ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Medieninformatik

Analyse und Bewertung des Laufzeitverhaltens einer Anwendung zur Modalitätsarbitrierung

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Rainer Hochdorfer
rainer.hochdorfer@uni-ulm.de

Gutachter:
Prof. Dr. Michael Weber

Betreuer:
Frank Honold

2013

1 Motivation

2 Problemstellung

2.1 Beschreibung

Im Teilprojekt B3 des Sonderforschungsbereichs Transregio 62 werden aktuell Regeln zur Bewertung der Ausgabegeräte verwendet. Dies läuft im Moment wie folgt ab:

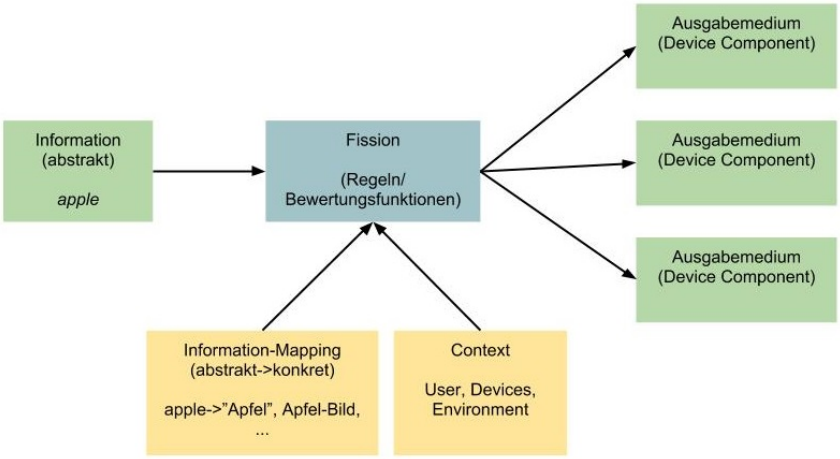

Abbildung 2.1: Vereinfachte Übersicht der Funktionsweise der Fission

Abstrakte Informationen werden vom Dialog-Manager an die Fission gesendet. Diese abstrakten Informationen werden dann von der Fission wie in Abbildung 2.1 zu sehen verarbeitet. Die Fission legt mittels eines Information-Mappings fest, wie die abstrakten Informationen als konkrete Informationen dargestellt werden können. So könnte zum Beispiel die abstrakte Information *apple* sowohl als Text als auch als Bild dargestellt werden. Diese konkreten Daten werden dann im Bezug auf die vorhandenen Kontextinformationen (wie User-, Environment- oder Device-Context bewertet. Die Bewertung basiert auf unterschiedlichen Bewertungsfunktionen. Eine Funktion repräsentiert dabei eine gestalterisch bewertete Aussage, z. B.: „Es ist sehr gut den akustischen Kanal einzusetzen wenn der Nutzer blind ist.“ Außerdem besitzt jede Regel einen Funktionswert, der positiv oder negativ sein kann. Alle Regeln werden dann auf alle Kombinationen aus konkreten Informationen und Ausgabemedien(Devices) angewendet. Dabei kann jede Regel mit einer unterschiedlichen Gewichtung in die Bewertung einfließen. Die Gewichtung geht aus dem Kontext hervor. Dieser Ansatz der Bewertung hat vermutlich exponentielle Laufzeitkomplexität. Davon ausge-

hend, dass bereits ein weitestgehend optimaler Algorithmus genutzt wird um die Bewertung der Regeln zu berechnen soll dieser Ansatz nun evaluiert werden. Dies dient dazu Aussagen über den Einfluss der Variablen auf die Laufzeit treffen zu können.

3 Theoretische Analyse

Wie in der Problemstellung beschrieben, soll aus einem gewissen Set an Informationen zur Verfügung. Aus diesen Informationen soll mit Hilfe eines Algorithmus das optimale Ausgabemedium gewählt werden. Da die eingegebenen Information zum Zeitpunkt der Berechnung feststehen und der Algorithmus auf ein festes Regelwerk zur Berechnung zurückgreift ist dieser deterministisch. Er liefert bei selber Eingabe eine eindeutige Lösung. Es handelt sich dabei um einen Algorithmus zur Lösung eines Optimierungsproblems.

Zur Lösung von Optimierungsproblemen gibt es verschiedene Kategorien zur Lösung, abhängig vom gewünschten Ergebnis und den zu verwendenden Datenstrukturen.

- Dynamisches Programmieren
- Greedy Algorithmen
- Optimierung von Baumstrukturen (z. B. Branch and Bound)

3.1 Beschreibung des aktuellen Ansatz (Algorithmus)

3.2 Theoretische Bewertung

3.3 Vergleich mit ähnlichen Lösungen

4 Evaluationsansätze vergleichbarer Systeme

In diesem Kapitel wird durch Literaturrecherche überprüft wie vergleichbare Systeme evaluiert wurden.

