Konzept und Realisierung eines Zustandsmaschinen-Editors für Interaktionen medizinischer Bildverarbeitung mit Debug-Funktionalität

Daniel Stein, Marcus Vetter, Ivo Wolf, Hans-Peter Meinzer

Abteilung für Medizinische und Biologische Informatik,
Deutsches Krebsforschungszentrum, D-69120 Heidelberg
d.stein@dkfz.de

Kurzfassung. In diesem Beitrag wird ein Open-Source-Plug-In für die Entwicklungsumgebung Eclipse vorgestellt, mit dem ein Programmablauf in Form von Zustandsmaschinen grafisch modelliert werden kann. Außerdem bietet das Konzept mit der Debug-Funktion die Möglichkeit, während der Laufzeit einer Applikation alle verwendeten Zustandsmaschinen und deren Zustandsfolgen zu beobachten. Dadurch lassen sich erstellte Zustandsmaschinen auf ihre korrekte Arbeitsweise hin überprüfen. Durch die Debug-Funktion wird das Aufspüren von Fehlern im Programmdesign deutlich vereinfacht, was gerade für die Qualitätskontrolle von medizinischen Anwendungen wichtig ist. Die erstellten Zustandsmaschinen werden beim Speichern in eine XML-Beschreibung überführt. Diese kann von anderen Applikationen direkt weiter verwendet werden.

1 Einleitung

Die Vielfalt an Möglichkeiten, die während einer Interaktion medizinischer Applikationen entstehen, kann durch eine Mealy-Zustandsmaschine [1] abgebildet werden. In der Theorie sind alle Programme Zustandsmaschinen, unabhängig davon, ob sie speziell als Zustandsmaschine programmiert sind oder über Variablen, Bedingungen oder Schleifen direkt im Programm-Code implementiert sind [2]. Zum Beispiel kann ein Regionenwachstums-Algorithmus nur angewendet werden, wenn zuvor die Bedingung „Saatpunkt in einem Bild gesetzt“ erfüllt ist, was einem Zustand „Saatpunkt gesetzt“ entsprechen könnte. In diesen Zustand würde man über das Ereignis „neuer Saatpunkt“ kommen und aus diesem Zustand gäbe es dann einen Übergang mit dem Ereignis „Regionenwachstum starten“ in den Folgezustand. Deswegen geht der Trend dahin, den Programmablauf direkt in Form von Zustandsmaschinen zu entwerfen. So ist zum Beispiel IGSTK (Image-guided Surgical Toolkit) gerade dabei, ein Architecture Validation Toolset zu entwickeln, bei dem jeder Programm-Code zunächst als Zustandsmaschine dargestellt wird. Dies soll die Patientensicherheit bei der Verwendung medizinischer Software durch einen zuverlässigen und stabilen Programmablauf

erhöhen [2, 3]. Jedoch ist das Erstellen von Zustandsmaschinen im Programm-Code sehr umständlich und bei einer wachsenden Anzahl an Zuständen auch sehr unübersichtlich [4].

Bei umfangreichen Zustandsmaschinen mit vielen Zuständen und Übergängen ist es mühsam und zeitaufwendig, Änderungen vorzunehmen. Zum Neuerstellen einer Zustandsmaschine muss der Ersteller den Ablauf dieser bereits im Kopf vorgeplant haben, um den Ablauf im Programm-Code implementieren zu können. In den meisten Fällen wird eine solche Zustandsmaschine noch auf dem Reißbrett entworfen, da auf Grund der Komplexität der Zustandsmaschine ein Diagramm zur Verdeutlichung des Ablaufs nötig ist. Mit Hilfe einer grafischen Oberfläche können komplexe Zustandsmaschinen schnell konstruiert, geändert und als Programm-Code gespeichert werden. Dies ermöglicht neuen Mitarbeitern ein schnelleres Einarbeiten in den Ablauf einer Zustandsmaschine.

