Der SceneMaker2VOnDA-Compiler

Bericht für das Softwareprojekt ‚Robots Talking Social'
Sommersemester 2018

Simon Ahrendt, Max Depenbrock, Jana Jungbluth

21. September 2018

Inhaltsverzeichnis

1 Nutzen **1**

2 Der SceneMaker-Parser **1**

3 Der Automat **1**
3.1 Aufbau 1
3.2 Knoten, Variablen, Scopes 2
3.3 Kanten 2

4 Der Compiler **3**
4.1 Funktionsweise 3
4.2 Die Ontologie 3
4.3 Die .rudi-Dateien 4

5 Nutzungsanleitung **5**

6 Mögliche Erweiterungen **6**

1 Nutzen

Der SceneMaker2VOnDA-Compiler ermöglicht die Übersetzung eines automatenbasierten Dialogmanagements für Dialogsysteme in ein äquivalentes, regelbasiertes VOnDA-Projekt. Das automatenbasierte Dialogmanagement kann mit dem Programm Visual SceneMaker [1] erstellt werden.

Visual SceneMaker bietet dabei als graphisches Tool einen erleichterten Einstieg in das Design des Dialogmanagements.

Die Übersetzung mit dem SceneMaker2VOnDA-Compiler ermöglicht es, ein mit Visual SceneMaker erstelltes Projekt innerhalb des VOnDA-Frameworks zu erweitern, und erreicht so den Vorteil des leichteren Zugangs, den Visual SceneMaker bietet, für VOnDA.

Dieser Projektbericht soll eine kurze Übersicht über die Funktionsweise des SceneMaker2VOnDA-Compiler geben.

2 Der SceneMaker-Parser

SceneMaker speichert Projekte in verschiedenen XML-Dateien. Der für den Compiler relevante Graph wird in `sceneflow.xml` gespeichert. Die Struktur dieser Datei ist vergleichsweise tief und damit komplex. So werden zum Beispiel arithmetische Ausdrücke nicht als Strings, sondern als geschachtelte XML-Elemente dargestellt. Daher wird zum Parsen ein DOM-basierter Parser verwendet, der die im Speicher generierte Baumstruktur der XML-Datei in entsprechende Knoten- und Kanten-Objekte umwandelt und ein `ScenemakerAutomaton`-Objekt zurückgibt, das den kompletten Scenemaker-Graphen umfasst.

3 Der Automat

3.1 Aufbau

Der Dialogmanagement-Automat (`ScenemakerAutomaton`) besteht aus einer Menge an Superknoten und Knoten, welche durch Kanten miteinander verbunden sind. Es können mehrere Knoten gleichzeitig aktiv sein. Ist ein Knoten aktiv, werden die dort definierten Befehle ausgeführt und, falls möglich, eine (oder im Fall von gabelnden Kanten mehrere) der von diesem Knoten ausgehenden Kanten zum nächsten (Super)knoten gewählt.

Falls keine ausgehende Kante gewählt werden kann, wird der aktuelle Superknoten verlassen und eine ausgehende Kante des Eltern-Superknotens gewählt. Falls keine solche Kante existiert oder gewählt werden kann, wird rekursiv auf dessen Eltern-Superknoten zugegriffen, und so weiter. Falls auf keiner höheren Ebene eine ausgehende Kante gewählt werden kann, so „stirbt“ der Prozess in diesem Knoten und er wird einfach aus der Liste der aktiven Knoten entfernt. Um einen Knoten über längere Zeit aktiv zu halten, muss also auf eine Schleife an diesem Knoten zurückgegriffen werden.

[1] Gebhard, P., Mehlmann, G., and Kipp, M. (2012). Visual SceneMaker—a tool for authoring interactive virtual characters. Journal on Multimodal User Interfaces, 6(1-2):3–11.

3.2 Knoten, Variablen, Scopes

Der Automat besteht aus einer Menge an Knoten, an denen Code definiert werden kann, welcher ausgeführt wird, wenn der Knoten aktiv wird.

Jeder Knoten besitzt eine Menge ausgehender sowie eingehender Kanten und einen Elternknoten.

Superknoten sind eine Unterklasse von Knoten. Dabei handelt es sich um Subautomaten, sie können also weitere Knoten als Kinder haben. Einige dieser Kinderknoten sind als Startknoten ausgezeichnet. Wird ein Superknoten aktiv, werden dadurch alle seine Startknoten aktiv (bei mehreren Startknoten führt das zu mehreren parallelen Prozessen innerhalb eines Superknotens).