4.1 Systeme zur Modalitätsarbitrierung

Quelle der Spanier

Es wurden bereits einige Ansätze zur Evaluation von Systemen zur Modalitätsarbitrierung versucht. Dabei wird im allgemeinen versucht zu überprüfen wie adaptiv ein System ist. Das bedeutet es werden die Ergebnisse qualitativ überprüft. Keiner der Ansätze überprüfte dabei das Laufzeitverhalten des Algorithmus. Darum werden im folgenden Abschnitt Evaluationsansätze für vergleichbare Algorithmen betrachtet.

4.2 Evaluationsansätze vergleichbarer Algorithmen

Wie im vorigen Abschnitt beschrieben existieren bisher keine Evaluationsansätze für die Analyse des Laufzeitverhaltens eines Systems zur Modalitätsarbitrierung. Aus diesem Grund sollen in diesem Teil nun vergleichbare Algorithmen betrachtet werden.

5 Evaluation

Das Ziel der Evaluation ist die Performance des Fissions-Algorithmus zu messen, welcher die Bewertungsfunktionen für verschiedene Dialog-Outputs berechnet. Wie in der Theoretischen Betrachtung festgestellt sind dabei folgende unabhängigen Variablensets zu beachten: *Personmodel*, *Surroundingsmodel*, *DistanceRelations*, die *Device*- und *Componentmodel* Permutationen, *Rules* und der jeweilige *DialogOutput*. Zudem gilt es die Hardwarekonfiguration des Testsystems zu beachten. Abhängige Variable ist die Zeit, die der Algorithmus zur Berechnung benötigt.

5.1 Entwicklung des Tools zur Evaluation

Um das Evaluationsvorhaben durchzuführen wird ein zusätzliches Tool entwickelt. Es hat insbesondere drei Aufgaben: Erzeugen von generischen Testdaten (unabhängige Variablen), Versand der Daten als Semaine-Nachrichten an die Fission und das Logging der Testergebnisse (abhängige Variablen).

5.1.1 Generische Testdaten

Bei der generischen Erzeugung der Testdaten gelten spezielle Anforderungen. Mit der Klasse *Random*, die durch das .Net Framework zur Verfügung gestellt können Zufallszahlen erzeugt werden. Jedoch bietet diese Klasse keine Funktionen an um Wahrscheinlichkeitsverteilungen zu erzeugen. Für die Erzeugung dieser Zufallswerte wurde die statische Klasse *Randomizer* erstellt. Diese bietet verschiedene Methoden an um die generischen Testdaten zu erzeugen.

auf Zufallswerte eingehen

Zufällige, gleichverteilte Wahrscheinlichkeitswerte

Eine dieser Anforderungen war es, für beliebig viele Entitäten eine Wahrscheinlichkeitsverteilung zu erzeugen, die in Summe eine vorgegebene Gesamtwahrscheinlichkeit ergibt. Die Einzelwerte sollen hierbei jedoch zufällig gleichverteilt und *nicht* normalverteilt sein. Eine mathematische Lösung für dieses Problem bietet folgender Ansatz[1]:

[1] http://www.matheboard.de/archive/501805/thread.html, 14.11.2013

1. Erzeuge n verschiedene Zufallszahlen $z_i, i = 1...n$ mit $z_i \in \mathbb{R}^+$
2. Summiere über alle Zufallszahlen: $z := \sum_{i=1}^{n} z_i$
3. Teile die gewünschte Summe der Zufallszahlen s durch die Summe der Zufallszahlen z: $\frac{s}{z} = m$
4. Multipliziere alle Zufallszahlen $z_1...z_n$ mit dem Faktor m und erhalte die gewünschten Zufallszahlen $y_1...y_n$ mit $s = \sum_{i=1}^{n} y_i$ mit $y_i = m * z_i$

Diese Schritte sind im Algorithmus (Listing 5.1) umgesetzt. Der Methode müssen ein *Random*-Objekt, die gewünschte Summe der Zufallszahlen (bei Wahrscheinlichkeitsverteilung z. B. 1 (=100%)) und die Anzahl der Werte übergeben werden. Das *Random*-Objekt wird an die Methode übergeben, da bei mehrfachem Aufruf der Methode dasselbe Objekt verwendet werden sollte, um eine breite Streuung der Zufallszahlen zu erhalten.

