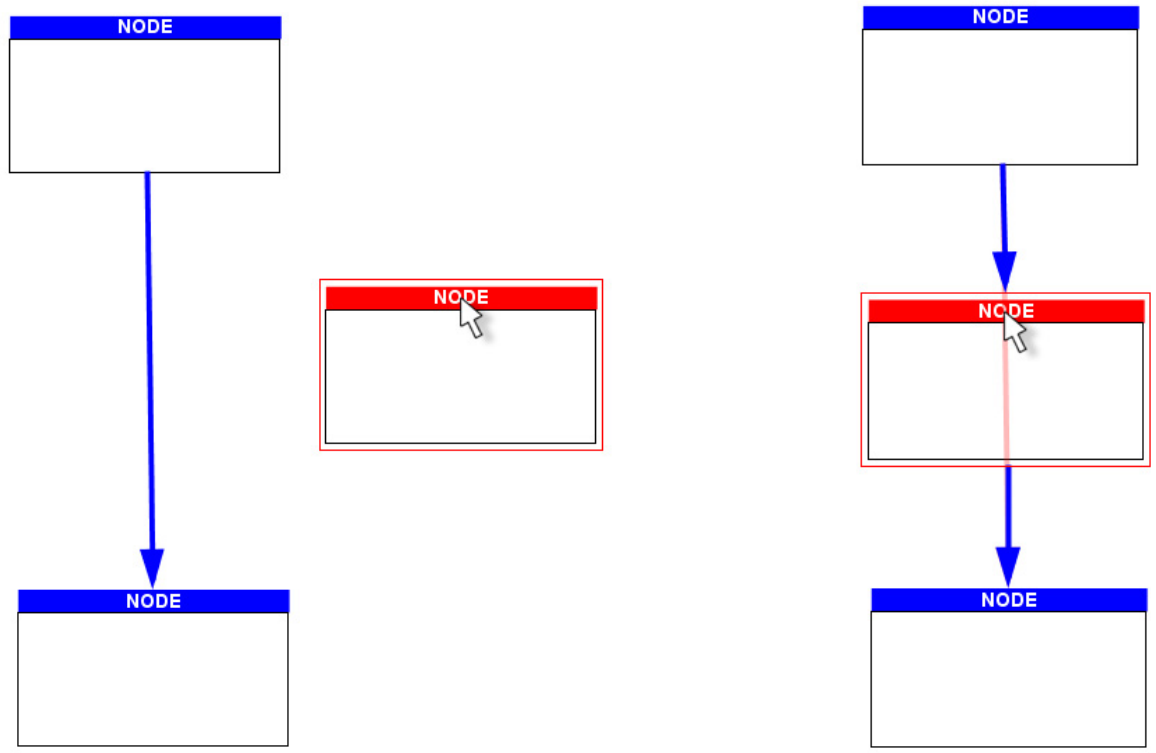
de nodes in de selectie verloren gaan. Dit terwijl het verplaatsen van meerdere nodes tegelijk meestal zal gebeuren met een gedeelte van het model waarvan we de interne structuur juist willen behouden, zoals reeds aangehaald in 3.3.3.
Merk op dat de spookversies van die nodes ook altijd in diezelfde interne structuur staan.

4.3 Preview methoden: implementaties

De preview methoden en hun voordelen zijn in hoofdstuk 3 reeds uitvoerig besproken en geanalyseerd. In dit onderdeel wordt de eigenlijke implementatie van die methoden besproken. De preview methoden hebben, zoals eerder gezegd, immers geleid tot het ontwikkelen van deze library bovenop Visual Library 2.0.

4.3.1 Het toevoegen van een node in een bestaande edge

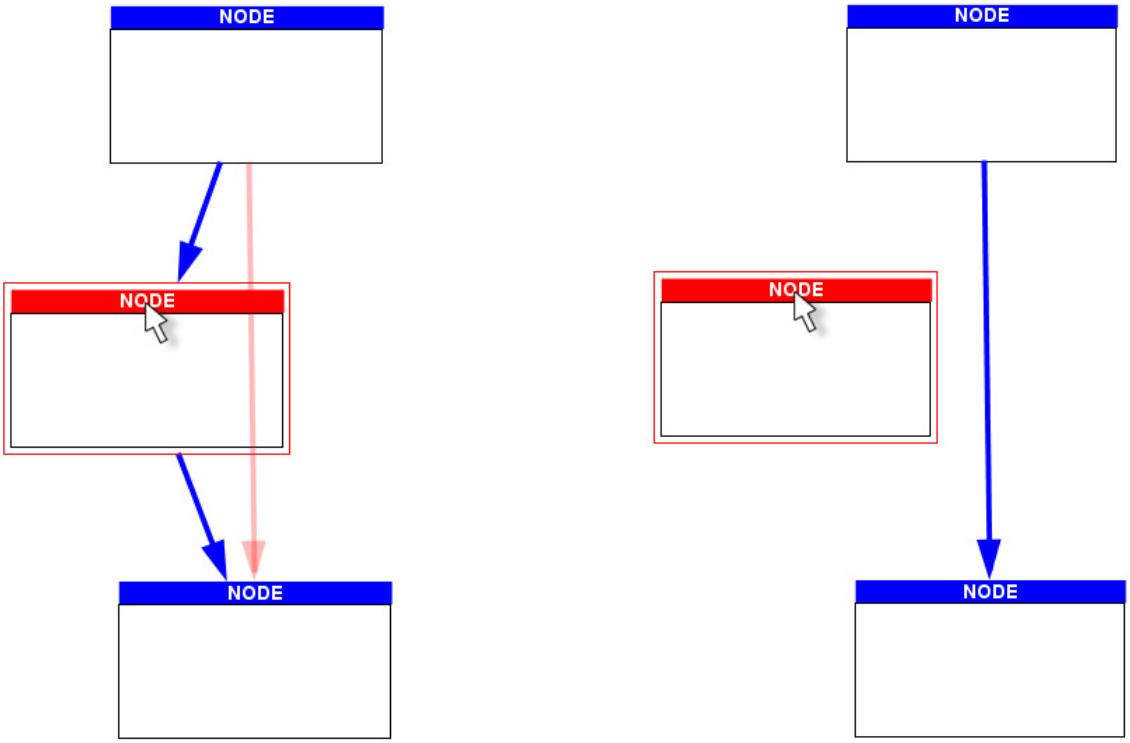
In 3.4 van deze thesis werd een methode voorgesteld om een node rechtstreeks aan een bestaande edge te kunnen toevoegen. Deze methode werd in 3.7 verder verfijnt tot een preview methode waar we een node in een bestaande edge kunnen toevoegen door deze simpelweg er overheen te slepen, en er terug uit kunnen halen door de node verder te slepen. Alles in figuur 29, en het vervolg in figuur 30, gebeurt dus in één enkele drag.
In figuur 29 starten we met een node die we overheen de edge slepen, waardoor hij daartussen gevoegd wordt. In figuur 30 slepen we deze node nog verder naar links.



Figuur 29 - Toevoegen van een node in een bestaande edge

Zoals zichtbaar in figuur 30 wordt de node niet onmiddellijk uit de edge verwijderd bij het verder slepen. Het lijkt alsof hij er nog even aan plakt. Wanneer de node effectief in deze edge

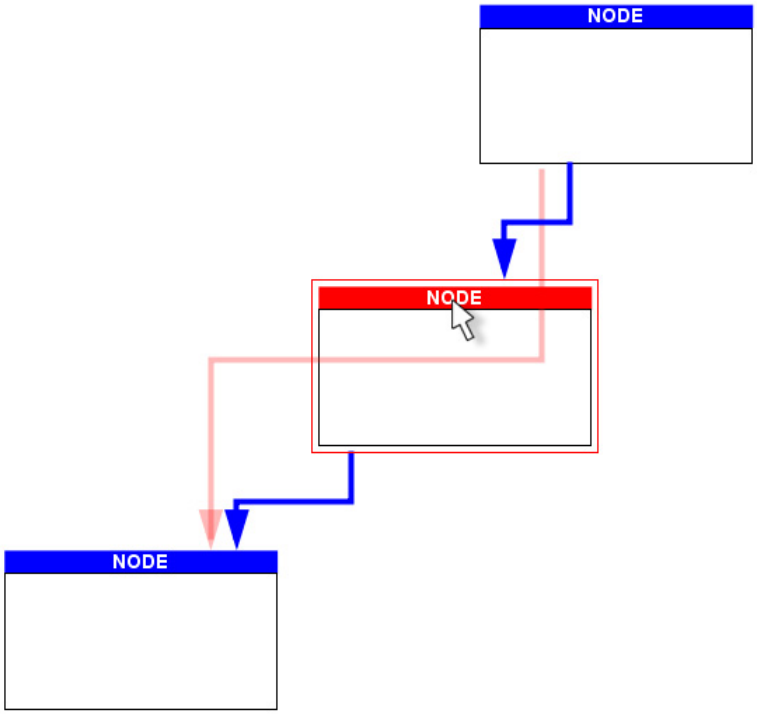
moet komen te staan, kan de gebruiker zo al kleine aanpassingen maken aan de locatie van die node. Wanneer we de node eruit willen halen, moeten we enkel de node wat verder slepen, waardoor de edge, als ware hij van rubber, terugspringt naar zijn initiële staat.



Figuur 30 - Toevoegen van een node in een bestaande edge, de node verder uit de edge slepen

Merk op dat hier de originele edge, in het doorschijnende rood, nog steeds getoond wordt zolang de nieuwe node in preview versie aangesloten is op die edge. Beweegt men verder, dan herstelt de originele versie van die edge zich volledig, en kan de node vrij verder bewogen worden. Laat men de muisknop los terwijl de node aangesloten is op de edge, wordt die status natuurlijk bevestigd, wordt de doorzichtige rode edge verwijderd, en worden de 2 nieuwe edges de definitieve versie.

Bovendien kan deze methode niet enkel toegepast worden bij het toevoegen van een nieuwe node uit de palette, maar ook tijdens het verplaatsen van een node die reeds in het model staat. Deze methode is te gebruiken met de DirectRouter en de OrthogonalSearchRouter voor edges, uitgelegd in 2.6.4. In voorgaande figuren wordt de DirectRouter gebruikt, maar het principe is met de OrthogonalSearchRouter, zoals gebruikt in figuur 31, exact hetzelfde.

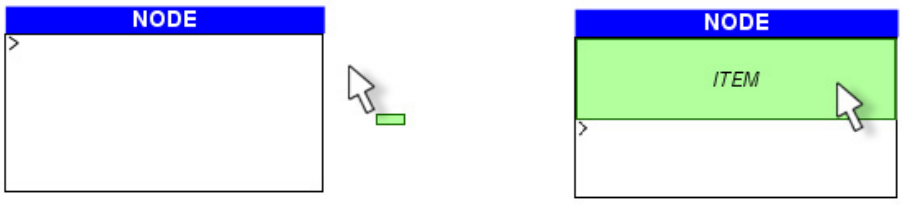


Figuur 31 - Toevoegen van een node in een bestaande edge, gebruik makend van de OrthogonalSearchRouter

Telkens wordt rekening gehouden met het aantal edges dat uit de gesleepte node kan vertrekken en er in kan aankomen. Zo zal men bijvoorbeeld een start operator nooit in een bestaande edge kunnen toevoegen, die node kan immers geen doel zijn van een edge.

4.3.2 Het toevoegen van een item aan een node

De laatste methode in 3.5.2 omschrijft de preview methode om items toe te voegen aan een node. Het principe hier was om het item al toe te voegen wanneer de gebruiker het item binnen de node sleept, en niet pas wanneer het daar gedropt wordt.



Figuur 32 - Toevoegen van een item aan een node

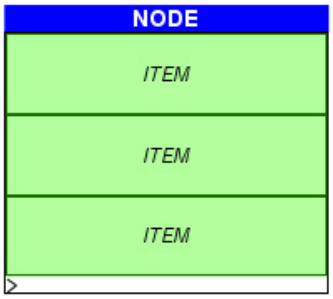
Door deze feedback kan de gebruiker sneller beslissen of dit de correcte actie was, en zo niet, de muis verder bewegen om zo het item uit de node te slepen. Dit alles kan dus gebeuren zonder de muisknop los te laten. Zo beginnen we in figuur 32 een item uit de palette te slepen, en bewegen we de muis over de node. We zien dat het item daar geplaatst kan worden doordat het daar automatisch toegevoegd wordt zonder de muisknop los te laten. Toch beslissen we in

figuur 33 het item verder naar links en uit de node te slepen. Let op, dit gebeurt nog steeds terwijl de muisknop ingedrukt is. Eens uit de node, wordt het item daar verwijderd, en terug aan de muis geplakt.

Figuur 33 - Toevoegen van een item aan een node, het item verder tot uit de node slepen

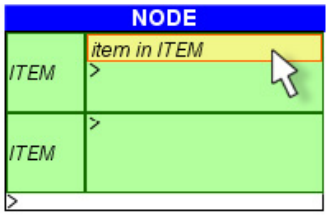
De gebruiker kan, zoals bij alle preview methoden, de muis op elk moment loslaten om de status van het model op dat moment te bevestigen. Laten we in de bovenstaande figuren de muisknop los terwijl het item toegevoegd is aan de node, zal dat item definitief toegevoegd worden. Laten we de muisknop los terwijl het item niet toegevoegd is, en een icoontje aan de muis plakt om het item voor te stellen, zal het item nergens toegevoegd worden.

Merk ook het >-teken op in de node. Dit teken geeft aan dat er items in die node kunnen geplaatst worden, dus dat deze TLabelWidget acceptables heeft, zoals uitgelegd in 4.2.4. We noemen dit de spacer. Het waarom wordt duidelijk in figuur 34, waar er zonder spacer geen items meer toegevoegd konden worden. De spacer zorgt er dus voor dat er altijd een plaats is waar we nieuwe items op kunnen slepen om ze zo toe te voegen. Een TLabelWidget heeft ook de optie om tussen elk toegevoegd item een spacer te plaatsen, op deze manier kan een nieuw item tussen bestaande items gevoegd worden, of helemaal bovenaan. Standaard worden nieuwe items telkens onderaan bijgeplaatst.