Der Automat selbst (`SceneMakerAutomaton`) ist als Unterklasse von Superknoten definiert. Er unterscheidet sich von Superknoten lediglich dadurch, dass er keinen Elternknoten besitzt.

Jeder Superknoten agiert als ein eigener Sichtbarkeitsbereich für Variablen. Diese werden entweder am Automaten oder an einem der Superknoten definiert. Ein Knoten kann damit nur auf jene Variablen zugreifen, welche in einem seiner übergeordneten Superknoten definiert sind. Der Automat selbst wird dabei als der alle andere Knoten dominierende Superknoten verstanden.

Die möglichen Typen für eine Variable sind Boolean, Integer, Float oder String.

3.3 Kanten

Der Automat erlaubt folgende Kantenarten zwischen Knoten:

Epsilon-Kanten (Epsilon Edges) ermöglichen den Zustandswechsel von einem Knoten zu einem anderen, ohne dass irgendeine Bedingung erfüllt sein muss. Ein Knoten darf höchstens eine ausgehende Epsilon-Kante und haben. Epsilon-Kanten können nur mit Konditional-Kanten oder Unterbrechenden Kanten kombiniert werden.

Timeout-Kanten (Timeout Edges) ermöglichen den Zustandswechsel nach Ablauf eines festgelegten Zeitintervalls (Angabe in Millisekunden). Sobald ein Knoten aktiv wird, der eine ausgehende Timeout-Kante hat, wird die Zeit heruntergezählt. Besitzt der Knoten keine weitere mögliche ausgehende Kante, bleibt der Knoten aktiv, bis das Zeitintervall abgelaufen ist. Nach Ablauf des Timers wird der Zustand gewechselt. Ein Knoten darf höchstens eine ausgehende Timeout-Kante haben. Timeout-Kanten können nur mit Konditional-Kanten oder Unterbrechenden Kanten kombiniert werden.

Wahrscheinlichkeitskanten (Probability Edges) ermöglichen den zufälligen Zustandswechsel. Bei mehreren ausgehenden Wahrscheinlichkeitskanten wird zufällig *einer* dieser Übergänge ausgewählt. Wahrscheinlichkeitskanten können mit keiner anderen Kantenart kombiniert werden.

Gabelnde Kanten (Fork Edges) ermöglichen den Zustandswechsel ohne Bedingung in mehrere Zustände gleichzeitig. Wird dieser Knoten verlassen, werden alle Zielknoten der ausgehenden Gabelnden Kanten gleichzeitig aktiv (der Prozess „splittet sich auf“). Gabelnde Kanten können mit keiner anderen

Kantenart kombiniert werden.

Konditional-Kanten (Conditional Edges) haben eine Bedingung. Der Zustand wird gewechselt, falls diese (nach dem Ausführen des Codes an diesem Knoten) zu `true` auswertet. Ein Knoten kann mehrere Konditional-Kanten besitzen, und diese können mit Epsilon-, Timeout- oder Unterbrechenden Kanten kombiniert werden. Falls die Bedingung einer Konditional-Kante erfüllt ist, hat diese eine höhere Priorität als Epsilon- oder Timeout-Kanten, aber eine geringere als Unterbrechende Kanten.

Unterbrechende Kanten (Interruptive Edges) haben eine Bedingung. Der Zustand wird gewechselt, wenn diese zu `true` auswertet. Anders als bei allen anderen Kantenarten kann dieser Wechsel allerdings nicht nur erst nach der Abarbeitung des Codes an dem Knoten erfolgen, sondern auch schon vorher, so dass der Code an diesem Knoten dann nicht ausgeführt wird. Geht die Unterbrechende Kante von einem Superknoten aus, werden alle darin ablaufenden Prozesse abgebrochen, sobald die Bedingung der Unterbrechenden Kante erfüllt ist. Unterbrechende Kanten können mit Epsilon-, Timeout- oder Konditional-Kanten kombiniert werden. Dabei haben sie Priorität vor allen anderen Kantenarten.

4 Der Compiler

4.1 Funktionsweise

Der SceneMaker2VOnDA-Compiler erhält einen Dialogmanagement-Automaten und erstellt daraus neben einer Ontologie in Form einer `.nt`-Datei eine Menge an `.rudi`-Dateien, welche die Funktionsweise des Automaten durch Regeln abbilden. Diese Dateien können dann in ein VOnDA Projekt eingebunden werden (für Näheres siehe Abschnitt 5 und insbesondere `kurzanleitung.pdf`).

