

Tcl für Animator4, Teil I

Ein wesentliches Merkmal des Postprozessors *Animator4* ist seine Kommandosprache. Fasst man Kommandos in Session-Files zusammen, lassen sich Benutzereingaben hervorragend reproduzieren. Die Steuerung von *Animator4* mit Session-Files stößt jedoch an ihre Grenzen, sobald man eine Flusskontrolle wie zum Beispiel Schleifen benötigt.

Diese Funktionalität kann über die Tcl-Schnittstelle von *Animator4* abgebildet werden. Tcl ist eine leistungsfähige Skriptsprache, mit der man nicht nur Kontrollstrukturen wie Schleifen oder if-Anweisungen nutzen, sondern auch einfach grafische Benutzeroberflächen erstellen kann.

Tcl wird seit vielen Jahren aktiv entwickelt. Die Nutzung des speziell für Tcl entwickelten Tk-Pakets zur Erstellung graphischer Oberflächen ist sehr intuitiv. Zahlreiche Anwendungen verwenden Tcl zur skriptbasierten Steuerung; zudem hat Tcl den Vorteil, dass darin geschriebene Programme plattformunabhängig sind und somit sowohl unter Unix als auch unter Windows und MacOS verwendet werden können. Aufgrund der großen Verbreitung wird der Einstieg in die Skriptprogrammierung mit Tcl/Tk durch zahlreiche Nachschlagewerke und Online-Tutorials erleichtert ([1], [2], [3]).

Animator4 erweitert Tcl um Kommandos, die Informationen aus einer laufenden *Animator4*-Sitzung verarbeiten können. Diese *Animator4*-Kommandos beginnen stets mit GNS_....

In dieser sowie in der nächsten Ausgabe soll die Tcl-Schnittstelle von *Animator4* anhand einfacher Beispiele erläutert werden. Dabei wird in dieser Ausgabe auf folgende Grundlagen von Tcl/Tk eingegangen:

- Schleifen
- Parameterübergabe
- Graphische Oberflächen und Funktionen

Die nächste Ausgabe wird sich mit einem erweiterten Beispiel zur Parameterübergabe und dem Zugriff auf Modelldaten beschäftigen.

Die Beispiele können von der GNS-Homepage unter <http://gns-mbh.com/?id=GeniusTCL> geladen werden.

Tcl-Skripte können mit Hilfe des *Animator4*-Kommandos `rea tcl <skript>` gestartet werden. Legt man die Skripte im Benutzerverzeichnis unter `.a4dir/Scripts` ab, kann die Angabe des Pfades zu der Datei entfallen. Die in diesem Beitrag angegebenen Skripte nutzen die Funktionen von *Animator4* (Version 1.4.1), in älteren Versionen von *Animator4* sind einzelne Funktionen möglicherweise nicht nutzbar.

Schleifen

Beispiel 1 zeigt die Verwendung von Schleifen in Tcl-Skripten. Während die `for`-Schleife eine Standard-Anweisung in Tcl darstellt, ist `GNS_exe_command` eine von *Animator4* definierte Anweisung, mit der ein *Animator4*-Kommando ausgeführt werden kann. Das Skript durchläuft die `for`-Anweisung 36 mal; bei jedem Durchlauf wird das Kommando `vro 10` ausgeführt. Dies führt dazu, dass die Ansicht in 36 Schritten um jeweils 10° rotiert wird.

News

- ➔ **Animator4, Version 1.4.3 veröffentlicht**
- ➔ **GNS Systems integriert neue Compute Server für Volkswagen Mexiko**
- ➔ **GNS Systems integriert Windows HPC Server für Imperia**

Beiträge

- ➔ **Tcl für Animator4, Teil I**
- ➔ **Management von Lizenzen und Lizenzdiensten - Technische, organisatorische und finanzielle Aspekte**
- ➔ **Eine Frage der Gerechtigkeit - Bewertung von Lizenzkosten bei gemeinschaftlicher Lizenznutzung**
- ➔ **Verbundvorhaben FEMMINER**
- ➔ **Verwendung von Variablen in Animator4**



GNS Systems

```
01. # Rotate view in steps of 10 degree
02. for {set i 1} {$i <= 36} {incr i} {
03.   GNS_exe_command "vie yro 10"
04. }
```

Beispiel 1: Drehung der Ansicht in 36 Schritten (simple.tcl)

Das Skript kann mit jeder beliebigen *Animator4*-Sitzung getestet werden, es sollte jedoch darauf geachtet werden, dass ein Model-View aktiv ist.