Listing 5.1: Gleichverteilte Wahrscheinlichkeitsverteilung für beliebig viele Werte

```
1  public static List<Double> RandomDistribution(Random rnd, Double
       sum, Int32 count)
2  {
3      // Liste für die erstellten Werte (Rückgabewert)
4      List<Double> randomValues = new List<Double>();
5      // Summe der Zufallszahlen
6      Double sumZ = 0;
7
8      for (Int32 i = 0; i < count; i++)
9      {
10         Double randomValue = rnd.NextDouble();
11         // Erzeuge die gewünschte Anzahl Zufallswärte
12         randomValues.Add(randomValue);
13         // Berechne Summe z der erzeugten Zufallswerte
14         sumZ += randomValue;
15     }
16
17     // Sei sum = x. Teile x/z = m.
18     if (sumZ <= 0)
19     {
20         // Nicht durch 0 teilen, Wert sollte positiv sein!
21         return null;
22     }
23     Double multiply = sum / sumZ;
24
25     // Multipliziere alle Zufallszahlen mit m.
26     for (Int32 i = 0; i < count; i++)
27     {
28         randomValues[i] = randomValues[i] * multiply;
29     }
30
```

```
31      return randomValues;
32  }
```

Zufällige Intgerwerte mit gleichverteilten Wahrscheinlichkeitswerten

Eine weitere Anforderung ist die zufällige Generierung von beliebig, jedoch nur einmal auftretenden Integer-Werten innerhalb eines bestimmten Intervalls. Diese wiederum müssen mit einer Wahrscheinlichkeitsverteilung versehen sein. So soll z. B. auf einer Skala von eins bis zehn, n zufällige Werte gewählt und mit einer Wahrscheinlichkeit belegt werden. Die Summe der n Wahrscheinlichkeitswerte soll eins ergeben (etwa: 3=0,2; 4=0,1; 6=0,4; 9=0,3). Der in Listing 5.2 dargestellte Algorithmus erzeugt eine zufällige Anzahl beliebiger jedoch distinkter Integer-Werte in einem vorgegebenen Zahlenbereich [min, max]. Diesen Zahlen wiederum wird dann mit der im vorangegangenen Abschnitt beschriebenen Methode eine Wahrscheinlichkeitsverteilung zugeordnet. Diese Zuordnung wird in einem *Dictionary<int,double>* gespeichert und zurückgegeben.

Listing 5.2: Gleichverteilte Wahrscheinlichkeitsverteilung für beliebige Integerwerte

```
1  public static Dictionary<int, double> IntAndProbabilityDistribuion
       ( Random rnd,  int min, int max, double sum) {
2
3          // Rückgabewert
4      Dictionary<int, double> distribution = new Dictionary<int,
           double>();
5
6      List<int> values = new List<int>();
7      // Erzeuge eine bestimmt Anzahl an Werten
8      int countValues = Randomizer.GenerateRandomNumber(min, max);
9      for (int i = 0; i<countValues; i++)
10     {
11         int newValue;
12         do {
13             newValue = Randomizer.GenerateRandomNumber(min, max);
14         }
15         while (values.Contains(newValue));
16         // Füge einzigartige Zahl hinzu
17         values.Add(newValue);
18     }
19     // Erzeuge Wahrscheinlichkeiten
20     List<double> valueProbabilities = Randomizer.
           RandomDistribution(rnd, sum, countValues);
21     // Füge Werte in Model ein
22     for (int i = 0; i < countValues; i++)
23     {
24         distribution.Add(values[i], valueProbabilities[i]);
25     }
```

```
26
27      return distribution;
28  }
```

5.1.2 Versand der Nachrichten

XML Nachrichten werden mit Hilfe der Klassen erzeugt, welche generische Testdaten erzeugen. Diese werden dann mit *Semaine Sender Components* an die Middleware und somit an die Fission verschickt. An dieser Stelle ist dies Beispielhaft für das PersonModel beschrieben:

Listing 5.3: Semaine Copmponent PersonModelSender

```
1  class PersonModelSender: Component
2  {
3      // Die Sender und Receiver
4      private XMLSender _personModelSender;
5      private Receiver _requestReceiver;
6
7      // Konstruktor
8      public PersonModelSender()
9          : base("B3.Budgie.PersonModelSender")
10     {
11         //Person Model request Receiver
12         _requestReceiver = new Receiver("semaine.data.user.request
               ");
13         _receivers.AddLast(_requestReceiver);
14         // Person Model Sender
15         _personModelSender = new XMLSender("semaine.data.user", "
               XML", this.GetType().Name);
16         _senders.AddLast(_personModelSender);
17     }
18
19     // Reagiere auf Requests
20     protected override void React(SEMAINEMessage message)
21     {
22         if (message.Text.Equals("UPDATE", StringComparison.
               OrdinalIgnoreCase)) {
23             GenericPersonModel person = new GenericPersonModel("
                   person_id");
24             XmlDocument doc = person.Person.XmlRepresentation;
25             _personModelSender.SendXML(doc, _meta.CurrentTime);
26         }
27     }
28 }
```

5.2 Durchführung

6 Auswertung und Diskussion

6.1 Zusammenfassung

6.2 Diskussion

6.3 Evtl. Verbesserungsansätze