Figuur 34 - Toevoegen van een item aan een node, nut van spacer

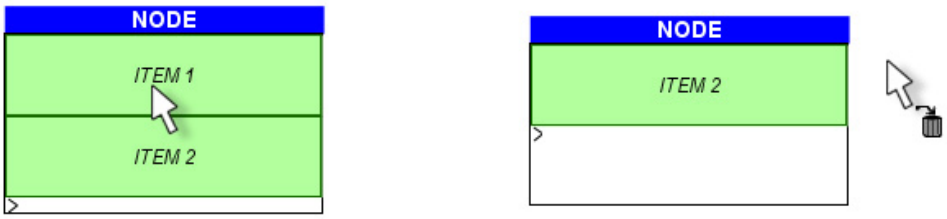
Als we nu bijvoorbeeld een item opstellen dat ook acceptables heeft, kunnen we zo relatief eenvoudig complexere nodes opstellen. In figuur 35 is er een ander item in een item gesleept. Dit gebeurt volgens dezelfde methoden als voor het toevoegen van items aan nodes.



Figuur 35 - Toevoegen van een item aan een item

4.3.3 Het verwijderen van een item uit een node

In 3.6.2 werd als laatste methode voorgesteld om items te verwijderen door ze simpelweg uit de node, of uit het onderdeel van de node waar ze toegevoegd zijn, te slepen. Deze methode wordt in Tlibrary gebruikt.



Figuur 36 - Verwijderen van een item uit een node

Het verwijderen verloopt natuurlijk ook als preview methode. Wanneer men het item uit de node sleept, wordt hier onmiddellijk de juiste feedback over gegeven door het item in kwestie te verwijderen. Beslist men toch dat deze actie niet het gewenste effect heeft, sleept men het item terug in de node. Dit alles zonder de muisknop los te laten. Om het item te verwijderen moet simpelweg de muisknop losgelaten worden wanneer het item uit de node gesleept is, om zo de verwijdering te bevestigen.
Het verwijderen van een item uit een ander item gebeurt op exact dezelfde wijze.

4.4 Voorbeelden

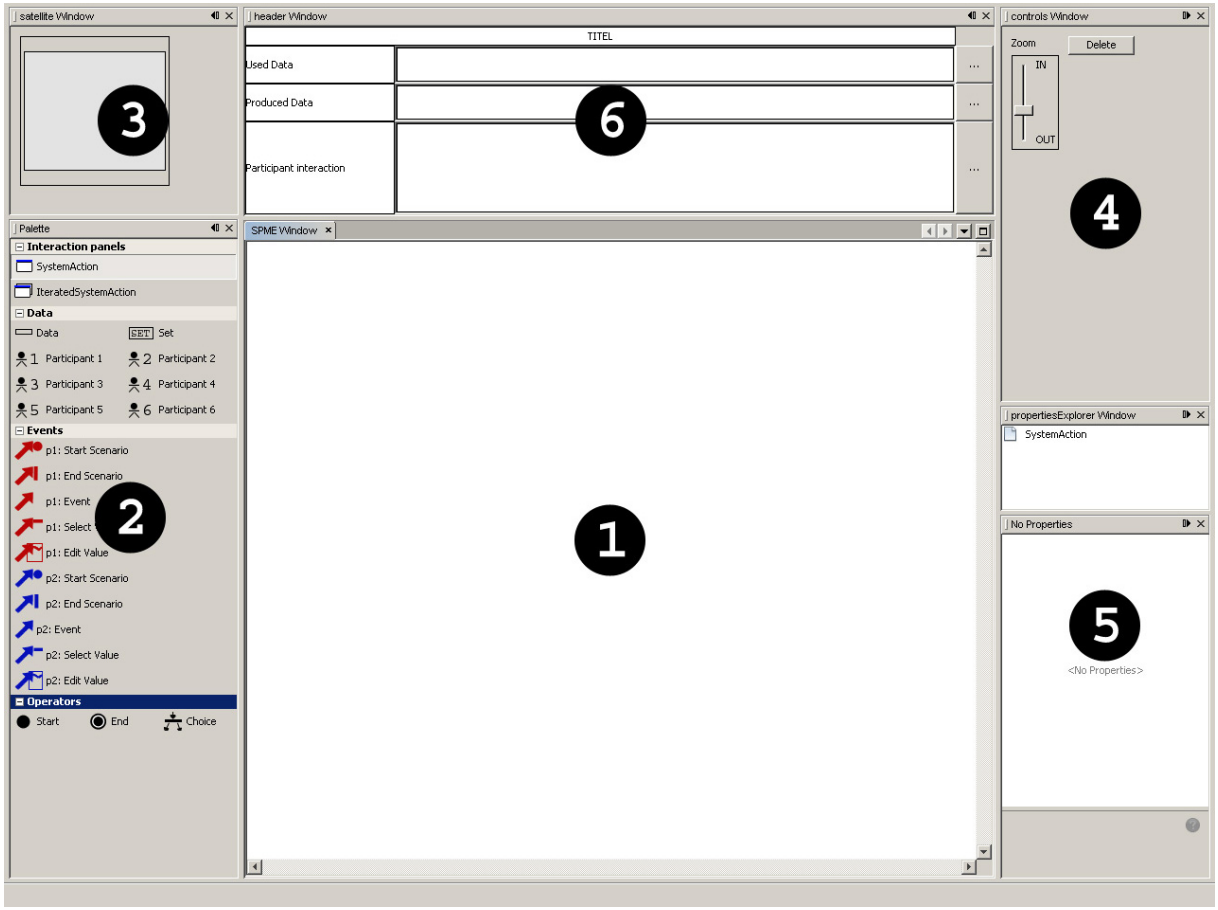
Hier bespreken we in het kort twee grafische model editors die geïmplementeerd zijn met behulp van Tlibrary, die hierboven in dit hoofdstuk besproken werd. De eerste is voor het SPME scene script, de ander voor een taakmodel.

4.4.1 SPME scene script

Een Staged Participatory Multimedia Event, of kortweg SPME, is een event waarin de regisseur en deelnemers gewone tv-kijkers zijn. Voor meer informatie over SPME refereren we graag naar [Vand08].

Een SPME kan opgezet worden aan de hand van enkele modellen in een graphical language. Het scene script is één zulk model hieruit.

Het scene script specificeert welke acties uitgevoerd kunnen worden door de kijkers, in welke volgorde, en wat hun resultaat is [Vand08].
Figuur 37 toont de implementatie hiervan in Tlibrary. Merk op dat de panelen vrij te verplaatsen zijn.



Figuur 37 - SPME scene script editor

We bespreken nu zeer beknopt de onderdelen van deze editor:

(1) Scene:
Hier zal het diagram in gebouwd worden.

(2) Palette:
Hier vindt men de bouwblokken nodig om het diagram op te bouwen. Deze kunnen toegevoegd worden aan het model door ze simpelweg van de palette naar de scene te verslepen.

(3) Satellite
In dit venster wordt een overzicht van de scene in het klein getoond, om zo sneller de algemene structuur te herkennen, en te zien op welk deel van het model de scene is ingezoomd.

(4) Controls

Met de slider kan de scene gezoomd worden, wat ook met behulp van het scrollwiel op de muis kan. De deleteknop dient om de geselecteerde nodes en edges te verwijderen.

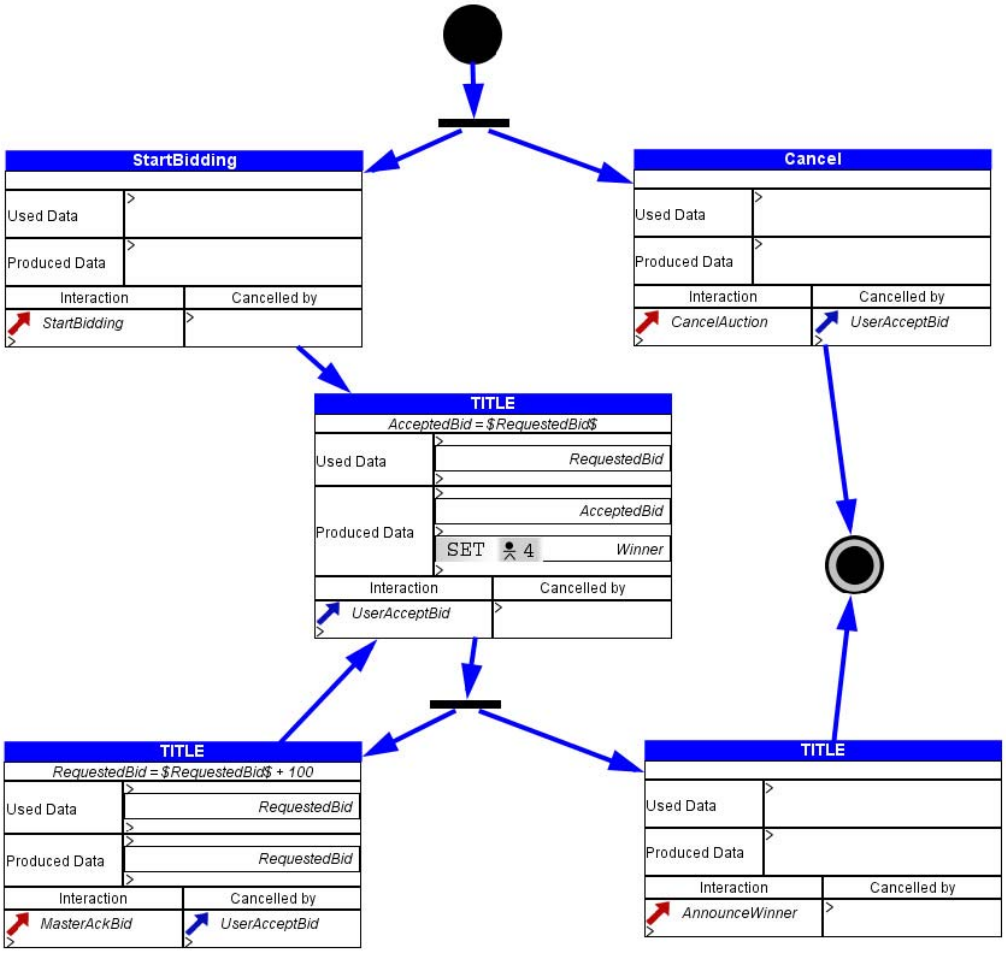
(5) Properties

Tlibrary biedt een zeer gelimiteerd basissysteem aan om properties aan een node toe te voegen. Hier is echter geen link met de scene, dus gemaakte veranderingen in properties die de developer zelf heeft toegevoegd (naast de standaard properties die altijd in Tlibrary zitten), hebben geen effect op het model.

(6) SPME header

Deze bevat de header naam van de spme-scene, rollen in de spme-scene, en gebruikte en gemaakte data gedurende de spme-scene. Hier is geen enkele link voorzien met de palette of de scene, en is hier dus enkel om esthetische redenen aanwezig. Het nut en de volledige uitleg over deze header, en SPME in het algemeen, vindt u in [Vand08].

Figuur 38 toont het scene script uit [Vand08], figuur 7, nagemaakt in deze editor. Dit model kon gemakkelijk opgesteld worden door de nodes uit de palette te slepen, waarna we de correcte items ook door een eenvoudige drag-and-drop konden toevoegen aan de nodes. Dit alles gebruik makende van de preview methoden en alle features van Tlibrary zoals uitgelegd in 4.2 en 4.3.



Figuur 38 - SPME scene script

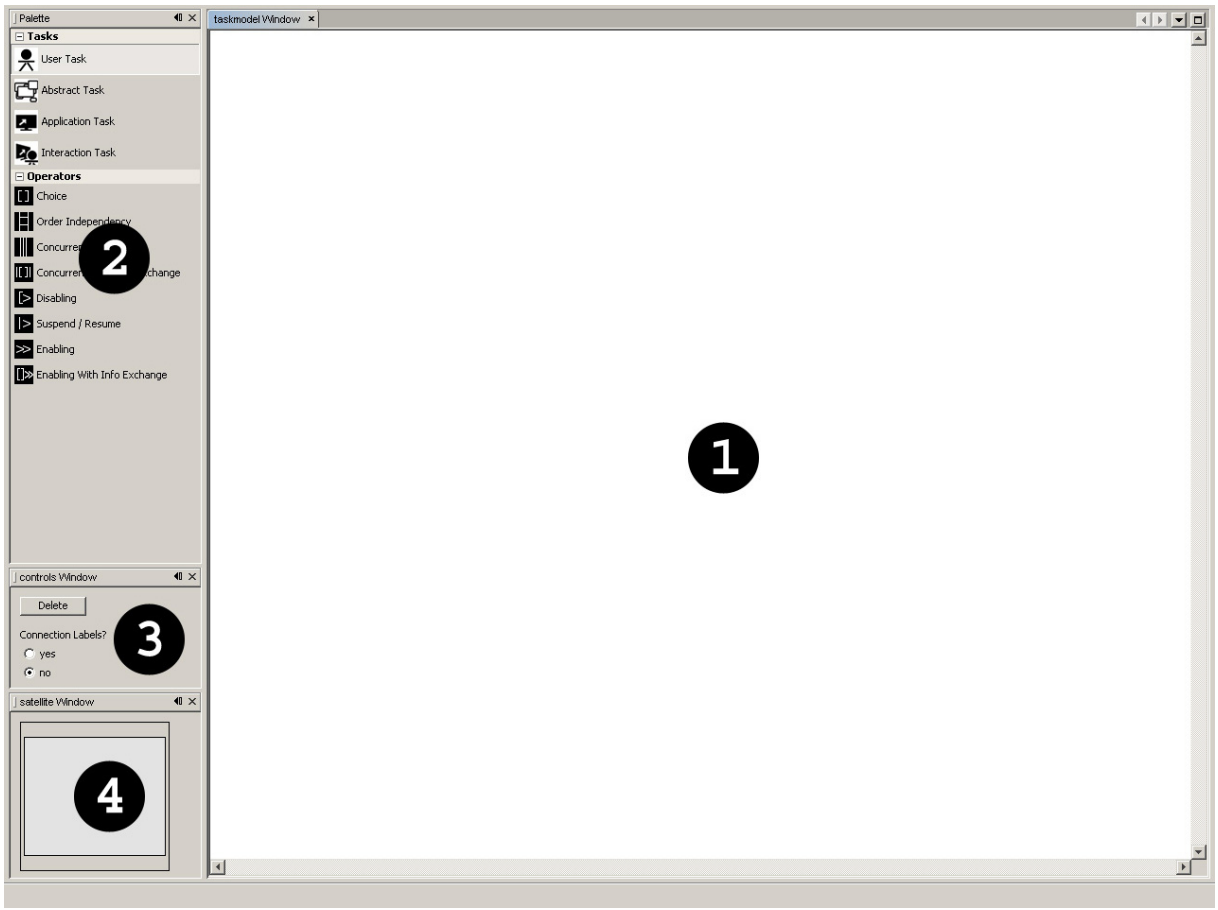
Hoeveel regels code heeft deze editor nu juist gekost?
Slechts om en bij de 1000 regels code, met de onderverdeling ongeveer als volgt:

- palette: 30 regels;
- header: 300 regels;
- satellite: 10 regels;
- controls: 30 regels;
- scene: 150 regels;
- nodes + items: 460 regels.