Parameterübergabe

Viele Arbeitsabläufe unterscheiden sich nur in wenigen Punkten wie Namen, Pfaden, Grenzwerten oder der Anzahl der zu untersuchenden Knoten. Ein effizienter Ansatz zur Behandlung derartiger Aufgaben ist die Nutzung von Tcl-Skripten mit Parametern.

In Beispiel 2 ist eine einfache Variante der Steuerung über einen Übergabeparameter realisiert. Hier ist das vorherige Beispiel erweitert, so dass die Anzahl der Schritte für die vollständige Drehung der Ansicht um 360° variabel ist.

Die Anzahl der übergebenen Parameter wird in der Variable `argc` gespeichert. Die Argumente selbst können über das Tcl-Array `argv` und über die Tcl-Liste `argl` abgefragt werden.

```
01. # set steps: use 1st argument or 36 as number of steps
02. set steps [expr { $argc != 1 ? 36 : $argv(1)}]
03.
04. # print a message in the message window
05. GNS_exe_command "opt ech Rotating view in $steps steps..."
06.
07. # compute delta angle to be used for each rotation step
08. set delta [expr {360.0/$steps}]
09.
10. # loop over number of steps
11. for {set i 1} {$i <= $steps} {incr i} {
12. # execute A4 command, insert variables to do so
13. GNS_exe_command "vie yro $delta"
14. }
```

Beispiel 2: Drehung der Ansicht in beliebigen Schritten (rotate.tcl)

Im angegebenen Beispiel führt der Aufruf von `rea tcl rotate.tcl 100` dazu, dass die Ansicht in 100 Schritten à 3,6° um 360° gedreht wird. Es bietet sich an, optionale Parameter mit einem sinnvollen Standardwert zu versehen - dies ist in Beispiel 2 in Zeile 2 implementiert. Ist ein Parameter angegeben, wird dieser verwendet, andernfalls wird der Standardwert 36 benutzt.

In Zeile 8 wird aus der Anzahl der Schritte der Winkel δ berechnet, welcher zu einer vollständigen 360°-Rotation führt. In den Zeilen 11-14 wird analog zu Beispiel 1 das Kommando zum Rotieren der Ansicht entsprechend der angegebenen Anzahl der Schritte aufgerufen. Hierbei wird der zuvor berechnete Winkel δ verwendet.

Graphische Oberfläche

Das Skript in Beispiel 3 fragt den Benutzer in einer Dialogbox nach einer Variablen (z.B. dem Namen einer Datei) und speichert die Antwort in eine *Animator4*-Variable. Mit einem solchen Skript kann man zum Beispiel Session-Files interaktiv gestalten.

Bevor die Oberfläche gestaltet wird, werden in den Zeilen 1-11 zunächst zwei Voraussetzungen geprüft. Damit das Tcl-Skript flexibel wiederverwendet werden kann, erwartet es den Namen

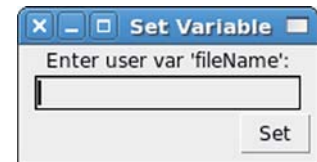


Abb. 1: Tcl/Tk-Dialog zur Eingabe einer Variablen

der zu setzenden Variablen als Kommandozeilenparameter. Ist dieser nicht angegeben, wird das Skript mit einer Fehlermeldung beendet (Zeile 2-5).

In den Zeilen 8-11 wird der aktuelle Wert der zu definierenden Variablen abgefragt, um diese in der Eingabezeile des Dialogs anzuzeigen. Ist diese noch nicht definiert, wird eine leere Zeichenkette als Ausgangswert verwendet und die Eingabezeile im Dialog bleibt leer.

Da Tcl an sich keine grafische Benutzeroberfläche erzeugen kann, wird in Zeile 14 das Paket Tk mit dem Befehl `package require Tk`; geladen.

Die Elemente des Dialoges werden in den Zeilen 17-38 definiert. Die Anordnung der Elemente erfolgt in den Zeilen 31-35, so dass sich der in Abbildung 1 dargestellte Aufbau ergibt. In Zeile 27 wird die Eingabetaste mit der Funktion `set_user_var` verknüpft, so dass beim Drücken der Eingabetaste diese Funktion aufgerufen wird. Der Dialog verfügt zudem über einen Set Button - auch dieser wird mit der Funktion verknüpft (Zeile 29).