4.2 Die Ontologie

Der SceneMaker2VOnDA-Compiler erstellt eine n-Tupel-Datei `ontology.nt`, welche die Ontologie darstellt, die für die Imitation der Funktionsweise des gegebenen Automaten benötigt wird.

In dieser Datei wird die RDF Klasse `Supernode` mit ihren Properties (siehe Tabelle 1) sowie für jeden Superknoten des Automaten eine eigene Unterklasse von `Supernode` definiert. Für jede an einem Superknoten definierte Variable erhält die RDF-Klasse dieses Knotens eine Property mit dem Namen der Variable. Der Typ dieser Property ist dabei der Typ der Variable.

Zuletzt wird für jeden Superknoten eine Instanz der entsprechenden Klasse erstellt. Grundsätzlich gilt, dass auch zur Laufzeit immer nur eine Instanz jedes Superknoten existiert, auf die unter Umständen mehrere Prozesse im gleichen Superknoten zugreifen können. Wenn kein Prozess im Superknoten aktiv ist, wird `active` auf `false` gesetzt und die Auswertung der zu diesem Superknoten gehörenden VOnDA-Regeln übersprungen.

Property	Funktion
`parent`	Referenz auf den Elternknoten des Superknotens
`super_children`	Liste der aktiven Kinder-Superknoten
`simple_children`	List der aktiven Kinderknoten, die keine Superknoten sind
`active`	Bool'scher Wert: Superknoten ist aktiv
`initiated`	Liste der im nächsten Schritt zu aktivierenden Kinderknoten (Superknoten)
`imminent_children`	Liste der aktiven Kinderknoten, deren Code noch nicht ausgeführt wurde (keine Superknoten)
`VAR_NAME`	Wert der Variable VAR_NAME (für jede am Supeknoten definierte Variable)

Tabelle 1: Properties der RDF-Klasse `Supernode`

4.3 Die .rudi-Dateien

Der Compiler erstellt zunächst die Datei `ChatAgent.rudi`, um Instanzen der in der Ontologie definierten Superknotten-Objekte als variablen zu deklarieren, sowie `MainAgent.rudi`, welche die Funktionsweise des Automaten im Regelformat imitiert und allgemeine Funktionsdefinitionen enthält. Außerdem wird für jeden Superknoten des Automaten eine Datei generiert, die die Funktionsweise des jeweiligen Superknotens abbildet.

Die Verwaltung der aktuell aktiven Knoten wird über die Properties der Superknoten realisiert (siehe Tabelle 1). Außerdem vermerken diese, ob ein Superknoten initialisiert werden muss (falls er im nächsten Schritt betreten werden soll) und ob der Code an einem Knoten bereits ausgeführt wurde. Abhängig davon werden die entsprechenden Regeln der Knoten ausgeführt.

Superknoten werden wie folgt realisiert: Nach einer Regel `setup`, die den Knoten falls gewünscht initialisiert, sorgt eine Regel für das Überspringen der nachfolgenden Regeln, falls der Superknoten nicht aktiv ist. Dann folgt der Code aller ausgehenden Unterbrechenden Kanten dieses Superknotens. Ist der Knoten aktiv und die Bedingung einer Unterbrechenden Kante trifft zu, werden alle laufenden Prozesse innerhalb des Superknotens abgebrochen und der Zielknoten der Unterbrechenden Kante aktiviert.

Ein weitere Regel steuert das Verhalten beim Betreten (`_in`) und eine jenes beim Verlassen des Superknotens (`_out`). Der Code, der in SceneMaker unmittelbar unter einem Superknoten steht, wird beim Betreten des Superknotens ausgeführt.

Pro Kinderknoten, der kein Superknoten ist, enthalten die Superknoten-Dateien eine Regel, die das Verhalten dieses Knotens imitiert. Zuletzt werden die `.rudi`-Dateien, welche die Kind-Superknoten dieses Superknotens imitieren, importiert.

Pro (einfachem) Knoten wird eine Regel nach folgendem Schema generiert: Die Regel wird ausgeführt, wenn der Knoten aktiv ist. Falls die Bedingung einer ausgehenden Unterbrechenden Kante zutrifft, wird der Zustand gewechselt und der Knoten nicht weiter verarbeitet. Ansonsten wird der an diesem Knoten definierte Code ausgeführt, wenn dies nicht bereits in einem vorherigen Durchlauf geschehen ist. Danach folgt der Code, der die weiteren ausgehenden Kanten des Knotens imitiert, in der Abfolge ihrer Priorität.

Timeout-Kanten werden mit von VOnDA bereitgestellten Timeout-Blocks realisiert.