Exclusief de header, die in dit voorbeeld in principe onafhankelijk werkt van het model, zou deze editor te implementeren zijn met 700 regels code. Het aantal regels is grofweg geteld, zonder witregels etc. correct in acht te nemen, en zal in realiteit waarschijnlijk nog lager uitvallen. Bovendien is de geproduceerde code zeer eenvoudig te begrijpen en te schrijven.

4.4.2 Taakmodel

Deze eenvoudige taakmodel editor is snel geïmplementeerd om te controleren hoe flexibel en snel dit model te maken was in Tlibrary. Het resultaat ziet u in figuur 39. Ook hier zijn de panelen natuurlijk naar eigen wens te verplaatsen.

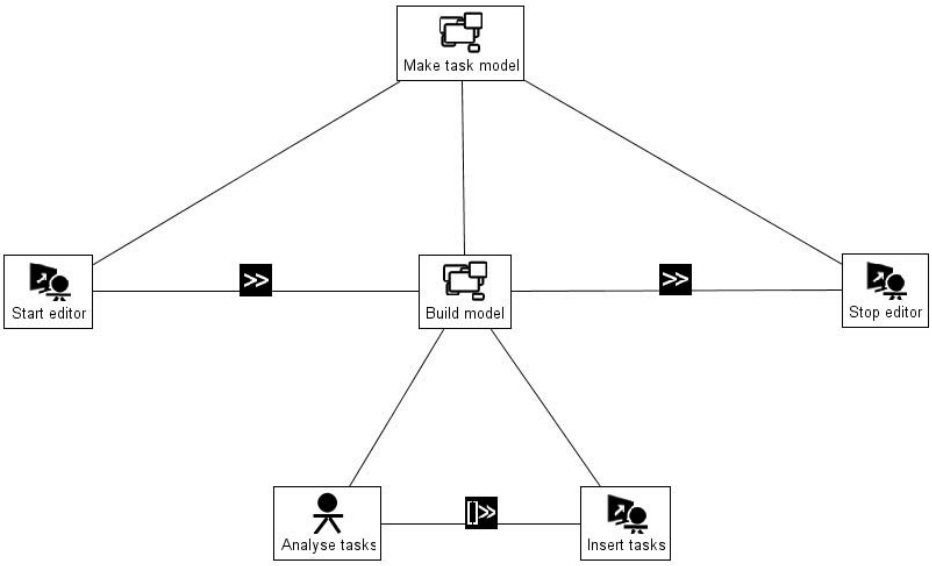


Figuur 39 - Taakmodel editor

De scene (1), de palette (2), de controls (3) en de satellite (4) herkent u uit 4.4.1., deze werken dan ook hetzelfde.

Door het gebrek aan een geschikte tree layout in Visual Library 2.0 is dit voorbeeld zonder deze layout geïmplementeerd, de nodes zijn dus vrij te verplaatsen. Bij het tekenen van een edge kan de gebruiker kiezen om een label eraan toe te voegen. Door middel van dit label kunnen verschillende temporele operators op een edge geplaatst worden door ze simpelweg uit de palette naar dat label te slepen. Dit alles natuurlijk weeral gebruik makend van de preview methoden zoals voorgesteld in 3.5, 3.6 en 3.7, en de features, inclusief de implementaties van die preview methoden, van Tlibrary zoals voorgesteld in 4.2 en 4.3.

Het kleine voorbeeld uit figuur 40 is met behulp van deze editor gecreëerd.



Figuur 40 - Taakmodel

Ook hier bekijken we hoeveel regels nodig waren om deze editor te implementeren:

- palette: 15 regels;
- satellite: 10 regels;
- controls: 55 regels;
- scene: 120 regels.

Dit voorbeeld is dus met slechts 200 regels code gerealiseerd, en hier vallen nog steeds enkele witregels en wat commentaarlijnen onder. Er werden geen specifieke nodes of items aangemaakt, aangezien deze simpele images zijn.
We moeten echter nog eens wijzen op het spijtige gebrek aan een automatische tree layout, wat in een taakmodel toch essentieel is.

4.5 Besluit

Dit hoofdstuk stelde de implementatie van Tlibrary voor. 4.2 liet ons kennis maken met de verschillende onderdelen van deze library. Speciale aandacht werd besteed aan de handige features die automatisch in de verschillende widget acties zitten. Zo implementeert de TNodeConnectAction bijvoorbeeld automatisch een systeem om de gebruiker tijdens het maken van een edge op de hoogte te houden van de mogelijkheden hiervan.

4.3 liet de implementaties van de preview methoden voorgesteld in 3.5, 3.6 en 3.7 in Tlibrary zien. Elke methode werd aan de hand van een bondige uitleg en de nodige figuren uit de doeken gedaan. De analyse achter de methoden vindt u in hoofdstuk 3 terug. 4.3 bewees dat deze methoden ook in de realiteit te gebruiken zijn, en niet enkel theoretisch van nut waren.

Tot slot beschreef 4.4 kort twee voorbeelden die met behulp van Tlibrary gecreëerd werden. Hierbij zagen we dat deze grafische editors met relatief weinig regels code gerealiseerd zijn.

De preview methoden vormden de basis voor de ontwikkeling van Tlibrary. Ze worden dan ook als standaard gebruikt in grafische editors gebouwd bovenop Tlibrary. Ook de TWidgetActions, die bovendien zeer eenvoudig toe te voegen zijn aan een node, hebben oog voor het gebruikersgemak. Zo probeert Tlibrary grafische model editors een basispakket van features aan te bieden die voor de developer eenvoudig te implementeren zijn, en voor de gebruikers eenvoudig en handig in gebruik zijn.

5 Evaluatie

5.1 Inleiding

In dit hoofdstuk gaan we de Tlibrary, voorgesteld in hoofdstuk 4, aan een evaluatie onderwerpen. Deze evaluatie kan als een tweeluik beschouwd worden. We kunnen de library op zich evalueren, of een applicatie gebouwd met behulp van die library, om zo de gebruikte methoden, met speciale aandacht voor de preview methoden die in hoofdstuk 3 werden voorgesteld, te kunnen evalueren.

De library evalueren we om te weten hoe handig hij in gebruik is, of hij gemakkelijk uitbreidbaar is, etc. Immers, als een library te moeilijk is in gebruik, gaan developers eerder de voorkeur geven aan een andere library die een gelijkaardige functionaliteit aanbiedt maar eenvoudiger te gebruiken is. Hetzelfde geldt voor de inhoud van de library, als deze onbelangrijk, overbodig, of zeer moeilijk uitbreidbaar is, zal eerder naar een beter alternatief gegrepen worden.

In hoofdstuk 3 hebben we op basis van enkele denkbare scenario's voor het opstellen van een model nieuwe methoden voorgesteld, de preview methoden. Per scenario werden telkens de vergelijkbare methoden besproken en geëvalueerd, om uiteindelijk tot een tijdsbesparende en gebruiksvriendelijkere methode te komen, die ook geëvalueerd werd.

Hoofdstuk 4 besprak de eigenlijke implementatie van de voorgestelde methoden. Op basis van de preview methoden werd een library bovenop Visual Library 2.0 gebouwd, zodat de nieuwe methoden eenvoudig te hergebruiken zijn in meerdere grafische model editors.

In dit hoofdstuk zullen we eerst Tlibrary als library zijnde aan een evaluatie onderwerpen, waarna we een typische grafische model editor, uiteraard geïmplementeerd met behulp van Tlibrary, zullen evalueren. Tot slot trekken we een beknopt besluit uit de uitgevoerde evaluaties.

5.2 Library evaluatie

Aangezien het hier een library betreft die boven Visual Library 2.0 is gebouwd, is deze natuurlijk ook afhankelijk van VL 2.0. Bovendien is Tlibrary eerst en vooral gebouwd om de nieuwe methoden te testen in applicaties, en is er pas in de tweede plaats voor gekozen om deze in library vorm aan te bieden. Toch is dit een filosofie die hier benadrukt zal worden, namelijk het aanbieden van kant en klare acties voor grafische model editors, zodat het geen tijdrovende bezigheid wordt voor de developer om in verschillende grafische editors zeer gelijkaardige acties te implementeren.

[Olse07] beschrijft waarom een standaard usability experiment moeilijk uit te voeren is voor het evalueren van een library of toolkit.
Voor een usability experiment is ten eerste een groep mensen nodig met vergelijkbare expertise. Dit is moeilijk te behalen voor een toolkit die zeer gespecialiseerde kennis vereist. De testpersonen moeten immers kennis van de toolkit hebben, wat voor een nieuwe toolkit

onmogelijk is. Ook een vergelijking met een bestaande toolkit zal moeilijk verlopen, aangezien de kennis van de andere toolkit een grote invloed zal hebben [Olse07].
Ten tweede is er moeilijk een taak te vinden om het experiment te specificeren. Een taak die met behulp van de te testen toolkit op te lossen is, zal een te grote inherente variatie hebben. Dit wil zeggen dat die taak op teveel manieren op te lossen is, zodat er niet kan aangetoond worden of de variatie in de resultaten aan de gebruikte technieken ligt, of aan andere factoren zoals de kennis van de testpersonen of hun aanpak van de taak. Dit blijkt een groot euvel te zijn, aangezien het juist die complexe problemen zijn die het gebruik van een toolkit zullen vereisen [Olse07].
Ten derde zal de schaal van het experiment te groot zijn. Usability tests moeten juist kort en relatief goedkoop zijn, wat niet meer opgaat voor het testen van een toolkit aangezien een applicatie maken meestal veel tijd en moeite, en dus ook geld, kost. Dit geldt des te meer wanneer een vergelijking met één of meerdere toolkits gemaakt moet worden [Olse07].

Hoe kunnen we dan deze library evalueren? [Olse07] stelt een evaluatie framework voor om User Interface Systems Research te kunnen evalueren. Hierin worden enkele belangrijke punten aangehaald waarop het product in beschouwing geëvalueerd kan worden. Het is dit framework dat we zullen hanteren om Tlibrary aan een evaluatie te onderwerpen. Per punt wordt uitgelegd wat het juist inhoudt, en natuurlijk hoe onze beschouwde library hierop presteert.

5.2.1 Importance

Eerst en vooral moet aangetoond worden dat de library in kwestie zijn nut heeft. Onder andere [Bott02], [Léde01] en [Ehri05] behandelen de vraag naar visual languages, hun bijhorende editors, en het gebruik van design modellen in het algemeen. Modellen worden niet enkel meer als design object gebruikt, maar ook vaker en vaker als manier van programmeren. De gebruiksvriendelijkheid van de grafische editors die daarmee hand in hand gaan, wordt dus alsmaar belangrijker. Dit is een weinig onderzocht onderwerp, waar deze thesis en de bijhorende library een invulling aan wil geven.

5.2.2 Problem not previously solved

Deze library snijdt geen volledig nieuw onderwerp aan. Usability is reeds lang een belangrijk issue waarvoor tal van onderzoekers in de Human-Computer Interaction wereld hard werk leveren. Deze library behandelt dus geen onopgelost probleem.
Maar doordat weinig onderzoek specifiek gericht is naar usability in grafische model editors, en door het groeiende publiek dat in aanraking komt met grafische editors voor het creëren van modellen, is verder onderzoek, en dus ook deze library, wel degelijk van nut.
Er zijn tal van nieuwe methoden geïmplementeerd in de library die gemakkelijk herbruikbaar zijn.

5.2.3 Generality

Wanneer door middel van de tool 3 verschillende problemen opgelost kunnen worden, wordt aangenomen dat die tool de meeste gelijkaardige problemen ook kan oplossen [Olse07]. Usability is echter niet meetbaar, dus deze stelling kan vaag geïnterpreteerd worden in deze library.

Tlibrary steunt op de preview methoden, beschreven in hoofdstuk 3, die tot een betere usability van grafische model editors moeten leiden. Als er na eventuele usability tests etc. besloten kan worden dat deze methoden effectief hun nut bewezen hebben, kan men eventueel

verder streven naar de implementatie van de gebruikte methoden in andere libraries, om zo de algemene usability in grafische editors vooruit te helpen.

5.2.4 Reduce solution viscosity

Belangrijk is dat de toolkit goed design aanmoedigt. Alternatieve oplossingen moeten gemakkelijk te implementeren zijn. Dus hoe moeilijker deze te maken zijn, hoe stugger de toolkit, hoe groter de viscositeit in het design proces [Olse07]. De viscositeit kan verkleind worden op 3 manieren:

Flexibility:
Wanneer gemakkelijk veranderingen in het design kunnen aangebracht worden, is de tool flexibel [Olse07]. Design veranderingen zijn moeilijk aan te brengen aangezien Tlibrary zeer strikt is in de opbouw van een applicatie. Bijvoorbeeld het toevoegen van nodes of edges via een andere manier dan de meegeleverde palette: deze activiteit is in de Tlibrary ingebouwd, maar alternatieven zijn echter niet voorhanden en moeilijk te implementeren met deze library. Hetzelfde geldt voor het toevoegen van items aan nodes, welke ook strak geregeld is door Tlibrary.