Eventuell auftretende Entwurfsfehler, die erst bei Ablauf einer Anwendung unter Verwendung der Zustandsmaschine auftreten, sind im Programm-Code sehr schwer zu lokalisieren, da der Ablauf einer komplexen Zustandsmaschine anhand des Programm-Codes nur mühsam nachzuvollziehen ist. So entsteht für den Benutzer ein hoher Arbeitsaufwand, der mit Hilfe eines geeigneten Tools zum Debuggen von Zustandsmaschinen minimiert werden kann.

2 Methoden

Der entwickelte Zustandsmaschinen-Editor wurde in Java als Plug-In für die Entwicklungsumgebung Eclipse erstellt. Der Editor basiert auf dem bereits bestehenden Plug-In Graphical Editing Framework (GEF). Das GEF bietet eine Klassenbibliothek als Grundgerüst eines grafischen Editors.

Mit dem Zustandsmaschinen-Editor ist ein Programm entstanden, das sich in zwei Programmteile gliedert.

Der erste Teil ist ein Editor, mit dem Zustandsmaschinen grafisch modelliert werden können. Es können sowohl bestehende Zustandsmaschinen aus einer XML-Datei geöffnet, als auch eine neue XML-Datei angelegt werden. Der Programmablauf vom Öffnen einer XML-Datei bis zur Visualisierung einer Zustandsmaschine im Editor ist als Objektdiagramm in Abbildung 1 dargestellt. Zunächst wird mit Hilfe von JDOM die XML-Datei ausgelesen und die darin vorhandenen Zustandsmaschinen an die StateMachinesList übergeben. Aus dieser Liste können mehrere Editorfenster gleichzeitig geöffnet werden (Abb. 2). Dabei beinhaltet jeder Editor eine Zustandsmaschine, die aus mehreren verschiedenen Zuständen besteht. Jeder Zustand kann mehrere Übergänge mit jeweils einem Event besitzen, die jeweils mehrere Aktionen bewirken können. Im Editor stehen alle zur Bearbeitung von Zustandsmaschinen benötigten Funktionen zur Verfügung (Abb. 2). Zusätzlich können neue Zustandsmaschinen erstellt, nicht mehr benötigte gelöscht, sowie bereits bestehende kopiert und unter anderem Namen gespeichert werden. Nach dem Speichern werden alle vorgenommenen Änderungen automatisch in XML-Code konvertiert und in die XML-Datei geschrieben.

Der zweite Teil des Programms wurde entwickelt, um eventuelle Modellierungsfehler festzustellen. Hierbei handelt es sich um einen Debugger, der parallel zum Betrieb einer Applikation ausgeführt wird. Dabei wird eine Client-Server-Verbindung zwischen dem Zustandsmaschinen-Editor und der Applikation aufgebaut. Der Zustandsmaschinen-Editor dient als Server, die Applikation als Client. Die Applikation sendet alle von ihr benutzten Zustandsmaschinen und alle bei ihr eingehenden Ereignisse an den Zustandsmaschinen-Editor. In diesem kann die gewünschte Zustandsmaschine geöffnet werden. Es werden der aktuelle Zustand und der aktuelle Übergang durch eine rote Hinterlegung markiert. Zusätzlich erhält der Benutzer die Möglichkeit, den Zustand der Zustandsmaschine zu einem früheren Ereignis zu beobachten. Dies ermöglicht dem Benutzer den gesamten Ablauf einer Zustandsmaschine zu einer aufgetretenen Ereignisfolge noch einmal nachzuvollziehen.

3 Ergebnisse

Mit dem Zustandsmaschinen-Editor wurde ein Open-Source-Programm erstellt, das sich auf einfache Art und Weise in die ebenso frei verfügbare Entwicklungsumgebung Eclipse einbinden lässt. Mit Hilfe der grafischen Oberfläche wird das Erstellen, Bearbeiten, Refactoring und Reengineering von Zustandsmaschinen nicht nur beschleunigt, sondern auch deutlich übersichtlicher und einfacher (Abb. 2).