Um den Arbeitsfluss zu erleichtern, wird der Fokus auf das Eingabefeld gesetzt, so dass der Benutzer sofort mit der Eingabe des Variablenwerts beginnen kann.

Das Skript arbeitet mit einer Funktion bzw. Prozedur, welche es ermöglicht, wiederkehrende Bestandteile des Skriptes zusammenzufassen. In den Zeilen 41-46 wird die Prozedur `set_user_var` definiert, welche im weiteren Verlauf verwendet wird, um die Variable mit Hilfe der *Animator4*-Tcl-Funktion `GNS_set_user_var` zu setzen und danach den Dialog zu schließen.

```
01. # check if the variable name is given
02. if {$argc != 1} {
03.     GNS_exe_command "opt ech Specify name of user variable!";
04.     return;
05. }
06.
07. # try to get current value of the variable
08. if { [catch {set value [GNS_get_user_var $argv(1)]} ] } {
09.     # set empty value if variable is not defined
10.     set value ""
11. }
12.
13. # load Tk package needed to create windows
14. package require Tk;
15.
16. # create window with title „Set Variable“
17. wm title . "Set Variable"
18. # create a frame
19. frame .frm -relief sunken
20. # create a label
21. label .vname -justify left -text [format "Enter user var ,%s':"
    $argv(1)];
22. # create an input field
23. entry .input;
24. .input insert end $value;
25. # event handling:
26. # set user variable when return key is pressed
27. bind .input <Return> {set_user_var $argv(1)}
28. # create a Set button
29. button .but -text "Set" -command {set_user_var $argv(1)}
30.
31. # arrange window elements
32. pack .vname -in .frm
33. pack .input -in .frm
34. pack .frm
35. pack .but -side right
36.
37. # set keyboard focus to the input field
38. focus .input
39.
40. # procedure to set the user variable and exit
41. proc set_user_var {var} {
42.     # set A4 user variable
43.     GNS_set_user_var $var [.input get]
44.     # close input dialog
45.     destroy .
46. }
```

Beispiel 3: Definition einer Variablen über einen Dialog (variable.tcl)

Zusammenfassung

In dieser Ausgabe wurde anhand einfacher Beispiele vorgestellt, wie die *Animator4*-Tcl-Schnittstelle zur Erleichterung der täglichen Arbeit mit dem *Animator4* eingesetzt werden kann. In der nächsten Ausgabe werden erweiterte Beispiele behandelt, welche unter anderem die Möglichkeiten zur Interaktion mit Modelldaten aufzeigen.

[1] https://www.bg.bib.de/portale/bes/pdf/Einfuehrung_Tcl.pdf (deutsches Tutorial)

[2] Das offizielle Tutorial: <http://www.tcl.tk/man/tcl/tutorial/tcltutorial.html> (englisches Tutorial)

[3] <http://www.invece.org/tclwise/index.html>

Björn Brodersen, GNS mbH,
Sascha Kremers, GNS mbH,
animator@gns-mbh.de



Management von Lizenzen und Lizenzdiensten -

Technische, organisatorische und finanzielle Aspekte

Software zur Durchführung und Automatisierung von Konstruktions- und Berechnungsprozessen ist integraler Bestandteil der modernen Produktentwicklung. Die Verfügbarkeit der notwendigen Lizenzen ist dabei für die Softwarenutzung eine wichtige Voraussetzung. Außerdem sind die Lizenzkosten meistens erheblich, so dass auch eine optimale Nutzung der Lizenzen unbedingt erforderlich ist.

Dieser Artikel betrachtet Möglichkeiten für die Optimierung der Verfügbarkeit und der Nutzung von Softwarelizenzen.

Obwohl sich der Artikel bei konkreten Beispielen auf den Lizenzdienst *FlexNet* bezieht, sind die Erläuterungen in den meisten Fällen auch für andere Lizenzdienste gültig.