Om de flexibiliteit te vergroten moeten enkele mogelijke design veranderingen, die met weinig moeite te gebruiken zijn, ingebouwd worden [Olse07].

Expressive Leverage:
Expressive leverage bekomt men wanneer de designer meer kan bereiken met een minimum aan inspanning. Dit wordt vaak behaald door de designer minder keuzes te laten maken om zijn oplossing te bereiken [Olse07].
Dit gebeurt in Tlibrary vooral door het aanbieden van een kant en klare scene, met bijhorende TWidgetActions om nodes te selecteren, te verplaatsen, te verbinden, en te herverbinden.

Bijvoorbeeld door volgende 3 regels code kan een node in Tlibrary geselecteerd, bewogen en verbonden worden, allen met de features zoals besproken in hoofdstuk 4:

```
widgetcontainer.getActions().addAction(TNodeSelectAction());
widgetcontainer.getActions().addAction(TNodeConnectAction());
widgetcontainer.getActions().addAction(TNodeMoveAction());
```

Deze 3 regels alleen al kunnen de developer enkele honderden regels code besparen!

De ingebouwde palette, voorgesteld in 4.2.1, bespaart de developer ook enkele honderden regels code. Dit terwijl een palette een vaak, of zelfs bijna altijd, gebruikte optie is om objecten aan een model toe te voegen. In Tlibrary is deze ingebouwd en met enkele regels code te gebruiken.

Expressive Match:
Expressive match is een maatstaaf voor hoe dicht de middelen voor het uitdrukken van design keuzes liggen tegen het probleem dat opgelost moet worden [Olse07].
Wat dit nu juist betekent, kan het best uitgelegd worden aan de hand van een GUI builder waarin een GUI gemaakt wordt. Diezelfde GUI die met de GUI builder geïmplementeerd werd, kan ook in code gecreëerd worden, maar de expressive match is hoger bij de builder [Olse07]. Het visueel plaatsen van widgets op een dialoogvenster heeft immers een betere expressive match dan het plaatsen van die widgets aan de hand van coördinaten.

Dit handelt vooral over de gebruikte programmeertaal en environment, en zal hier niet behandeld worden. We gebruiken Tlibrary immers gewoon in Netbeans [NetB08a], welke hier niet geëvalueerd zal worden, en implementeren hiervoor geen extra tools.

5.2.5 Empowering new design participants

Stelt de tool nieuwe gebruikers in staat deel te nemen aan het UI design proces? Dit wordt vooral bereikt door een betere expressive leverage en expressive match [Olse07].
Aangezien Tlibrary in Netbeans gebruikt wordt en dus geen verbeterde expressive match zal hebben, zal Tlibrary geen nieuwe design participants bereiken.

Tlibrary maakt het voor bestaande design participants echter wel gemakkelijker. Door het beschikbaar stellen van kant en klare widget acties, en de te nemen beslissingen te beperken, zullen bestaande design participants sneller de stap naar deze library kunnen nemen dan naar een library waar deze vaak tijdsrovende en moeilijke opties zelf geïmplementeerd moeten worden. Zoals in 5.2.4 beschreven, komt dit door de verbeterde expressive leverage.
Doordat met behulp van deze library relatief gemakkelijk nieuwe grafische model editors kunnen gemaakt worden, kunnen meerdere design alternatieven getest en vergeleken worden.

5.2.6 Addressing a new problem

De library is gebaseerd op de preview methoden, beschreven in hoofdstuk 3. Deze methoden willen het voor de gebruiker zo gemakkelijk mogelijk maken om een model samen te stellen in een grafische editor. Aangezien er nog maar weinig onderzoek is verricht naar usability in het construeren van diagrammen of modellen in graafstructuur, kan dit enigszins beschouwd worden als een nieuw probleem, al is usability reeds lang een hot topic in de informatica.
Daarnaast moet het door de stijgende populariteit van modellen en hun bijhorende grafische editors mogelijk zijn om deze sneller en eenvoudiger te kunnen implementeren.

Tlibrary snijdt dus geen nieuw probleem aan, maar gaat eerder verder in het efficiënt creëren van grafische model editors, met automatische ondersteuning voor tal van methoden die de usability van die editors moeten verbeteren. Echter, zowel de efficiëntie van Tlibrary als de usability van de methoden voorgesteld in hoofdstuk 3 moeten verder onderzocht worden om tot definitieve conclusies te kunnen komen.

5.2.7 Power in combination

Als meerdere basiscomponenten gecombineerd kunnen worden tot een nieuwe oplossing, bewijst dit vaak de kracht van de toolkit [Olse07].

Inductive Combination:
Het idee hierachter is dat de toolkit enkele primitieve design componenten heeft, die gecombineerd kunnen worden tot complexere oplossingen [Olse07].
In Tlibrary wordt er echter geen basiscomponent gebruikt. Het TLabelWidget kan beschouwd worden als basis, maar hieruit is moeilijk een nieuw widget te ontwikkelen, tenzij door het combineren van verschillende TLabelWidgets en andere widgets tot een nieuw widget.
Een verdere uitbreiding van de Tlibrary is dan ook essentieel. Hier komt het grootste euvel van deze library naar boven, hij is er vooral op gericht om gemakkelijk en snel nieuwe grafische model editors te ontwikkelen, maar uitbreiding met nieuwe technieken en widgets is weinig tot niet ondersteund.

Simplifying Interconnection:

In veel gevallen moet de toolkit onder beschouwing kunnen communiceren met andere componenten. Dit kan zowel aan input als aan output kant gerealiseerd worden [Olse07].
Tlibrary bevat geen enkele mogelijkheid tot communicatie buiten de library. Hier moet verder onderzocht worden of er bepaalde standaarden zijn waarin modellen opgeslagen kunnen worden. Als library die eenvoudigheid promoot, zou Tlibrary ondersteuning moeten geven om deze standaarden zowel te kunnen openen als opslaan.
Ook het toevoegen van nieuwe TWidgetActions of andere nieuwe componenten zal een moeilijke opdracht zijn in Tlibrary, waardoor Tlibrary geen goede 'ease of combination' heeft.
In deze thesis concentreert Tlibrary zich echter vooral op de usability van de grafische editors, en de snelheid waarmee deze geïmplementeerd kunnen worden.

5.2.8 Can it scale up?

Is de library ook bij complexere problemen van nut?
Ook hier stoot men op de beperkingen van Tlibrary. Zolang er een eenvoudige grafische editor gebouwd moet worden, is er geen probleem. Voor de ontwikkeling van complexere editors met bijvoorbeeld complexe properties voor de nodes, kleurpaletten, persistentie van het model, etc. moet Tlibrary nog verder uitgebreid worden. Het implementeren van het eigenlijke model, zelfs in relatief complexe gevallen zoals het SPME scene script [Vand08], is echter geen probleem. Zolang er geen specifieke noden voor het model vereist zijn, voldoet Tlibrary voor de implementatie ervan.

Het aantal nodes dat per diagram op de scene gezet kan worden zonder een al te dramatische verlaging in performantie hangt grotendeels af van de complexiteit van die nodes, het gebruikte systeem, en van Visual Library 2.0. In het SPME scene script voorbeeld beschreven in 4.4.1 is een verlaagde performantie pas merkbaar wanneer er 20 nodes geplaatst zijn. Dit op een dual-core systeem van 2,1ghz en 1gb geheugen. Eén enkele node in dit voorbeeld bestaat uit een TContainerWidget met 13 TLabelWidgets erin geplaatst. 20 nodes zou in Visual Library 2.0 termen dus neerkomen op 280 labelwidgets. Verder onderzoek is hier geboden, ook naar de efficiëntie van de TWidgetActions.

5.3 Usability evaluatie

In hoofdstuk 3 werden enkele methoden, inclusief de nieuw voorgestelde preview methoden, voor verschillende scenario's reeds besproken en geëvalueerd aan de hand van GOMS modellen.

Aangezien er geen heuristieken beschikbaar zijn die specifiek gericht zijn op de usability van grafische model editors, gebruiken we hier de alom bekende 10 usability heuristieken van Nielsen [Jako08]. Deze bekijken we natuurlijk in het kader van grafische editors, wat enige aanpassing vereist ten opzicht van traditionele user interfaces. In dit onderdeel bespreken we een typische grafische model editor geïmplementeerd met Tlibrary. We baseren ons dus niet op één specifieke implementatie, om zo een ruimer beeld te krijgen van de features die Tlibrary ons aanbiedt.

5.3.1 Visibility of system status

Tlibrary biedt een infoMessage aan die automatisch geüpdate wordt met wat er gaande is in de TGraphScene. Het enige dat de developer moet doen, is deze optie implementeren in de

statusbalk, of waar hij maar wil. Een voorbeeld hiervan kan bekeken worden in figuur 23 en 25, waar de infoMessage getoond wordt in de statusbalk.

Naast de infoMessage geeft het systeem tijdens alle acties constant feedback over het effect op het model. Een duidelijk voorbeeld hiervan wordt getoond in figuur 22 en 24, waar dynamisch getoond wordt of een edge al dan niet aangemaakt kan worden. Ook tijdens de preview methoden wordt een constante feedback aangegeven. Hiervoor verwijzen we u naar hoodstuk 3 voor de omschrijving en analyse van alle methoden, en naar hoofdstuk 4 voor de eigenlijke implementaties en de volledige uitleg over de zojuist gerefereerde figuren.

5.3.2 Match between system and the real world

Aangezien de geproduceerde grafische model editors vaak gebruikt worden door mensen met gespecialiseerde kennis over het gebruikte model ervan, is deze heuristiek niet onmiddellijk van toepassing voor ons.

We kunnen hieronder eventueel wel verstaan dat de interactie met het model zo realistisch mogelijk moet zijn, zodat de match tussen het systeem en de echte wereld zo groot mogelijk is. Drag-and-drop is hiervoor uitermate geschikt. Nodes en items worden toegevoegd door ze uit een palette te slepen, waardoor de gebruiker het gevoel krijgt ze er letterlijk uit te slepen en in het model te zetten. Ook het verplaatsen van nodes kan door deze simpelweg te verslepen. Een mooie toegevoegde waarde zou zijn wanneer nodes ook met een drag-and-drop beweging weggegooid zouden kunnen worden.

5.3.3 User control and freedom

Undo en redo zijn helaas niet ondersteund door de library. Deze optie zou een welgekomen, of zelfs een verplichte, uitbreiding zijn.

Om verkeerde beslissingen te verkomen zijn echter wel tal van preview methoden gebruikt, om bijvoorbeeld items aan een node toe te voegen. Door deze preview methoden krijgt de gebruiker constant feedback over de status van het model, en kan hij zijn acties hierop afstellen. Deze methoden zijn erop gericht om vergissingen te voorkomen in plaats van naderhand op te lossen, en dit alles in een zo kort mogelijke uitvoeringstijd. Voorkomen is immers beter dan genezen, waardoor de uitvoeringstijd van een undo taak en het opnieuw, deze keer correct, uitvoeren van de ongedane taak volledig bespaard worden.

Voor verdere uitleg verwijzen we naar hoofdstuk 3, meerbepaald 3.5.2, 3.6.2 en 3.7, waar de preview methoden en de filosofie daarachter uitgebreid besproken en geanalyseerd worden. In 4.3 worden de uiteindelijke implementaties van de preview methoden, die ingebed zijn in Tlibrary, bekeken.

5.3.4 Consistency and standards

Het toevoegen van een node of item aan het model gebeurt op een consistente manier, beide kunnen eenvoudig toegevoegd worden door ze uit de palette en in het model te slepen.
Ook het verwijderen van een item uit een node werkt op een gelijkaardige wijze, deze kan immers eenvoudig uit de node gesleept worden om alzo verwijderd te worden. Op deze manier heeft de gebruiker het gevoel effectief objecten aan het model toe te voegen, en items te kunnen weggooien.

Het verwijderen van een edge of node daarentegen kan enkel door het selecteren van het gewenste object, om naderhand op de deleteknop te klikken. Op dezelfde manier kunnen meerdere nodes tegelijk verwijderd worden.

Het toevoegen van items of nodes werkt door een simpele drag-and-drop beweging, evenals het verwijderen van items uit een node. Een gelijkaardige actie voor het verwijderen van nodes zou dan ook interessant zijn, om zo tot een volledig consistent geheel te kunnen komen. Een mogelijkheid zou bijvoorbeeld zijn om een node letterlijk te kunnen weggooien door een drag beweging. We nemen de node vast en gooien deze met een snelle drag beweging weg, eventueel richting een prullenbak of ander icoon.

5.3.5 Error prevention

Door de constante feedback die gegeven wordt door de preview methoden, bekomen we een zeer goede error prevention.
Zo wordt er onder andere bij het maken van een edge gecontroleerd of deze wel mogelijk is, en wordt de gebruiker hier ook dynamisch over geïnformeerd, zoals in figuur 22 en 24 te zien is. Enkel als de edge toegelaten is door het systeem kan hij gemaakt worden. Hetzelfde geldt voor het herverbinden van bestaande edges.

Bij het verplaatsen van één of meerdere nodes wordt de locatie pas aangepast als de gewenste locatie ook effectief beschikbaar is. Zoniet wordt er een soort van spookversie van de versleepte nodes getoond om aan te geven dat de locatie waar de gebruiker ze wil plaatsen niet mogelijk is, zoals in figuur 28. Figuur 27 illustreert dat bij het automatisch aanpassen van de locatie bij het verslepen van één node, in geval van clipping met een andere node natuurlijk, ook wordt gecontroleerd of de voorgestelde locatie niet eventueel zou clippen met nog een andere node.