Durch die integrierte Debug-Funktion, die alle benutzten Zustandsmaschinen und Ereignisse einer Applikation registriert, steht ein nützliches Tool zur Aufspürung von Designfehlern innerhalb einer Zustandsmaschine zur Verfügung.

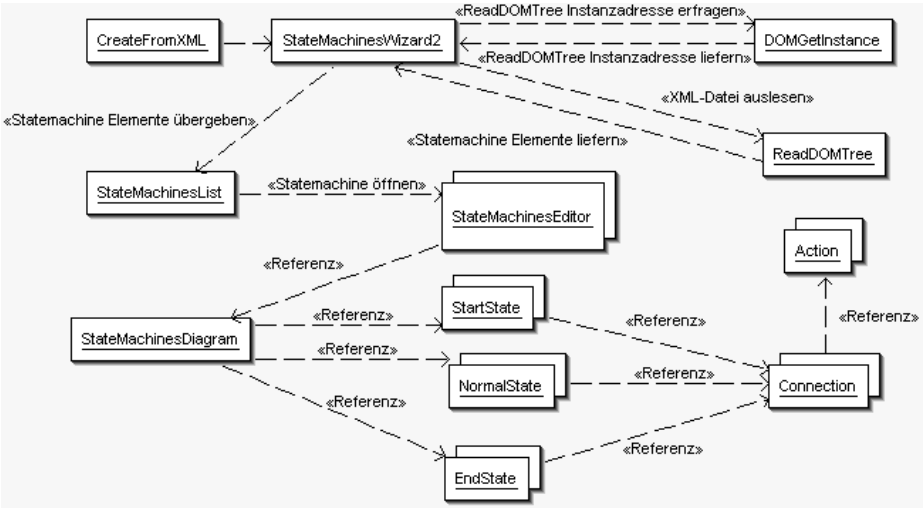

Abb. 1. Objektdiagramm: Ablauf des Zustandsmaschinen-Editor-Plug-Ins vom Öffnen einer XML-Datei bis zur Visualisierung einer Zustandsmaschine im Editor

Im Medical Imaging and Interaction Toolkit (MITK) [5] ist der Editor, sowie die Debug-Funktion bereits erfolgreich im Einsatz um neue Zustandsmaschinen zu definieren und komplexe Abläufe zu überprüfen und nachzuvollziehen. Es zeigen sich bereits die erwarteten Vorteile beim Refactoring, sowie beim Reengineering von bestehenden Zustandsmaschinen.

4 Diskussion

Durch das hier vorgestellte Tool wird es möglich sein, Zustandsmaschinen einfach, schnell und übersichtlich zu entwerfen. Des Weiteren steht mit der Debug-Funktion ein Tool zur Verfügung, mit dem man den Ablauf einer Zustandsmaschine während der Laufzeit der entwickelten Applikation auf die korrekte Arbeitsweise hin überprüfen kann.

Der Zustandsmaschinen-Editor wurde in Java mit der Programmieroberfläche Eclipse programmiert, da mit Hilfe dieser Open-Source-Entwicklungsumgebung ein neues Plug-In mit wenig Aufwand installiert und sofort genutzt werden kann. Als Programmiersprache wurde Java gewählt, weil Eclipse selbst in Java programmiert ist und sich die definierten Ziele in Java am besten verwirklichen ließen. Hinzu kommt, dass für Eclipse mit dem Graphical Editing Frame-