Lizenzverfügbarkeit

Um die Lizenzverfügbarkeit sicherzustellen, müssen sowohl die Konfiguration als auch die Überwachung der Lizenzdienste betrachtet werden. Bei der Konfiguration geht es insbesondere um folgende Punkte:

- Gewährleistung der Ausfallsicherheit durch Redundanz
- Zuverlässige und einheitliche Methoden zum Starten und Stoppen der Lizenzdienste
- Prüfung des Inhaltes neuer Lizenzdateien im Vergleich zur Beschaffung
- Erprobung und Dokumentation der Methoden für die Bereitstellung von neuen Komponenten
 - Lizenzen
 - Allgemeine Lizenzdienste (*lmgrd*)
 - Anwendungsspezifische Lizenzdienste (Vendor Daemons)
- Systemtechnische Protokollierung der Lizenzdienstaktivitäten

	Vendor	Lic Manager	Gathering	Status	Action
1	pam_lmd	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
2	oasys	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
3	amlsd	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
4	gns	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
5	ansyslmd	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
6	mim	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
7	abaquslm	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
8	cdlmd	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
9	ugs_lmd	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
10	mnc	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️
11	altair_lm	FlexNet	Realtime	● Up	⚙️ 🔄 ✖️

Abb. 1: Werkzeug für die Überwachung von Lizenzdienstkomponenten.

Bei der Überwachung der Lizenzdienste geht es vor allem um folgende Punkte:

- Überwachung von Lizenzdienstkomponenten
 - Netzwerktechnische Erreichbarkeit der Lizenzserver
 - Verfügbarkeit des allgemeinen Lizenzdienstes (*lmgrd*)
 - Verfügbarkeit der anwendungsspezifischen Lizenzdienste (Vendor Daemons)
- Dokumentation und Überwachung der zeitlichen Lizenzgültigkeit

Um Probleme analysieren zu können, die in der Vergangenheit aufgetreten sind, sollten die Überwachungsergebnisse für notwendige Zeiträume gespeichert werden. Somit können auch Probleme, die während der betriebsfreien Zeiten auftreten, insbesondere nachts oder am Wochenende, nachträglich analysiert werden.

Abbildung 1 zeigt ein Werkzeug für die Überwachung von Lizenzdienstkomponenten. Als Ergänzung zu dieser Darstellung können auch andere Meldemechanismen, z.B. E-Mail- oder SMS-Versand, verwendet werden, um auf aktuelle Probleme aufmerksam zu machen.

Lizenznutzung

Beim Management der Lizenznutzung geht es vor allem darum, die Verwendung der Lizenzen zu steuern bzw. zu optimieren. Wichtige Voraussetzung für spätere Entscheidungen ist die Protokollierung der Lizenznutzung. Um strukturierte Abfragen zu vereinfachen, sollten die relevanten Nutzungsdaten in einer Datenbank hinterlegt werden. Eine typische Abfrage einer solchen Datenbank sieht folgendermaßen aus:

```
SELECT date,avail,usergroup,util FROM  
cax_soft  
WHERE DATE > '2011-08-07'  
AND DATE < '2011-08-18'  
ORDER BY date,avail,usergroup  
;
```

Diese Abfrage ermittelt die Nutzung eines Lizenz-Features in dem angegebenen Zeitraum durch verschiedene Benutzergruppen. Die Ergebnisse solcher Abfragen können in passender Form tabellarisch oder grafisch dargestellt werden. Abbildung 2 zeigt eine Darstellung der Ergebnisse der obenstehenden Datenbankabfrage.



Abb. 2: Darstellung der Lizenznutzung durch verschiedene Nutzergruppen.

Analysen der Lizenznutzung führen ggf. zu geeigneten Optimierungsmaßnahmen. Mögliche Maßnahmen sind z.B. folgende:

- Nutzung der Lizenzen durch unterschiedliche organisatorische Einheiten

Hierbei soll u.a. die Lizenzauslastung gesteigert werden, indem die Anwenderzahl erhöht wird. Falls die Anwendergruppen auch zeitlich zueinander versetzt sind, können Lizenzen für größere Tageszeiträume verwendet werden.

- Zeitliche Verlagerung der Lizenznutzung

Um die maximal benötigte Anzahl von Lizenzen zu reduzieren, können bestimmte Vorgänge von Zeiten der hohen Nutzung auf Zeiten mit geringer Auslastung (nachts oder am Wochenende) verlegt werden.

- Reduzierung der Lizenzanzahl

Falls bestimmte Lizenzen nie oder nur selten in vollem Umfang genutzt werden, kann der beschaffte Umfang reduziert werden.