Verder kan een item enkel aan een node of een deel ervan worden toegevoegd als dat item ‘acceptable’ is op die plaats, waardoor zich geen problemen kunnen voordoen met incorrect geplaatste items. Meer informatie over deze acceptables is te vinden in hoofdstuk 4, meerbepaald 4.2.1, 4.2.2 en 4.2.4.

Er wordt dus alles in het werk gesteld om errors of foutieve acties te voorkomen, een gebruiker kan dus weinig tot niets verkeerd doen tijdens het opstellen van een diagram.

5.3.6 Recognition rather than recall

Hoe bijvoorbeeld nodes moeten worden toegevoegd aan het model wordt vaak niet aangegeven. Gebruikers zullen door de palette echter vaak automatisch een drag-and-drop beweging willen uitvoeren. Gebruikers met ervaring in grafische editors zullen hier zeker geen enkel probleem hebben. Het maken van edges kan voor de ongetrainde gebruiker ook eventueel een klein probleem vormen.

De preview methoden om items toe te voegen, te verwijderen, en om een node tussen een bestaande edge te zetten, stellen de gebruiker in staat onmiddellijk het effect van zijn acties te zien in het model. Voor de volledige uitleg over deze methoden verwijzen we u door naar respectievelijk 3.5, 3.6 en 3.7. De eigenlijke implementaties hiervan in Tlibrary worden in 4.3 besproken.
Wanneer een gebruiker bijvoorbeeld een item aan een node toevoegt, krijgt hij, zonder de muisknop los te laten, al te zien hoe, waar, en welk item toegevoegd wordt. Dit is duidelijk zichtbaar in 4.3.2. De naam en het icoon in een palette geven meestal wel een duidelijk idee

over welk item het gaat, maar door deze methode te hanteren zijn alle vergissingen uitgesloten.

De gebruiker heeft dankzij de preview methoden en de andere features in Tlibrary, voorgesteld in hoofdstuk 4, altijd een goed idee van hoe het model eruit ziet of eruit zal zien.

5.3.7 Flexibility and efficiency of use

Sneltoetsen voor het samenstellen van het model zijn niet beschikbaar, en ook moeilijk denkbaar. Al zouden sneltoetsen voor het toevoegen van een nieuwe node wel handig zijn, moeten enkele problemen hiermee in achting genomen worden, zoals waar die node terecht zal komen, wat door het gebrek aan automatische layout moeilijk is.

De voorgestelde methoden in hoofdstuk 3 zorgen voor een optimale uitvoeringstijd van de taken die door de gebruiker uitgevoerd moeten worden. Deze zijn voor zowel beginnelingen als voor gevorderde gebruikers zeer gemakkelijk te gebruiken.
3.3 beschrijft en analyseert de basistaken die tijdens het opstellen van een model vaak terugkomen. 3.4, 3.5 en 3.6 omschrijven elk een scenario, op basis waarvan een grondige analyse gemaakt wordt om de uitvoeringstijd van de bijhorende taken te reduceren tot een zo kort mogelijke tijd.

5.3.8 Aesthetic and minimalist design

Een standaard applicatie gemaakt met Tlibrary ziet er minimalistisch uit, maar beschikt toch over alle nodige functionaliteit. Met enkel de scene en de palette kan in principe al een volledig model samengesteld worden, aangezien nodes en items toegevoegd worden door drag-and-drop interactie. Voor het verplaatsen van één of meerdere nodes, of voor het herverbinden van edges, zijn geen aparte tools nodig, en dus bijgevolg ook geen overbodige menubalken.

Een satellite view kan eventueel in een panel getoond worden zodat de zichtbaarheid van dit panel zelf in te stellen is. Hetzelfde geldt voor het properties panel en eventueel andere panels met bijvoorbeeld de deleteknop in. Duidelijk is dat vooral de scene en de palette zeer belangrijk zijn, en dus het duidelijkst zichtbaar moeten zijn.

5.3.9 Help users recognize, diagnose, and recover from errors

De infoMessage, aangeboden door de TGraphScene, toont automatisch de status van de scene in zijn huidige staat, dus ook als er zich problemen voordoen. Willen we bijvoorbeeld een edge verbinden met een node die deze edge niet kan accepteren, wordt dit visueel duidelijk gemaakt, zoals zichtbaar is in figuur 22, en is in de infoMessage te lezen waarom dit probleem zich voordoet, wat we in figuur 23 kunnen zien. Figuur 23 geeft de infoMessage weer die bij figuur 22 hoort.
De infoMessage geeft dus telkens beknopt en in duidelijke taal weer wat het probleem juist is wanneer er zich een error voordoet.

Hetzelfde geldt natuurlijk tijdens het toevoegen van een item op een plaats waar deze niet acceptable is, etc. Telkens wordt zoveel mogelijk visueel duidelijk gemaakt wat er met het model gebeurt, wat uiteindelijk het uitgangspunt is van de preview methoden beschreven in hoofdstuk 3. In het bijzonder 3.5, 3.6 en 3.7 behandelen het nut van het tijdig geven van feedback, zodat gebruikers onmiddellijk hierop kunnen reageren.

5.3.10 Help and documentation

Documentatie over het geïmplementeerde model kan voorhanden zijn, al worden deze grafische model editors meestal gebruikt door mensen die reeds kennis hebben van het te maken model. Voor het gebruik van de preview methoden kan een korte inleiding aan de gebruiker volstaan, al is deze waarschijnlijk niet eens nodig, aangezien de methoden gebruik makend van drag-and-drop interactie zeer intuïtief werken.

5.4 Besluit

De sterke kant van Tlibrary is dat hij nieuwe methoden implementeert om de usability van zijn grafische editors te verbeteren. Deze methoden zijn beschikbaar via kant en klare widget acties waar de developer weinig tot niets aan moet aanpassen, wat er naast een enorme tijdswinst ook voor zorgt dat het programmeren van een grafische editor aanzienlijk eenvoudiger wordt. De gebruikte methoden, die vooral steunen op het tijdig en uitgebreid geven van feedback, helpen de gebruiker om op een eenvoudige en eenduidige manier de editor te gebruiken.

De zwakke kant van Tlibrary is dat hij in zijn huidige staat niet voldoende is om volledige complexe grafische model editors te creëren. Een uitbreiding van het aantal beschikbare widget acties, en de mogelijkheid deze via parameters aan te passen, zou een eerste stap in de goede richting zijn. Tevens leent de filosofie van Tlibrary zich perfect voor een grafische editor om andere grafische editors te maken, naar het voorbeeld van GMF, maar dan met eenvoudigere acties die ook oog hebben voor de gebruiksvriendelijkheid van de toekomstige gebruiker.

In hoofdstuk 3 zijn de preview methoden al uitgebreid besproken en geëvalueerd. Toch zijn uitgebreide user tests nodig om hier definitieve conclusies uit te trekken. Deze methoden kunnen dan een stap zijn naar verdere ontwikkelingen voor een verbeterde usability in grafische model editors.

Conclusie

Ondanks de alsmaar stijgende populariteit van modellen in al zijn mogelijkheden, wordt het belang van de grafische editors die gebruikt worden om modellen samen te stellen maar al te vaak onderschat. Laat die grafische editor voor het model nu juist als doel hebben het de gebruiker, die bijvoorbeeld in het geval van DSM ook een developer is, gemakkelijker te maken en hem in staat te stellen meer en betere resultaten te bekomen in minder tijd. De usability van grafische model editors is dus essentieel.
Meer informatie over de context van deze thesis vond u in de inleiding, hoofdstuk 1.

In hoofdstuk 2 werden enkele frameworks besproken die van toepassing kunnen zijn bij het maken van een grafische model editor. Veelal is een grondige kennis van het framework in kwestie nodig om er een goede editor mee te implementeren. Daarenboven is die implementatie ook vaak een tijdsrovende bezigheid door het gebrek aan standaard kant en klare objecten, zoals de vaak terugkomende palette, de mogelijkheid om nodes te bewegen, edges te herverbinden, etc.
Hieruit volgde een korte evaluatie en de verklaring voor de ontwikkeling van een nieuwe library die zich concentreert op de usability van de grafische model editor.

Om de usability te verbeteren bekeken we in hoofdstuk 3 eerst uitgebreid de basistaken waaruit het opbouwen van een model bestaat. Dit deel geeft een overzicht van de huidige mogelijkheden van grafische model editors, en geeft tevens de verschillen aan tussen de methoden die gebruikt kunnen worden om een taak te voltooien.
Op basis van enkele vaak voorkomende scenario's probeerden we de gebruiksvriendelijkheid van deze editors te verbeteren. Per scenario werden verschillende methoden bekeken om het scenario tot een goed einde te brengen. Elke methode werd uitvoerig besproken en geëvalueerd aan de hand van een figuur en een KLM. Telkens volgde er een methode die de tekortkomingen van de voorgaande probeerde te verbeteren, om zo tot een laatste voorgestelde methode te komen.
In de laatste methode van elk scenario is een zelfde filosofie te herkennen, het zo snel mogelijk geven van feedback aan de gebruiker zodat deze indien nodig nog tijdens de huidige taak kan bijsturen. Deze vroegtijdige feedback die aan de gebruiker aangeboden wordt, noemen we een preview. De methoden die deze filosofie volgen, noemen we dan ook zeer toepasselijk preview methoden.

Het grote voordeel van een preview is dat de gebruiker aanpassingen aan zijn huidige acties kan doen nog voor de volledige taak afgelopen is. De gebruiker kan beter inschatten wat het effect van zijn acties zal zijn, en op basis hiervan beslissingen nemen over het verder verloop van de taak. Zo besparen we de gebruiker van eventuele fouten, de nodige acties om dit ongedaan te maken, en vaak om de taak correct opnieuw uit te voeren.

Hoofdstuk 4 stelde Tlibrary voor, een library voor het creëren van grafische model editors die de voorgestelde preview methoden automatisch implementeren als gebruikte widget acties. Naast een bespreking van de opbouw van deze library, werd ook aandacht besteed aan alle features, zoals de preview methoden, die het samenstellen van een model zo gemakkelijk mogelijk proberen te maken voor de gebruiker.

Deze library en een typische editor die ermee gecreëerd is, inclusief alle features die in hoofdstuk 4 werden besproken, werden in hoofdstuk 5 aan een evaluatie onderworpen. Om de

library te evalueren gebruikten we het evaluatie framework uit [Olse07]. Door het gebrek aan heuristieken specifiek voor grafische model editors, werd de grafische editor geëvalueerd aan de hand van de alom bekende 10 usability heuristieken van Nielsen, welke in grote mate toepasselijk bleken te zijn.

Als volledige library volstaat Tlibrary helaas niet. Hiervoor zijn uitbreidingen als onder andere model persistentie, of de mogelijkheid om een modeling language te specificeren aan de hand van een meta-taal, nodig. Door de opkomst van Domain-Specific Modeling wordt ook code generatie uit modellen steeds belangrijker, een optie die in deze library ontbreekt.

Tlibrary is dus geen complete library die gebruikt kan worden om complexe en bruikbare grafische model editors te creëren. De library was eerder opgesteld met als bedoeling een richting aan te geven in library development voor grafische model editors, hoofdzakelijk in het aanbieden van kant en klare opties die eenvoudig geïmplementeerd kunnen worden. Zo zullen verscheidene acties over zeer veel grafische editors dezelfde zijn. Om de developer veel tijd en moeite te besparen zouden deze dus als standaard aangeboden kunnen worden. Hiernaast moeten die standaard acties niet enkel gemakkelijk te implementeren zijn, maar ook gebruiksvriendelijkheid hoog in het vaandel dragen. We willen immers dat de ontwikkelde grafische model editor aangenaam en tijdsbesparend zal zijn om mee te werken.

De gebruiksvriendelijkheid en de uitvoeringstijd van de preview methoden, die als basis worden gebruikt in Tlibrary, werden in hoofdstuk 3 uitvoerig besproken en geëvalueerd. Dit gebeurde niet enkel aan de hand van een KLM en zijn uitvoeringstijd, maar we hadden hier ook oog voor de simpliciteit en hoe de gebruiker met de methode zou omgaan. Toch moet er nog grondig onderzoek en eventuele experimenten plaatsvinden om het gedrag en de acties van gebruikers tijdens het maken van een model te analyseren.

Verder wordt een grafische model editor interessanter als er een automatische layout wordt toegepast op het model, zonder de gebruiker te beperken in zijn mogelijkheden. Dit op zich is een volledige onderzoekstak en geniet reeds veel aandacht in de academische wereld. Een kant en klare implementatie van verschillende layouts in frameworks zou zowel de developer als de gebruiker van de grafische editor veel tijd besparen. Daarnaast kan de juiste layout voor een bepaald model van toegevoegde waarde zijn voor de gebruiker, aangezien deze bepaalde relaties in de structuur van het model, en het overzicht van dat model, duidelijker kan maken.

Ook het herkennen van gestures kan handig zijn in een grafische model editor. Denken we bijvoorbeeld aan gestures om een edge van richting te veranderen, om een edge te verwijderen, of om een node te dupliceren. Tal van mogelijkheden zijn hier denkbaar.

Bibliografie

[Alde01]
Alder, G.: *The JGraph Swing Component*. Diploma thesis, Department of Computer Science, Federal Institute of Technology ETH, Zurich, Switzerland, 2001. Online op: http://www.alderg.com/ . Laatst bezocht: 30 april 2008.

[Bézi06]
Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: *Model-Based DSL Frameworks*. In Companion to the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, Portland, Oregon, USA, ACM Press, 2006, p.602-616.

[Bott02]
Bottoni, P., Costagliola, G.: *On the Definition of Visual Languages and Their Editors*. In Proceedings of the Second International Conference on Diagrammatic Representation and Inference, Springer-Verlag Berlin Heidelberg, 2002, p.305-319.

[Camp06]
Campos, P., Nunes, N. J.: *Principles and Practice of Work Style Modeling: Sketching Design Tools*. In Proceedings of HWID'06 - Human-Work Interaction Design, 2006.

[Card80]
Card, S. K., Moran, T. P., Newell, A.: *The Keystroke-Level Model for User Performance Time with Interactive Systems*. Commun. ACM 23, 7 July, 1980, p.396–410.