Bei Wahrscheinlichkeitskanten werden die möglichen Kantenübergänge in

eigenen Propose-Blöcken an den Action Selection Mechanismus übergeben, welcher einen der vorgeschlagenen Zustandswechsel auswählt. Damit haben die an den Wahrscheinlichkeitskanten definierten Wahrscheinlichkeiten keinen Einfluss auf die Auswahl des Kantenübergangs, welcher stattdessen ausschließlich von den Kriterien des Action Selection Mechanismus abhängig ist.

Im Falle eines erfolgreichen Kantenübergangs wird der aktuelle Knoten aus der Liste der aktiven Kinderknoten seines Elternknotens gelöscht und der Zielknoten der Kante wird der Liste der aktiven Kinderknoten seines Elternknotens hinzugefügt. Bei Gabelnden Kanten werden alle Zielknoten hinzugefügt.

Konditional-Kanten werden wie Unterbrechende Kanten umgesetzt. Sie unterscheiden sich nur dadurch, dass ihre Bedingung erst nach der Ausführung des am Knoten definierten Codes geprüft wird und erst zu diesem Zeitpunkt der Übergang erfolgt.

An den Knoten definierter Code wird in der VOnDA-Datei an der entsprechenden Stelle unverändert eingefügt, falls der Compiler mit der Option `-n` aufgerufen wird. Ansonsten ersetzt der Compiler als einzige Anpassung alle Vorkommen von `emitDA(` und `lastDA()>=` durch `emitDA(#` bzw. `lastDA() >= #`, da das Zeichen ‚`#`‘ von SceneMaker nicht verarbeitet werden kann.

Die Variablen werden statisch aufgelöst. Ihr Wert wird dabei in der gleichnamigen RDF-Property des Superknotens, an dem die Variable definiert ist, gespeichert. Ein Variablenaufruf im Code wird daher dementsprechend durch den Aufruf der entsprechenden Property des richtigen Superknotens ersetzt.

Ist kein Knoten mehr aktiv, wird VOnDAs `shutdown()` aufgerufen. Ein darauffolgender Input durch den Nutzer führt zu undefiniertem Verhalten. Dieser Aufruf von `shutdown()` könnte durch den Dialogdesigner aber z.B. durch einen Default-Knoten mit Selbstschleife im Automaten verhindert werden.

5 Nutzungsanleitung

Grundsätzlich wird der Compiler durch das Shell-Skript `start.sh` aufgerufen. Dieses Skript nimmt zwei Argumente:

- Den Pfad zu einer von *Visual SceneMaker* erstellten *Sceneflow-Datei*, die den zu übersetzenden Dialog definiert
- Den Pfad zu einem Output-Ordner, in dem der erstellte VOnDA-Code und die Ontologie-Datei abgelegt werden

Zusätzlich gibt es zwei Flags:

- `-b` gibt an, dass der angegebene Output-Ordner bereits ein angefangenes VOnDA-Projekt ist, das nach dem Hinzufügen des VOnDA-Codes und der Ontologie-Datei an die richtigen Stellen kompiliert und gebaut werden kann. In diesem Fall versucht der SceneMaker2VOnDA-Compiler das Projekt bereits zu bauen, um ein schnelleres Testen des Dialoges zu ermöglichen.
- `-n` unterdrückt ein Post-Processing des erstellten VOnDA-Codes, das standardmäßig ausgeführt wird, um Einschränkungen durch den *Visual SceneMaker* zu umgehen.

Die Datei `kurzanleitung.pdf` enthält weitere Informationen zur Verwendung des SceneMaker2VOnDA-Compilers und eine genauere Beschreibung der beiden Flags.

6 Mögliche Erweiterungen

Dank der abstrakten Automaten-Definition kann der SceneMaker2VOnDA-Compiler leicht von der Übersetzung von SceneMaker-Automaten auf andere automatenbasierte Dialogmanagement-Systeme erweitert werden. Dazu wird lediglich ein Parser benötigt, der aus dem Ausgabeformat eines beliebigen automatenbasierten Dialogmanagement-Systems ein `ScenemakerAutomaton`-Objekt erstellt.

Die Auswahl von Wahrscheinlichkeitskanten erfolgt nach der Übersetzung durch den Compiler unabhängig von ihren angegebenen Wahrscheinlichkeiten. Der Compiler könnte derart erweitert werden, dass die jeweiligen Kanten mit ihrer entsprechenden Wahrscheinlichkeit ausgewählt werden, oder die Wahrscheinlichkeiten zumindest den *Action Selection Mechanism* bei der Wahl des nächsten Zustands beeinflussen.