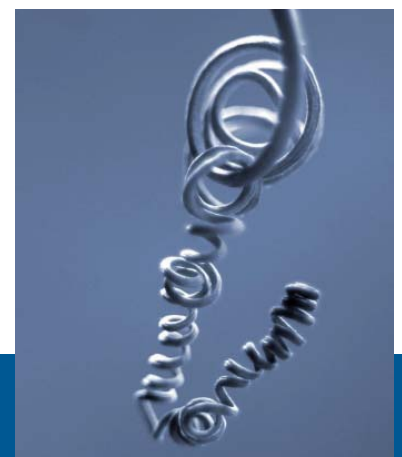
Entsprechende Analysen können auch gewisse Trends aufzeigen, z.B. dass die Nutzung bestimmter Anwendungen zunimmt oder abnimmt. Solche Analysen können ebenfalls Aufschluss über andere Optimierungsmaßnahmen geben:

- Notwendige Änderungen der vorhandenen Hardwareinfrastruktur, um diese ggf. dem analysierten Ressourcenbedarf anzupassen.
- Beschaffung neuer, schnellerer Hardware, um dadurch die Nutzungszeiten der Lizenzen zu reduzieren.

Fazit

Durch die Optimierung der Lizenzverfügbarkeit und der Lizenznutzung können die Anwenderzufriedenheit erhöht und die Lizenzkosten ggf. signifikant gesenkt werden.

Jan Martini, GNS Systems GmbH
jan.martini@gns-systems.de



Eine Frage der Gerechtigkeit - Bewertung von Lizenzkosten bei gemeinschaftlicher Lizenznutzung

In einem Unternehmen können mehrere Abteilungen die Lizenzen einer Software gemeinschaftlich nutzen. Die Vorteile liegen im gemeinsamen Lizenzeinkauf und Lizenz-Hosting, einer besseren Gesamtausnutzung der verfügbaren Lizenzen und einem geringeren Lizenzumfang, verglichen mit dedizierten Lizenz-Pools einzelner Abteilungen.

Allerdings stellt sich die Frage, wie die Kosten der Software auf die Abteilungen verteilt werden sollen. In diesem Artikel werden Aspekte der Aufteilung von Lizenzkosten diskutiert.

Vor einer gemeinschaftlichen Nutzung von Softwarelizenzen sind zunächst die vertraglichen Rahmenbedingungen zu klären. Vorhandene Lizenzverträge, -umfänge und -konditionen sind zu identifizieren und zu konsolidieren. Insbesondere sind dabei Nutzungsbeschränkungen z.B. einer gemeinschaftlichen Nutzung durch mehrere Gesellschaften bzw. in mehreren Ländern zu berücksichtigen.

Die Lizenzierung einer Software besteht meist aus (deutlich) mehr als einem Lizenz-Feature. Die verschiedenen Lizenz-Features liegen i.A. in verschiedenen Anzahlen vor. Als Voraussetzung für eine Aufteilung der Lizenzkosten sind zunächst die Lizenzpakete auf die tatsächlichen Lizenz-Features und -anzahlen im Lizenzservice abzubilden. Ziel ist es, für die Abrechnungsperiode von z.B. einem Jahr jedem kostenpflichtigen Lizenz-Feature einen Betrag zuzuordnen zu können. Bei einer Kombination von Lizenzmiete und -kauf mit Wartungskosten, verschiedenen Rabattierungen und Umwandlung von Arbeitsplatz-gebundenen Lizenzen kann dies bereits einiges Kopfzerbrechen mit sich bringen.

Die zweite Voraussetzung für eine Aufteilung der Lizenzkosten ist eine Darstellung der tatsächlichen Lizenznutzung im Abrechnungszeitraum. Dies setzt eine funktionierende Proto-

kollierung der Lizenznutzung voraus. Eine Nutzungsprotokollierung liefert auch wertvolle Informationen für die zu beschaffenden Lizenzumfänge einer Software. Technisch kann die Nutzungsprotokollierung mit regelmäßigen Snapshots der aktuellen Nutzung oder über die Auswertung von Lizenzserver-Logfiles durchgeführt werden. Beide Methoden haben Vor- und Nachteile. So wird bei Snapshots z.B. alle 10 Minuten ein nur kurz genutztes Lizenz-Feature meist übersehen. Bei Logfiles kann schon eine einzelne fehlende „Check-In“-Zeile die Nutzungsdaten erheblich verfälschen.

Die Lizenznutzung kann z.B. über Benutzer- oder Workstation-Namen auf die Nutzergruppen abgebildet werden. Der Datenschutz gebietet dabei eine sofortige Abbildung auf Nutzergruppen und die Vermeidung von personenbezogenen Nutzungsdaten.