[Choc03]
Chock, S.S., Marriott, K.: *Automatic Generation of Intelligent Diagram Editors*. In ACM Transactions on Computer-Human Interaction, Vol.10 No.3, 2003, p.244-276.

[Davi08]
David E. Kieras, http://www.eecs.umich.edu/~kieras/ . Laatst bezocht: 30 april 2008.

[Dogr02]
Dogrusoz, U., Feng,Q., Madden, B., Doorley, M., Frick, A.: *Graph Visualization Toolkits*. In IEEE Computer Graphics and Applications, Vol.22 No.1, 2002, p.30-37.

[DSLT08]
Domain-Specific Language Tools, http://msdn2.microsoft.com/en-us/library/bb126235.aspx . Laatst bezocht: 30 april 2008.

[DSMF08]
DSM Forum: Domain-Specific Modeling, http://www.dsmforum.org/ . Laatst bezocht: 30 april 2008.

[Ecli08a]
Help - Eclipse SDK, http://help.eclipse.org/help33/index.jsp . Laatst bezocht: 30 april 2008.

[Ecli08b]
Eclipse, http://www.eclipse.org/ . Laatst bezocht: 30 april 2008.

[Ehri05]
Ehrig, K., Ermel, C., Hänsgen, S., Taentzer, G.: *Generation of Visual Editors as Eclipse Plug-Ins*. In Proceedings of the 20th IEEE/ACM International

Conference on Automated Software Engineering, New York, NY, USA, ACM Press, 2005, p.134-143.

[Ells03]
Ellson, J., Gansner, E., Koutsofios, E., North, S., Woodhull, G.: *Graphviz and Dynagraph—Static and Dynamic Graph Drawing Tools*. In Junger, M., Mutzel, P. (Eds.), Graph Drawing Software, Springer-Verlag, 2003, p.127-148.

[EMF08]
Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/ . Laatst bezocht: 30 april 2008.

[GEF08a]
Eclipse Graphical Editing Framework, http://www.eclipse.org/gef/ . Laatst bezocht: 30 april 2008.

[GEF08b]
Create an Eclipse-based Application Using the Graphical Editing Framework, http://www.ibm.com/developerworks/opensource/library/os-gef/ . Laatst bezocht: 30 april 2008.

[GMF08a]
Graphical Modeling Framework, http://www.eclipse.org/gmf/ . Laatst bezocht: 30 april 2008.

[GMF08b]
GMF Tutorial, http://wiki.eclipse.org/index.php/GMF_Tutorial . Laatst bezocht: 30 april 2008.

[GMF08c]
Learn Eclipse GMF in 15 Minutes, http://www-128.ibm.com/developerworks/opensource/library/os-ecl-gmf/ . Laatst bezocht: 30 april 2008.

[GMF08d]
Introducing the GMF Runtime, http://www.eclipse.org/articles/Article-Introducing-GMF/article.html . Laatst bezocht: 30 april 2008.

[Grap08]
Graphviz - Graph Visualization Software, http://www.graphviz.org/ . Laatst bezocht: 30 april 2008.

[Jako08]
Jakob Nielsen on Usability and Web Design, http://www.useit.com/ . Laatst bezocht: 30 april 2008.

[Jgra08]
Java Graph Visualization and Layout, http://www.jgraph.com/ . Laatst bezocht: 30 april 2008.

[John96]
John, B.E., Kieras, D.E..: *The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast*. In ACM Transactions on Computer-Human Interaction, Vol. 3, No. 4, 1996, p.320-351.

[Kier01]
Kieras, D.: *Using the Keystroke-Level Model to Estimate Execution Times*. The University of Michigan, Unpublished Report, 2001. Online op: ftp://www.eecs.umich.edu/people/kieras/GOMS/KLM.pdf . Laatst bezocht: 30 april 2008.

[Léde01]
Lédeczi, A., Bakay, A., Maroti, M., Völgyesi, P., Nordstrom, G., Sprinkle, J., Karsai, G.: *Composing Domain-Specific Design Environments*. IEEE Computer, 2001, p.44-51.

[Mina95]
Minas, M.,Viehstaedt, G.: *DiaGen: A Generator for Diagram Editors Providing Direct Manipulation and Execution of Diagrams*. In Proceedings of the 11th IEEE International Symposium on Visual Languages, 1995, p.203-210.

[NetB08a]
NetBeans, http://www.netbeans.org/ . Laatst bezocht: 30 april 2008.

[NetB08b]
NetBeans Visual Library 2.0, http://graph.netbeans.org/ . Laatst bezocht: 30 april 2008.

[NetB08c]
NetBeans Visual Library 2.0 Documentation, http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/doc-files/documentation.html . Laatst bezocht: 30 april 2008.

[Olse07]
Olsen, D.R.: *Evaluating User Interface Systems Research*. In Proceedings of the 20th annual ACM symposium on User interface software and technology, Newport, Rhode Island, USA, ACM, 2007, p.251-258.

[Pele06]
Pelechano, V., Albert, M., Muñoz, J., Cetina, C.: *Building tools for model driven developmens, comparing microsoft dsl tools and eclipse modeling plug-ins*. XV Jornadas de Ingenieria del Software y Bases de Datos, JISBD, 2006.

[Picc08]
Piccolo, http://www.cs.umd.edu/hcil/piccolo/index.shtml . Laatst bezocht: 30 april 2008.

[TomS08]
Tom Sawyer Software, http://www.tomsawyer.com/home/index.php . Laatst bezocht: 30 april 2008.

[Vand08]
Van den Bergh, J., Bruynooghe, B., Moons, J., Huypens, S., Hemmeryckx-Deleersnijder, B., Coninx, K.: *Using High-Level Models for the Creation of Staged Participatory Multimedia Events on TV*. In Multimedia Systems, Springer Berlin / Heidelberg, 2008. Online First, DOI: 10.1007/s00530-008-0116-2.

Appendix: Tlibrary javadoc

Deze appendix toont de javadoc van Tlibrary. Hierbij worden enkel de relevante klassen en functies uitgespreid. De volledige javadoc vindt u terug in Tlibrary.

Appendix: inhoudstabel

1 Tlibrary

Packages	
org.netbeans.tlibrary	
org.netbeans.tlibrary.palette	
org.netbeans.tlibrary.properties	

2 Package org.netbeans.tlibrary

Class Summary	
TContainerWidget	TContainerwidget moet altijd de container zijn van de node, elke node in de TGraphScene is dus een TContainerWidget.
TGraphScene	TGraphScene, de scene die gebruikt moet worden in een Tlibrary gebaseerd project.
TLabelWidget	TLabelWidget is hier de standaard widget in Tlibrary.
TNode	TNode is een node in een graaf-model (in een graphscene of TGraphScene dus).
TTableLayout	TTableLayout is een layout die de kinderen in een TLabelwidget langst elkaar plaatst volgens de in tabs meegegeven waarden.

2.1 Class TContainerWidget

```
java.lang.Object
  └org.netbeans.api.visual.widget.Widget
      └org.netbeans.tlibrary.TContainerWidget
```

All Implemented Interfaces:

javax.accessibility.Accessible

```
public class TContainerWidget
extends org.netbeans.api.visual.widget.Widget
```

TContainerwidget moet altijd de container zijn van de node, elke node in de TGraphScene is dus een TContainerWidget.

Author:

Frederik Smolders

Constructor Detail

TContainerWidget

```
public TContainerWidget(org.netbeans.api.visual.widget.Scene scene)
```

Constructor. Maakt nieuwe TContainerWidget, de scene moet meegegeven worden als parameter.

Parameters:

`scene` - is de scene waarin de TContainerWidget moet komen te staan.

Method Detail

enableChildren

`protected void` **`enableChildren`**`()`

---

isTitleEnabled

`public boolean` **`isTitleEnabled`**`()`

Zegt of de titel van de TContainerWidget enabled is.

Returns:

titleEnabled attribute die aanduidt of title enable is of niet.

---

disableTitle

`public void` **`disableTitle`**`()`

Disablet de titel van deze TContainerWidget.

---

setTitle

`public void` **`setTitle`**`(java.lang.String title)`

Stelt de titel van deze TContainerWidget in.

Parameters:

`title` - de string die als titel van deze TContainerWidget gebruikt zal worden.

---

getTitle

`public java.lang.String` **`getTitle`**`()`

Geeft de titel van de TContainerWidget terug als string.

Returns:

de string van de titel van deze TContainerWidget.

---

setColorTitleBackground

`public void` **`setColorTitleBackground`**`(java.awt.Color color)`

Stelt de kleur van de titelbalk achtergrond van deze TContainerWidget in.

Parameters:

`color` - de kleur die de titelbalk achtergrond van deze TContainerWidget moet krijgen.

---

getColorTitleBackground

`public java.awt.Color` **`getColorTitleBackground`**`()`

Geeft de kleur van de titelbalk achtergrond van deze TContainerWidget terug als Color.

Returns:

de Color van de titelbalk achtergrond van deze TContainerWidget.

---

setColorTitleForeground

`public void` **`setColorTitleForeground`**`(java.awt.Color color)`

Stelt de kleur van de titelbalk voorgrond van deze TContainerWidget in.

Parameters:

`color` - de kleur die de titelbalk voorgrond van deze TContainerWidget moet krijgen.

getColorTitleForeground

```
public java.awt.Color getColorTitleForeground()
```

Geeft de kleur van de titelbalk voorgrond van deze TContainerWidget terug als Color.

Returns:

de Color van de titelbalk voorgrond van deze TContainerWidget.

---

setColorTitleSelectedBackground

```
public void setColorTitleSelectedBackground(java.awt.Color color)
```

Stelt de kleur van de titelbalk achtergrond van deze TContainerWidget in wanneer deze geselecteerd is.

Parameters:

`color` - de kleur die de titelbalk achtergrond van deze TContainerWidget moet krijgen wanneer deze geselecteerd is.

---

getColorTitleSelectedBackground

```
public java.awt.Color getColorTitleSelectedBackground()
```

Geeft de kleur van de titelbalk achtergrond van deze TContainerWidget wanneer deze geselecteerd is terug als Color.

Returns:

de Color van de titelbalk achtergrond van deze TContainerWidget wanneer deze geselecteerd is.

---

setColorTitleSelectedForeground

```
public void setColorTitleSelectedForeground(java.awt.Color color)
```

Stelt de kleur van de titelbalk voorgrond van deze TContainerWidget in wanneer deze geselecteerd is.

Parameters:

`color` - de kleur die de titelbalk voorgrond van deze TContainerWidget moet krijgen wanneer deze geselecteerd is.

---

getColorTitleSelectedForeground

```
public java.awt.Color getColorTitleSelectedForeground()
```

Geeft de kleur van de titelbalk voorgrond van deze TContainerWidget wanneer deze geselecteerd is terug als Color.

Returns:

de Color van de titelbalk voorgrond van deze TContainerWidget wanneer deze geselecteerd is.

---

setTitleBackgroundColor

```
public void setTitleBackgroundColor(java.awt.Color unselColor,
                                    java.awt.Color selColor)
```

Stelt de kleur van de titelbalk achtergrond van deze TContainerWidget in, wanneer deze niet en wel geselecteerd is.

Parameters:

`unselColor` - de kleur die de titelbalk achtergrond van deze TContainerWidget moet krijgen wanneer deze niet geselecteerd is.
`selColor` - de kleur die de titelbalk achtergrond van deze TContainerWidget moet krijgen wanneer deze geselecteerd is.

---

setTitleForegroundColor

```
public void setTitleForegroundColor(java.awt.Color unselColor,
                                    java.awt.Color selColor)
```

Stelt de kleur van de titelbalk voorgrond van deze TContainerWidget in, wanneer deze niet en wel geselecteerd is.

Parameters:

`unselColor` - de kleur die de titelbalk voorgrond van deze TContainerWidget moet krijgen wanneer deze niet geselecteerd is.
`selColor` - de kleur die de titelbalk voorgrond van deze TContainerWidget moet krijgen wanneer deze geselecteerd is.

2.2 Class TGraphScene

```
java.lang.Object
  └org.netbeans.api.visual.widget.Widget
      └org.netbeans.api.visual.widget.Scene
          └org.netbeans.api.visual.model.ObjectScene
              └
org.netbeans.api.visual.graph.GraphScene<TNode,java.lang.String>
                  └org.netbeans.tlibrary.TGraphScene
```

All Implemented Interfaces:

javax.accessibility.Accessible

```
public abstract class TGraphScene
extends org.netbeans.api.visual.graph.GraphScene<TNode,java.lang.String>
```

TGraphScene, de scene die gebruikt moet worden in een Tlibrary gebaseerd project. Dit is een abstract class en moet eerst uitgebreid worden met een eigen addNode method.

Author:

Frederik Smolders

Field Detail

mainLayer

```
public org.netbeans.api.visual.widget.LayerWidget mainLayer
```

Deze layer moet alle nodes bevatten.

---

connectionLayer

`public org.netbeans.api.visual.widget.LayerWidget` **`connectionLayer`**

Deze layer moet alle edges bevatten.

Constructor Detail

TGraphScene

`public` **`TGraphScene`**`()`

Constructor. Maakt nieuwe TGraphscene.

Method Detail

attachNodeWidget

`protected abstract org.netbeans.api.visual.widget.Widget`
`attachNodeWidget``(TNode node)`

Voegt een nieuwe node toe. Deze moet geïmplementeerd worden om op basis van de naam van de node verschillende soorten nodes toe te laten (bv een node, start, stop, choice-operator).

Specified by:

`attachNodeWidget` in class `org.netbeans.api.visual.graph.GraphScene<TNode,java.lang.String>`

Parameters:

`node` - de node die uit de palette gesleept wordt. Op basis hiervan kunnen we beslissen wat we aan de scene gaan toevoegen.

---

attachEdgeWidget

`protected org.netbeans.api.visual.widget.Widget`
`attachEdgeWidget``(java.lang.String edge)`

Voegt een nieuwe edge toe. Indien nodig kan deze ook ge-override worden om custom connections aan te maken.