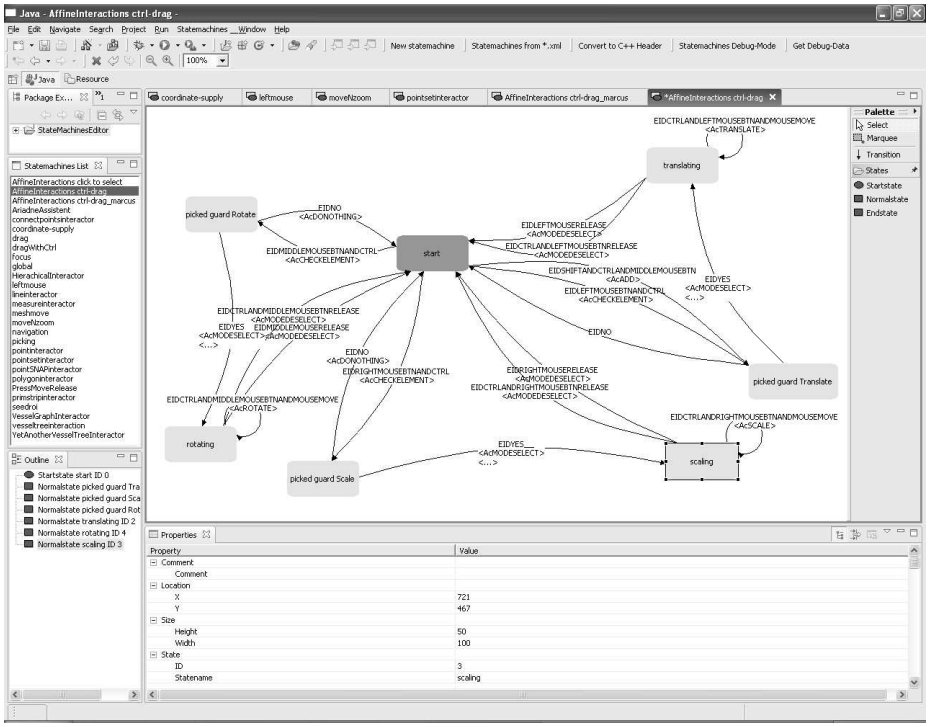

Abb. 2. Möglicher Aufbau der Benutzeroberfläche des Zustandsmaschinen-Editors

work (GEF) bereits ein in Java geschriebenes Plug-In zur Verfügung steht, das die Rahmenfunktionen zur Erstellung eines grafischen Editors bietet.

Das Konzept, die Zustandsmaschinen im XML-Format zu speichern, macht den Editor unabhängig von der Programmiersprache der Applikation, die diese Zustandsmaschinen nutzt. Die entstehende XML-Datei kann in jede Applikation, die Zustandsmaschinen im XML-Format verwendet, eingebunden werden.

Durch die Client-Server-Architektur zwischen dem Editor und der Applikation, die die Zustandsmaschinen verwendet, wird auch bei der Debug-Funktion die Unabhängigkeit zwischen Applikation und Editor gewahrt. In der Applikation muss nur ein passender Client implementiert werden, der sich mit dem als Server agierenden Editor verbindet. Dadurch, dass der Editor und die Applikation in verschiedenen Threads ablaufen und die Applikation nur die verwendeten Zustandsmaschinen und auftretenden Ereignisse an den Editor schickt, ändert sich die Performanz nicht wesentlich.

Der entstandene Zustandsmaschinen-Editor ist open-source und mit dem MITK zusammen frei verfügbar. Eine Installationsanleitung ist im Internet: http://www.mitk.org/documentation/doxygen/StatemachineEditor.html.

Literaturverzeichnis

1. Mealy GH. A method for synthesizing sequential circuits. Bell System Techn J. 1955;34(5):1045–79.
2. Gary K, Kokoori S, David B, et al. An architecture validation toolset for ensuring patient safety in an open source software toolkit for image-guided surgery applications. In: IJ - MICCAI Open Science Workshop. The Insight Journal; 2006.
3. Cheng P, Zhang H, Kim HS, et al. IGSTK: Framework and example application using an open source toolkit for image-guided surgery applications. Proc SPIE. 2006;6141:590–8.
4. Wegner I, Vetter M, Wolf I, et al. Ein generisches Interaktionskonzept mit Undo für die medizinische Bildverarbeitung. Proc BVM. 2004.
5. Wolf I, Vetter M, Wegner I, et al. The medical imaging interaction toolkit (MITK). Proc SPIE. 2004;5367:16–27.