Eine nutzungsbasierte Lizenzkostenaufteilung ist retrospektiv, sie kann erst am Ende des Abrechnungszeitraums die tatsächlichen Nutzungskosten ausweisen. Für eine Planung der Lizenzkosten können die Werte des letzten Zeitraums herangezogen werden. Bei der Lizenzkostenabrechnung hat es sich als sinnvoll herausgestellt, die Lizenzkostenaufteilung inkrementell, z.B. täglich, automatisiert zu erzeugen und den Ansprechpartnern der Nutzergruppen im Intranet des Unternehmens zur Verfügung zu stellen. Die Kostenprognose wird so im Verlauf des Abrechnungsintervalls kontinuierlich präziser und endet im exakten Abrechnungsbetrag der Periode.

Die Aufteilung der Lizenzkosten wird durch eine zentrale Einrichtung im Unternehmen erstellt, deren Kunden die Nutzergruppen der betrachteten Software sind. Das Vertrauen der Nutzergruppen in die Korrektheit der monetären Bewertung der Lizenznutzung ist eng mit einer einfachen, transparenten und nachvollziehbaren Kostenaufteilung verbunden. Das Aufteilungsverfahren ist detailliert zu dokumentieren. Den Ansprechpartnern der Nutzergruppen werden alle für eine Verifizierung nötigen

Daten zur Verfügung gestellt. Eine einfache Kostenaufteilung im Beispiel: Im Jahr 2010 standen fünf Lizenzen des Lizenz-Features „ABC“ zur Verfügung, also $5 \cdot 365 \cdot 24 = 43.800$ Lizenzstunden. Die gesamten Lizenzkosten für „ABC“ belaufen sich auf 12.000 EUR. Die tatsächliche Gesamtnutzung der Lizenz war 30.000 Lizenzstunden. Somit kann eine Lizenzstunde mit $12.000/30.000 = 0,40$ EUR bewertet werden. Abteilung „XYZ“ nutzte 12.500 Lizenzstunden „ABC“, was mit $12.500 \cdot 0,40 = 5.000$ EUR bewertet wird.

Dieses einfache Beispiel lässt sich auf eine Nutzung vieler Lizenz-Features erweitern. Zu beachten sind auch Änderungen der Lizenzanzahlen innerhalb der Abrechnungsperiode. Je nach Nutzungsverhalten können bestimmte Zeiträume wie Wochenenden und für interaktive Software die Nachtstunden bei der Lizenzkostenaufteilung ausgeklammert werden.

Spannend wird die Bewertung von Lizenz-Features, die angeschafft, aber nicht oder kaum genutzt werden. Auch ganz ohne Nutzung fallen die Kosten für diese Lizenz-Features an. Wird ein solches Feature genau einmal genutzt, könnten die gesamten Kosten dieser einen Nutzung zugeordnet werden. Da dies oft als nicht gerecht angesehen wird, können Nutzungsschwellwerte eingeführt werden, unterhalb derer eine Nutzung nicht zugeordnet oder abgerechnet wird. Fraglich ist auch, ob marginale Nutzungskosten wie 0,23 EUR im Jahr einer Abteilung überhaupt in Rechnung gestellt werden sollten. Zudem können bei der Softwarebereitstellung und beim Support Lizenznutzungen anfallen, deren Kosten anteilig den produktiven Nutzergruppen zugeschlagen werden sollten.

All diese und weitere Aspekte führen zu einem komplexen, aber möglichst gerechten Abrechnungsverfahren und bieten Raum für Gestaltungsmöglichkeiten im Sinn der Unternehmensziele.

*Dr. Marcus Renner, GNS Systems GmbH
marcus.renner@gns-systems.de*

Verbundvorhaben FEMMINER

Im Rahmen des *KMU-Innovationsprogramms* des Bundesministeriums für Bildung und Forschung hat GNS Fördermittel zur Umsetzung des Verbundvorhabens *FEMMINER* bewilligt bekommen.

Kooperationspartner von GNS ist das *Fraunhofer Institut für Algorithmen und Wissenschaftliches Rechnen (SCAI)*. Das Projekt behandelt das Thema „Data Mining“ und besteht aus den Bausteinen:

- einfaches Simulationsdatenmanagement
- Variantenexploration
- nichtlineare Klassifikation

Die Projektlaufzeit beträgt zwei Jahre, das Entwicklungsvolumen ca. 1 Million Euro. Derzeit gibt es Gespräche zum Ersteinsatz mit der *AUDI AG* und der *VOLKSWAGEN AG*.