Specified by:

`attachEdgeWidget` in class `org.netbeans.api.visual.graph.GraphScene<TNode,java.lang.String>`

Parameters:

`edge` - de naam van de edge die gemaakt moet worden (moet niets eme gedaan worden).

---

attachEdgeSourceAnchor

```
protected void attachEdgeSourceAnchor(java.lang.String edge,
                                      TNode oldSourceNode,
                                      TNode sourceNode)
```

Specified by:

`attachEdgeSourceAnchor` in class `org.netbeans.api.visual.graph.GraphScene<TNode,java.lang.String>`

---

attachEdgeTargetAnchor

```
protected void attachEdgeTargetAnchor(java.lang.String edge,
                                      TNode oldTargetNode,
                                      TNode targetNode)
```

Specified by:

attachEdgeTargetAnchor in class
org.netbeans.api.visual.graph.GraphScene<TNode,java.lang.String>

---

addAcceptable

```
public void addAcceptable(java.lang.String widgetName,
                          int amount)
```

Voegt een acceptable toe aan deze TGraphScene.

Parameters:

widgetName - de naam van de acceptable die toegevoegd moet worden.

amount - het aantal dat acceptable is voor de meegeleverde widgetName.

---

isAcceptable

```
public java.lang.Boolean isAcceptable(java.lang.String widgetName)
```

Geeft in een boolean weer of een bepaalde widgetName acceptable is in deze TGraphScene.

Returns:

true wanneer de widgetName een acceptable is in deze TGraphScene.

---

setPreviewInWidget

```
protected void
setPreviewInWidget(org.netbeans.api.visual.widget.Widget widget,
                                    java.lang.Boolean bool)
```

Waarschuwt de TGraphScene dat er ergens een preview getoond wordt. Deze functie wordt enkel gebruikt in TLabelWidgets, die de TGraphScene laten weten waar de preview getoond wordt previewInWidget houdt constant bij of er een preview getoond wordt ergens, en in previewedWidget in welke node die preview getoond wordt

---

getCurrentMousePoint

```
protected java.awt.Point getCurrentMousePoint()
```

---

changeZoom

```
public void changeZoom(double zoomfactor)
```

Stelt de zoomfactor van de TGraphScene in.

Parameters:

zoomfactor - de zoomfactor die voor deze TGraphScene ingesteld wordt.

---

getInfoMessage

```
public java.lang.String getInfoMessage()
```

Geeft de ingebouwde infoMessage van deze TGraphScene terug.

Returns:

de string die de huidige staat van deze TGraphScene weergeeft.

---

setInfoMessage

```
public void setInfoMessage(java.lang.String message)
```

Stelt de infoMessage van de TGraphScene in, als we deze willen aanpassen.

Parameters:
`message` - de infoMessage die voor deze TGraphScene ingesteld wordt.

---

getInfoImage

`public java.awt.Image` **`getInfoImage`**`()`

Geeft de ingebouwde infoImage, de grafische tegenhanger van de infoMessage, van deze TGraphScene terug. Dit is niet verder uitgewerkt en wordt niet gebruikt in de voorbeelden.

Returns:
de image die de huidige staat van deze TGraphScene weergeeft.

---

setInfoImage

`public void` **`setInfoImage`**`(java.lang.String imageLocation)`

Stelt de infoImage van de TGraphScene in, als we deze willen aanpassen.

Parameters:
`imageLocation` - de locatie van de infoImage die voor deze TGraphScene ingesteld wordt.

---

getCurrentThumbnail

`protected java.awt.Image` **`getCurrentThumbnail`**`()`

Functie gebruikt door TLabelWidget.

---

deleteSelectedNodes

`public void` **`deleteSelectedNodes`**`()`

Verwijdert de huidig geselecteerde node of edge, of de selectie van nodes.

---

setColorEdge

`public void` **`setColorEdge`**`(java.awt.Color color)`

Stelt de kleur van de edges in deze TGraphScene in.

Parameters:
`color` - de kleur die de edges in deze TGraphScene moeten hebben.

---

colorEdge

`public java.awt.Color` **`colorEdge`**`()`

Geeft de kleur van de edges in deze TGraphScene terug als Color.

Returns:
de kleur van de edges in deze TGraphScene.

---

setColorEdgeSelected

`public void` **`setColorEdgeSelected`**`(java.awt.Color color)`

Stelt de kleur van de edges in deze TGraphScene in wanneer deze geselecteerd zijn.

Parameters:
`color` - de kleur die de edges in deze TGraphScene moeten hebben wanneer ze geselecteerd zijn.

---

colorEdgeSelected

`public java.awt.Color` **`colorEdgeSelected`**`()`

Geeft de kleur van de edges in deze TGraphScene wanneer deze geselecteerd zijn terug als Color.

Returns:

de kleur van de edges in deze TGraphScene wanneer ze geselecteerd zijn.

---

setAutomaticReconnect

`public void` **`setAutomaticReconnect`**`(boolean bool)`

Stelt in of een edge automatisch gereconnect moet worden als er een node verwijderd wordt. Dit is huidig alleen mogelijk als er uit de te verwijderen node juist één edge vertrekt en één aankomt. Wanneer we die node verwijderen, worden die twee edges als het ware samengevoegd tot één.

Parameters:

`bool` - stelt in of een edge automatisch gereconnect moet worden als er een node verwijderd wordt.

---

getAutomaticReconnect

`public boolean` **`getAutomaticReconnect`**`()`

Geeft terug of automaticReconnect al dan niet aanstaat in deze tGraphScene.

Returns:

boolean of automaticReconnect al dan niet aanstaat in deze tGraphScene.

---

getMainLayer

`public org.netbeans.api.visual.widget.LayerWidget` **`getMainLayer`**`()`

Geeft de mainLayer van deze TGraphScene terug, deze is ook rechtstreeks te benaderen.

Returns:

de mainLayer van deze TGraphScene.

---

getConnectionLayer

`public org.netbeans.api.visual.widget.LayerWidget` **`getConnectionLayer`**`()`

Geeft de connectionLayer van deze TGraphScene terug, deze is ook rechtstreeks te benaderen.

Returns:

de connectionLayer van deze TGraphScene.

TNodeMoveAction

`public org.netbeans.api.visual.action.WidgetAction` **`TNodeMoveAction`**`()`

TNodeMoveAction is een MoveAction voor nodes in TGraphScene. Door deze actie aan nodes toe te voegen kunnen nodes verplaats worden (features als multi-move, clipping detectie enz zijn ingebouwd).

TNodeSelectAction

`public org.netbeans.api.visual.action.WidgetAction` **`TNodeSelectAction`**`()`

TNodeSelectAction is een SelectAction voor nodes in TGraphScene. Door deze actie aan nodes toe te voegen kunnen nodes geselecteerd worden.

TNodeConnectAction

`public org.netbeans.api.visual.action.WidgetAction` **`TNodeConnectAction`**`()`

TNodeConnectAction is een ConnectAction voor nodes in TGraphScene. Door deze actie aan nodes toe te voegen kunnen nodes met elkaar verbonden worden.

TEdgeSelectAction

`public org.netbeans.api.visual.action.WidgetAction` **`TEdgeSelectAction`**`()`

TEdgeSelectAction is een SelectAction voor edges in TGraphScene. Door deze actie aan edges toe te voegen kunnen edges geselecteerd worden (nodig om ze te verwijderen, of te herverbinden). Deze kan gebruikt worden bij custom edges.

TEdgeReconnectAction

`public org.netbeans.api.visual.action.WidgetAction` **`TEdgeReconnectAction`**`()`

TEdgeSelectAction is een SelectAction voor edges in TGraphScene. Door deze actie aan edges toe te voegen kunnen edges herverbonden worden. Deze kan gebruikt worden bij custom edges.

TEdgeOrthogonalRouter

`public org.netbeans.api.visual.router.Router` **`TEdgeOrthogonalRouter`**`()`

TEdgeOrthogonalRouter is een router voor edges in TGraphScene. TEdgeOrthogonalRouter zoekt automatisch een pad tussen de gewenste nodes. Deze kan gebruikt worden bij custom edges.

TEdgeDirectRouter

`public org.netbeans.api.visual.router.Router` **`TEdgeDirectRouter`**`()`

TEdgeDirectRouter is een router voor edges in TGraphScene. TEdgeDirectRouter maakt edges met een rechtstreekse verbinding tussen de gewenste nodes. Deze kan gebruikt worden bij custom edges.

---

setEdgeRouter

`public void` **`setEdgeRouter`**`(java.lang.String edgeRouter)`

Stelt de gebruikte router voor de edges in deze TGraphScene in.

Parameters:

`edgeRouter` - de naam van de te gebruiken router ("orthogonal" of "direct").

---

setNodeSelectBorder

`public void`
`setNodeSelectBorder``(org.netbeans.api.visual.border.Border border)`

Stelt de border in voor de nodes in deze TGraphScene wanneer ze geselecteerd zijn.

Parameters:

`border` - de Border die gebruikt moet worden wanneer een node geselecteerd wordt.

---

setNodeSelectBorderSwing

`public void` **`setNodeSelectBorderSwing`**`(javax.swing.border.Border border)`

Stelt de border in voor de nodes in deze TGraphScene wanneer ze geselecteerd zijn.

Parameters:

`border` - de SwingBorder die gebruikt moet worden wanneer een node geselecteerd wordt.

---

setEdgeLoopPermitted

`public void` **`setEdgeLoopPermitted`**`(boolean bool)`

Stelt voor deze TGraphScene in of looping edges toegelaten zijn, dit is enkel mogelijk wanneer de orthogonalRouter voor edges gebruikt wordt.

Parameters:

`bool` - stelt voor deze TGraphScene in of looping edges toegelaten zijn.

---

getIsEdgeLoopPermitted

`public boolean` **`getIsEdgeLoopPermitted`**`()`

Geeft terug of voor deze TGraphScene in of looping edges toegelaten zijn.

Returns:

boolean om aan te duiden of voor deze TGraphScene in of looping edges toegelaten zijn.

2.3 Class TLabelWidget

```
java.lang.Object
  └org.netbeans.api.visual.widget.Widget
      └org.netbeans.api.visual.widget.LabelWidget
          └org.netbeans.tlibrary.TLabelWidget
```

All Implemented Interfaces:

javax.accessibility.Accessible

```
public class TLabelWidget
extends org.netbeans.api.visual.widget.LabelWidget
```

TLabelWidget is hier standaard widget in de Tlibrary. Een node (dus een TContainerwidget) kan volledig opgevuld worden met deze TLabelWidgets. Aan deze widgets kunnen acceptables toegevoegd worden, waardoor items uit de palette in de gewenste TLabelWidget gesleept kunnen worden.

Author:

Frederik Smolders

Constructor Detail

TLabelWidget

```
public TLabelWidget(org.netbeans.api.visual.widget.Scene scene,
                    java.lang.String label)
```

Constructor. Maakt nieuwe TLabelWidget. Deze optie wordt meestal gebruikt om gewone labels aan te maken, deze zullen dus normaal ook geen acceptables hebben.

Parameters:

`scene` - is de scene waarin de TLabelWidget moet komen te staan

`label` - is de naam van het label dat in de TLabelWidget moet komen te staan.

TLabelWidget

```
public TLabelWidget(org.netbeans.api.visual.widget.Scene scene)
```

Constructor. Maakt nieuwe TLabelWidget. Hier is geen String label vereist, deze optie wordt meestal gebruikt als we later items in deze TLabelWidget zullen slepen.

Parameters:

`scene` - is de scene waarin de TLabelWidget moet komen te staan

Method Detail

registerWidget

```
public void registerWidget(org.netbeans.api.visual.widget.Widget widget,
                           java.awt.Point point,
                           org.netbeans.api.visual.widget.Widget newWidget)
```

Wanneer een item met widgetcategory "widget" over een TLabelWidget gesleept wordt (en deze acceptable is natuurlijk) wordt de functie 'accepwidget' uitgevoerd. Deze moet door de developer zelfs geïmplementeerd worden door een override. In die accepwidget functie kunnen we op basis van de titel van het item beslissen welke widget aan de TLabelwidget moet worden toegevoegd (gelijkend op de manier waarop nodes aan de TGraphScene worden toegevoegd). Dat preview widget moeten we via deze registerWidget functie registreren. Waarom? Om deze eventueel te kunnen verwijderen als men verder sleept. Deze functie vervangt dus als het ware de addChild.

Parameters:

`widget` - het huidige widget.

`point` - het point waar de muis het TLabelWidget is binnegekomen tijdens het slepen van een nieuw item naar de TLabelWidget.

`newWidget` - het nieuwe widget dat men heeft aangemaakt als reactie op het slepen van een item in de TLabelWidget. Dit nieuwe widget (het item dus) moet registreerd worden.

---

addAcceptable

```
public void addAcceptable(java.lang.String widgetName,
                          int amount)
```

Voegt een acceptable toe aan deze TLabelWidget.

Parameters:

`widgetName` - de naam van de acceptable die toegevoegd moet worden.

`amount` - het aantal dat acceptable is voor de meegeleverde widgetName.

---

setSpacersBetween

`public void` **`setSpacersBetween`**`(boolean bool)`

Moet er tussen elk kind van deze TLabelwidget een spacer gebruikt worden? Een spacer is nodig wanneer we de volgorde bij het toevoegen van een nieuw item willen bepalen. Slepen we bijvoorbeeld een nieuw item in de bovenste spacer, zal dat nieuwe item helemaal vanboven toegevoegd worden, in plaats van vanonder.

Parameters:

`bool` - duidt aan of er spacersBetween gebruikt moeten worden.

---

getSpacersBetween

`public boolean` **`getSpacersBetween`**`()`

Zegt of er spacersBetween gebruikt worden.

Returns:

boolean die zegt of er spacersBetween gebruikt worden.

---

setNoSpacer

`public void` **`setNoSpacer`**`()`

Stel in dat er voor deze TlabelWidget helemaal geen spacers gebruikt moeten worden. (door een acceptable toe te voegen wordt automatisch een spacer toegevoegd)

---

isAcceptable

`public java.lang.Boolean` **`isAcceptable`**`(java.lang.String widgetName)`

Geeft in een boolean weer of een bepaalde widgetName acceptable is in deze TGraphScene.