Steckbrief

Problemstellung

Die Einführung von Elektrofahrzeugen stellt durch die Forderung nach extremem Leichtbau und vollständig neuen Fahrzeugkonzepten besondere, neue Anforderungen an die Produktentwicklung. Hier müssen in kurzer Zeit neue Ansätze bewertet, erfolgreiche Trends gefunden und gleichzeitig der Überblick über eine Vielzahl unterschiedlicher Ansätze gewahrt werden.

Lösungsansatz

Durch die Entwicklung neuer Methoden zur Analyse von Simulationsdaten sollen dem Berechnungsingenieur Hilfsmittel in Form von Softwaremodulen bereitgestellt werden, welche die Sicht auf Verzweigungspunkte in der Entwicklungshistorie eines Produktes ermöglichen.

Kernarbeiten des Projektes

Zur Realisierung dieser neuartigen Datenexplorationsmethode werden drei Module entwickelt:

- simpleSDM (schlankes Simulationsdatenmanagement)
- CAE-Modell Variantenexploration
- nichtlineare Klassifikation

Diese Module werden in die *GNS Produkte Generator3* und *Animator4* integriert. Eine intuitiv bedienbare Oberfläche erleichtert das Auffinden von bestimmten Varianten. Außerdem können Verhaltensmuster ermittelt werden, welche bei einer Vielzahl von Simulationen auftreten.

Verwertungspotential, Anwendernutzen, Bundesinteresse

Das im Rahmen des Forschungsvorhabens zu entwickelnde System wird ein besseres Verständnis bzgl. der Zusammenhänge von Ursache und Wirkung in der Produktentwicklung ermöglichen.

Die dem System zugrundeliegenden Verfahren können überall dort zum Einsatz kommen, wo Finite-Elemente-Simulationen die Produktentwicklung begleiten. Sie sind also in vielen Industriebereichen einsetzbar. Der volkswirtschaftliche Nutzen ergibt sich durch eine erhebliche Effizienzsteigerung während der Produktentwicklung.

Claudius Schöne, GNS mbH
schoene@gns-mbh.com

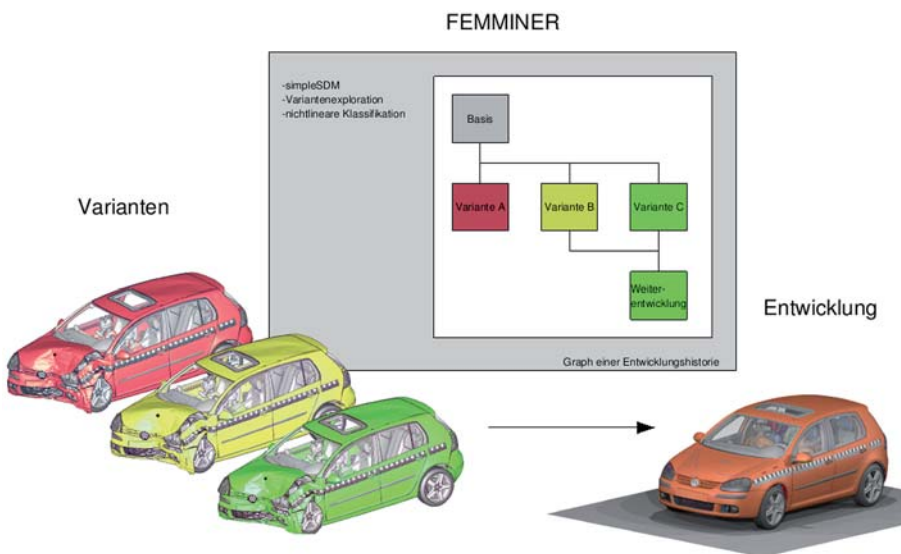


Abb. 1: Entwicklungshistorie eines Produktes



Termine

Im Augenblick stehen keine Termine fest.

Bis zum Erscheinen des nächsten GeNiuS können Sie unsere Termine auf den jeweiligen Webseiten von GNS mbH und GNS Systems GmbH erfahren.