Returns:

true wanneer de widgetName een acceptable is in deze TLabelWidget.

---

disable

`public void` **`disable`**`()`

---

enable

`public void` **`enable`**`()`

---

setAmountAcceptables

```
public void setAmountAcceptables(java.lang.String layout,
                                 int amount)
```

Stelt in hoeveel kinderen (items) er mogelijk zijn in deze TLabelwidget. Dit is een algemeen aantal, ongeacht welke acceptable. Per acceptable kan ingesteld worden hoeveel er van dat type mogelijk zijn, wanneer de acceptable wordt toegevoegd met addAcceptable.

Parameters:

`layout` - de layout die gebruikt wordt om de kinderen te verdelen. Dit kan horizontal, vertical, of absolute zijn.

`amount` - het aantal kinderen dat er mogelijk zijn in deze TLabelwidget.

---

setEditable

`public void` **`setEditable`**`(java.lang.Boolean bool)`

Stelt in of de TLabelWidget te editeren is, zo kan zijn label aangepast worden. dit is normaliter niet van toepassen op een TLabelWidget die acceptables heeft.

Parameters:

`bool` - stelt in of de TLabelWidget te editeren is.

---

setRemovable

`public void` **`setRemovable`**`()`

Wanneer een item met widgetcategory "widget" over een TLabelWidget gesleept wordt (en deze acceptable is natuurlijk) wordt de functie 'accepwidget' uitgevoerd. Deze moet door de developer zelfs geëmplementeerd worden door een override. In die accepwidget functie kunnen we op basis van de titel van het item beslissen welke widget aan de TLabelwidget moet worden toegevoegd (gelijken op de manier waarop nodes aan de TGraphScene worden toegevoegd). Dat preview widget moet ook bestaan uit TLabelWidgets. Als we het gemaakte widget eventueel terug willen verwijderen uit de node, moet hij ingesteld worden als removable door deze functie.

---

acceptWidget

```
public void acceptWidget(org.netbeans.api.visual.widget.Widget widget,
                         java.awt.Point point,
                         PaletteItem item)
```

Wanneer een item met widgetcategory "widget" over een TLabelWidget gesleept wordt (en deze acceptable is natuurlijk) wordt de functie 'accepwidget' uitgevoerd. Deze moet door de developer zelfs geïmplementeerd worden door een override. In die accepwidget functie kunnen we op basis van de titel van het item beslissen welke widget aan de TLabelwidget moet worden toegevoegd (gelijken op de manier waarop nodes aan de TGraphScene worden toegevoegd). Dat preview widget moeten we via deze registerWidget functie registreren. Waarom? om deze eventueel te kunnen verwijderen als men verder sleept. Standaard is deze acceptWidget functie leeg, hij is enkel te implementeren door de developer, het is aan hem deze te overrider en te declareren wat er moet gebeuren als een item met widgetCategorie "widget" geaccept wordt!

Parameters:

`widget` - het huidige widget.

`point` - het point waar de muis het TLabelWidget is binnegekomen.

`item` - het paletteItem dat uit het palette is gesleept. Hierop moet men afhankelijk van de naam op reageren.

2.4 Class TNode

```
java.lang.Object
  └org.netbeans.tlibrary.TNode
```

```
public class TNode
extends java.lang.Object
```

TNode is een node in een graaf-model (in een graphscene of TGraphScene dus). Hier wordt de informatie over de node opgeslagen, zoals bijvoorbeeld hoeveel connecties er van en naar mogelijk zijn.

Author:
Frederik Smolders

Field Detail

vPropertyValues

`public java.util.Vector<Property> vPropertyValues`

Bijhouden van extra properties

Constructor Detail

TNode

```
protected TNode(java.lang.String label,
                java.lang.String paletteCategory)
```

Method Detail

getCategory

`public java.lang.String getCategory()`

---

setCategory

`public void setCategory(java.lang.String category)`

---

getLabel

`public java.lang.String getLabel()`

---

setLabel

`public void setLabel(java.lang.String label)`

---

setAmountConToPermitted

`public void setAmountConToPermitted(int amountConToPermitted)`

Stelt in hoeveel edges er in deze node mogen aankomen.

Parameters:

`amountConToPermitted` - hoeveel edges er in deze node mogen aankomen.

---

setAmountConFromPermitted

`public void setAmountConFromPermitted(int amountConFromPermitted)`

Stelt in hoeveel edges er in deze node mogen aankomen.

Parameters:

`amountConToPermitted` - hoeveel edges er in deze node mogen aankomen.

---

addNodeProperty

```
public void addNodeProperty(java.lang.String propertyName,
                            java.lang.String propertyValue)
```

Voegt een property (string) aan de node toe. Deze properties worden in het property window getoond als de node geselecteerd wordt.

Parameters:

`propertyName` - de naam van de nieuwe property.

`propertyValue` - de default waarde van de property.

---

addNodeProperty

```
public void addNodeProperty(java.lang.String propertyName,
                            int propertyValue)
```

Voegt een property (int) aan de node toe. Deze properties worden in het property window getoond als de node geselecteerd wordt.

Parameters:

`propertyName` - de naam van de nieuwe property.

`propertyValue` - de default waarde van de property.

---

addNodeProperty

```
public void addNodeProperty(java.lang.String propertyName,
                            java.lang.String propertyValue,
                            boolean readonly)
```

Voegt een property (string) aan de node toe. Deze properties worden in het property window getoond als de node geselecteerd wordt.

Parameters:

`propertyName` - de naam van de nieuwe property.

`propertyValue` - de default waarde van de property.

`readonly` - duidt aan of de property read only is of niet.

---

addNodeProperty

```
public void addNodeProperty(java.lang.String propertyName,
                            int propertyValue,
                            boolean readonly)
```

Voegt een property (int) aan de node toe. Deze properties worden in het property window getoond als de node geselecteerd wordt.

Parameters:

`propertyName` - de naam van de nieuwe property.

`propertyValue` - de default waarde van de property.

`readonly` - duidt aan of de property read only is of niet.

---

setPropertyValue

```
public void setPropertyValue(java.lang.String propertyName,
                             java.lang.String propertyValue)
```

Stelt de waarde van een bepaalde property (string) in.

Parameters:

`propertyName` - de naam van de property waarvan de waarde ingesteld moet worden.

`propertyValue` - de string waarde die de aangeduide property krijgt.

---

setPropertyValue

```
public void setPropertyValue(java.lang.String propertyName,
                             int propertyValue)
```

Stelt de waarde van een bepaalde property (int) in.

Parameters:

`propertyName` - de naam van de property waarvan de waarde ingesteld moet worden.

`propertyValue` - de int waarde die de aangeduide property krijgt.

2.5 Class TTableLayout

```
java.lang.Object
  └org.netbeans.tlibrary.TTableLayout
```

All Implemented Interfaces:

org.netbeans.api.visual.layout.Layout

```
public class TTableLayout
extends java.lang.Object
implements org.netbeans.api.visual.layout.Layout
```

TTableLayout is een layout die de kinderen in een TLabelwidget langst elkaar plaatst volgens de in tabs meegegeven waarden.

Author:

Frederik Smolders

Constructor Detail

TTableLayout

```
public TTableLayout(java.util.Vector<java.lang.Integer> tabs,
                    int minimumHeight)
```

Method Detail

layout

```
public void layout(org.netbeans.api.visual.widget.Widget widget)
```

Specified by:

`layout` in interface `org.netbeans.api.visual.layout.Layout`

---

requiresJustification

```
public boolean
requiresJustification(org.netbeans.api.visual.widget.Widget widget)
```

Specified by:

`requiresJustification` in interface `org.netbeans.api.visual.layout.Layout`

---

justify

`public void` **`justify`**`(org.netbeans.api.visual.widget.Widget widget)`

Specified by:

`justify` in interface `org.netbeans.api.visual.layout.Layout`

---

3 Package org.netbeans.tlibrary.palette

Class Summary	
PaletteItem	intern gebruik
PaletteManager	Met behulp van de PaletteManager kan de in Tlibrary geÃ¯ntegreerde palette gevuld worden.
PaletteSupport	Klasse om palette aan te maken (palette moet in de lookup van het betreffende topcomponent gezet worden).

3.1 Class PaletteManager

```
java.lang.Object
  └org.netbeans.tlibrary.palette.PaletteManager
```

```
public class PaletteManager
extends java.lang.Object
```

Met behulp van de PaletteManager kan de in Tlibrary geïntegreerde palette gevuld worden.

Author:
Frederik Smolders

Constructor Detail

PaletteManager

```
public PaletteManager()
```

Method Detail

addPaletteChild

```
public void addPaletteChild(java.lang.String paletteCategory,
                            java.lang.String widgetCategory,
                            java.lang.String title,
                            java.lang.String imageThumb,
                            java.lang.String image)
```

Voegt een nieuwe entry in bij de palette.

Parameters:

`paletteCategory` - onder welke categorie naam in de palette?

`widgetCategory` - is het item een 'image' of een 'widget'? (later nodig om bv hele widgets te slepen in een TLabelWidget). voor scene acceptables maakt deze categorie niet uit!

`title` - de naam van het item in de palette. Dit is de naam die doorgegeven wordt aan de TGraphScene en de TLabelWidgets om het item te identificeren. Op basis hiervan gaan we dus beslissen wat we aan de TGraphScene of de TLabelWidget gaan toevoegen.
`imageThumb` - het icoontje dat in de palette komt te staan naast de title van het item.
`image` - als de widgetCategory 'image' is, moet hier de gewenste image locatie meegegeven worden. Van deze image wordt dan automatisch een widget versie van gemaakt.

---

addPaletteChild

```
public void addPaletteChild(java.lang.String paletteCategory,
                            java.lang.String widgetCategory,
                            java.lang.String title,
                            java.lang.String imageThumb)
```

Voegt een nieuwe entry in bij de palette.

Parameters:

`paletteCategory` - onder welke categorie naam in de palette?
`widgetCategory` - is het item een 'image' of een 'widget'? (later nodig om bv hele widgets te slepen in een TLabelWidget). voor scene acceptables maakt deze categorie niet uit!
`title` - de naam van het item in de palette. Dit is de naam die doorgegeven wordt aan de TGraphScene en de TLabelWidgets om het item te identificeren. Op basis hiervan gaan we dus beslissen wat we aan de TGraphScene of de TLabelWidget gaan toevoegen.
`imageThumb` - het icoontje dat in de palette komt te staan naast de title van het item.

---

addPaletteCategory

```
public void addPaletteCategory(java.lang.String paletteCategory)
```

Voegt een nieuwe categorie toe aan de palette. Dit moet gebeuren voor we de palette gaan vullen.

Parameters:

`paletteCategory` - de naam van de nieuwe categorie die toegevoegd moet worden.

---

getPaletteCategories

```
public static java.util.Vector<java.lang.String> getPaletteCategories()
```

Geeft paletteCategories van de palette terug. Wordt gebruikt om de palette automatisch aan te maken.

Returns:

vPaletteCategories de vector die alle paletteCategories bijhoudt.

---

getPaletteChildren

```
public static java.util.Vector<PaletteItem> getPaletteChildren()
```

Geeft alle items van de palette terug. Wordt gebruikt om de palette automatisch aan te maken.

Returns:

vPaletteCategories de vector die alle paletteChildren bijhoudt.

---

getItem

```
public static PaletteItem getItem(java.lang.String title)
```

Geeft het item met de gevraagde title terug. Wordt gebruikt om de palette automatisch aan te maken.

Returns:

het overeenkomstig PaletteItem dat bij de gevraagde title hoort.

3.2 Class PaletteSupport

```
java.lang.Object
  └org.netbeans.tlibrary.palette.PaletteSupport
```

```
public class PaletteSupport
extends java.lang.Object
```

Klasse om palette aan te maken (palette moet in de lookup van het betreffende topcomponent gezet worden).

Author:

Frederik Smolders

Constructor Detail

PaletteSupport

```
public PaletteSupport()
```

Method Detail

createPalette

```
public static org.netbeans.spi.palette.PaletteController createPalette()
```

Maakt de palette aan. Voorbeeld: associateLookup(Lookups.fixed(new Object[] { PaletteSupport.createPalette() }));

---

4 Package org.netbeans.tlibrary.properties

Class Summary	
explpropertiesAction	Action which shows explproperties component.
Property	Property voor intern gebruik in Tlibrary
PropertyExplorer	PropertyExplorer bevat een explorerManager die de properties van de geselecteerde node in de TGraphScene toont.
PropertyInt	Property (int) voor intern gebruik in Tlibrary
PropertyNode	Hier worden de properties van een node opgeslagen.
PropertyString	Property (string) voor intern gebruik in Tlibrary

4.1 Class PropertyExplorer

```
java.lang.Object
  └org.netbeans.tlibrary.properties.PropertyExplorer
```

```
public class PropertyExplorer
extends java.lang.Object
```

PropertyExplorer bevat een explorerManager die de properties van de geselecteerde node in de TGraphScene toont.

Author:
Frederik Smolders

Field Detail

explorerManager

```
public static org.openide.explorer.ExplorerManager explorerManager
```

Constructor Detail

PropertyExplorer

```
public PropertyExplorer()
```

Method Detail

updateProperties

```
public static void
updateProperties(org.openide.nodes.AbstractNode abstractNode,
                                    java.lang.String displayName)
```

Update explorerManager om de gegevens van de nieuw geselecteerde node te tonen. Deze functie wordt intern in TGraphScene gebruikt.

Parameters:

`abstractNode` -

`displayName` -

---

registerPropertyPanel

```
public static void
registerPropertyPanel(org.openide.windows.TopComponent topcomp)
```

Registreer een zelf gemaakte TopComponent dat de ExplorerManager.Provider implementeert. Door deze te registreren weet Tlibrary dat hier de properties getoond moeten worden.

Parameters:

`topcomp` - de TopComponent dat de ExplorerManager.Provider implementeert.