GNS mbH

Am Gaußberg 2
38114 Braunschweig
Telefon: 05 31-8 01 12 0
Fax: 05 31-8 01 12 79
www.gns-mbh.com



GNS Systems GmbH

Am Gaußberg 2
38114 Braunschweig
Telefon: 05 31-1 23 87 0
Fax: 05 31-1 23 87 11
www.gns-systems.de

Impressum

Ausgabe 1/2011
Erscheinungstermin: Oktober 2011
Herausgeber: GNS Systems GmbH

Verantwortlich: Jan Martini
Redaktion: Anette Tröger
Layout: Anette Tröger

Alle Rechte vorbehalten.
Vervielfältigung, auch auszugsweise, nur mit schriftlicher Genehmigung des Herausgebers.
Alle Markennamen sind in der Regel eingetragene Warenzeichen der entsprechenden Hersteller oder Organisationen.

Verwendung von Variablen in Animator4

Eine Übersicht aller Variablen findet man unter *User/Variables*. Im Drop-Down-Menü kann man zwischen verschiedenen Variablentypen von *Animator4* wählen. Alle Variablen teilen sich den gleichen Namensraum. Variablen kann man in jedem Befehl benutzen. Hierzu gibt man {VNAME} ein, wobei NAME der Variablenname ist. Ein Doppelklick auf einen Variablennamen im Variablendialog fügt die Variable in die Kommandozeile ein.

1. User Variables

Über den Befehl "set var ..." wird einer Variable ein Wert zugewiesen. Dieser Befehl kann benutzt werden, um Einstellungen für Modellvarianten, wie z.B. Dateinamen, vorzunehmen.

"set var str ins Dateiname Variante.DSY" erzeugt die Variable Dateiname mit dem Inhalt Variante.DSY.

Variablen können auch beim Aufruf von *Animator4* in der Kommandozeile gesetzt werden, beispielsweise mit a4 -VDateiname=Variante.DSY. Variablen, die mit "set con" statt "set var" erzeugt werden, sind konstant, d.h. schreibgeschützt.

Auch das Ergebnis einer Berechnung kann in einer Variablen gespeichert werden:

```
c2d cal NAME = amax( [0:"curvename"] )
```

In der Variablen NAME wird der größte y-Wert der Kurve curvename aus Modell "0" gespeichert.

2. System Variables

Diese Variablen werden von *Animator4* zur Verfügung gestellt und enthalten Werte wie den Modellnamen ({Vs1ot:SLOTNAME}) oder die Anzahl der Verschiebungszustände ({Vs1ot:NUMSTATE}). Anstelle von slot muss hier eine Modellnummer verwendet werden.

3. Environment Variables

Umgebungsvariablen, die von der aufrufenden Shell an *Animator4* mitgegeben werden, können hier modifiziert werden.

Außerdem ist es möglich, eigene Variablen zu erzeugen.

4. Result Variables

Viele Befehle generieren automatisch Variablen (in den folgenden Beispielen mit (*) gekennzeichnet). Alle Informationen, welche im Message-Fenster angezeigt werden, werden auch in Ergebnisvariablen gespeichert. Die Namen der Ergebnisvariablen beginnen mit _ und sie sind nur solange gültig, bis ein neuer Befehl Ergebnisvariablen erzeugt.

Ein einfaches Beispiel, um in den Verschiebungszustand mit der größten Verschiebung zu wechseln:

```
dis inf all all max (*)  
sta set {V_STAMAX}
```

Ein zweites Beispiel zeigt, wie im Modellfenster ein Text in blauer Schrift mit Informationen (Dicke und Material) über ein Bauteil angezeigt werden kann:

```
ide pid @si (*)  
txt pid add {V_PI} "Part Name={V_NAME}\nMaterial ID={V_MID}\nThickness={V_THICK}" (*)  
s[ {V_SLOTID} ]v[ {V_VIEWID} ]:txt  
mod col for blue {V_TEXTID}
```

Das letzte Beispiel verdeutlicht, wie man in Abhängigkeit der Kurvenwerte den Bereich einer Achse anpassen kann, wenn der Standardbereich (von 0 bis 120) überschritten wird. Hierbei wird auf 10er Werte gerundet.

1. Y-Achse selektieren:
pre sel axi 2 1
2. Maximale y-Werte der zugehörigen Kurven: pre ide pro (*)
3. Wenn ein y-Wert größer als 120 ist, anpassen des Wertebereichs der Achse:
c2d cal ymax = if (120 < {V_MAX_CURVERANGE}, 10*ceil({V_MAX_CURVERANGE}/10.), 120) (*)
pre set ran yax 0 {V_R}

Die im letzten Befehl verwendete Ergebnisvariable _R enthält immer das Ergebnis des letzten "c2d cal"- oder "pre cal"-Befehls.

Carsten Thunert, GNS mbH
animator@gns-mbh.com