Seminar Langzeitarchivierung Der Einsatz des UVC in der Koninklijken Bibliotheek

Gerrit Kahn
1005156
8. Juni 2009

Aufgabensteller:
Prof. Dr. Uwe M. Borghoff
Betreuer:
Dipl.-Inform. Nico Krebs

Institut für Softwaretechnologie
Fakultät für Informatik
Universität der Bundeswehr München

Für große Datenbestände, wie z.B. die von Bibliotheken, stellt die digitale Langzeitarchivierung (LZA) eine große Herausforderung dar. Die Idee des „Universal Virtual Computer“ (UVC) von Raymond A. Lorie ermöglicht ein Archivierungskonzept, das einige entscheidende Vorteile gegenüber den Standardmethoden der LZA (Emulation und Migration) bietet.
Der UVC ist eine Art virtueller Allzweck-Computer, dessen Spezifikation so einfach gehalten ist, dass davon ausgegangen werden kann, dass er auch in ferner Zukunft noch verstanden und implementiert werden kann. Das durch ihn ermöglichte Konzept ist vor allem bei großen Datenbeständen sehr viel weniger aufwändig als z.B. die Migration.
Die niederländische Koninklijke Bibliotheek hat (in Zusammenarbeit mit IBM) im Rahmen einer Machbarkeitsstudie einen UVC implementiert und seinen Einsatz zur LZA von digitalen Bildern getestet.
Obwohl auch der UVC kein „Allheilmittel“ gegen die drohende Unzugänglichkeit von alternden Daten bietet, sind die Ergebnisse sehr vielversprechend und rechtfertigen weitere Forschungen auf diesem Gebiet.

Inhaltsverzeichnis

1 Einleitung

Viele Autoren bzw. Verleger geben ihre Werke inzwischen in digitaler Form heraus. Dies kann z.B. in Form von PDFs, Webseiten oder Multimedia-CDs geschehen. Vor allem im sogenannten „Science-Technology-Medicine (STM)“ Bereich werden immer mehr Forschungsergebnisse nur noch digital publiziert. Dies bietet natürlich viele Vorteile, birgt jedoch auch einige Gefahren. Die wohl größte davon zeigt sich in Hinblick auf die Langzeitarchivierung (LZA) dieser digitalen Objekte. In Zeiten, in denen ständig neue (oft auch noch proprietäre) Dateiformate erscheinen, immer wieder neue Software herausgebracht wird (die ältere Software unnütz macht) und sich die Hardwarehersteller ein technologisches Wettrennen liefern, durch das einjährige Hardware bereits als veraltet gilt, ist die Gefahr sehr groß, dass heute gespeicherte Informationen in Zukunft unzugänglich sind.
Die Koninklijke Bibliotheek (KB) hat sich diesem Problem gestellt und unter anderem die *„Sicherung der permanenten Zugänglichkeit von Informationen und Wissen“* zum Ziel gemacht [3]. Hierzu wurde das Konzept des „Universal Virtual Computer (UVC)“ von Raymond A. Lorie [6] aufgegriffen und in Zusammenarbeit mit IBM implementiert [4].
Im folgenden Text wird beschrieben, was zur Entwicklung des UVC geführt hat, was der UVC genau ist, wie er in der KB eingesetzt werden sollte und wie es um seine Praxistauglichkeit bestellt ist.

2 Gründe für die Entwicklung des UVC

Auch der UVC ist nicht von heute auf morgen entwickelt worden, dennoch verwendet er keinerlei neue Technologien sondern kombiniert altbekannte und bewährte Methoden auf neue Art und Weise. Es wird nun beschrieben, welche Gründe zu seiner Entwicklung bzw. Erprobung bei der KB geführt haben.

2.1 Probleme der LZA bezogen auf den UVC

Die beiden bisher meist genutzten Verfahren, die Migration und Emulation, bieten beide ihre Vor- und Nachteile. Da diese beiden Verfahren nicht speziell zur LZA konzipiert wurden, sind sie in diesem Gebiet nicht uneingeschränkt bzw. ohne Mehraufwand verwendbar.
So kann wiederholt ausgeführte Migration mit jedem neuen Migrationszyklus zu immer stärkeren Verfälschungen der Wiedergabe führen. Zudem ist dieser Ansatz für große Datenbestände wie den der KB (im November 2007 über 5.000 e-journal Titel und 10 Millionen e-journal Artikel; insgesamt ca. 11 Terabyte [1]) extrem aufwendig, da es auf jede einzelne Datei angewendet

werden muss.
Auch der Emulationsansatz ist nicht problemlos auf die LZA anwendbar. So ist es z.B. schwierig bis unmöglich, an die Spezifikationen veralteter Systeme zu gelangen, ohne die jedoch die Entwicklung einer korrekten Emulation sehr aufwendig wird.
Zur ausführlichen Beschreibung der Emulation/Migration sowie deren Vor- und Nachteile siehe z.B. [2].

2.2 Bedürfnisse der KB

Wie in der Einleitung schon erwähnt wurde, werden immer mehr Publikationen nur noch digital veröffentlicht. Aber auch andere (z.B. kulturell wertvolle) Werke werden immer öfter auch digital gespeichert. Um diese digitalen Objekte archivieren zu können, ist es notwendig besondere Verfahren und Softwaresysteme zu entwickeln, denn den IT Unternehmen ist das Problem der nur relativ kurzen Zugänglichkeit zu digitalen Objekten erst kürzlich bewusst geworden [10]. Die KB hat hierzu zusammen mit IBM ein System entwickelt: das sogenannte *e*-Depot.
Die KB spielt mit ihrem System eine große Rolle im „Safe Places Network“ (eine systematische Kooperation mit Herausgebern und Verlegern, die ihre Werke an eine begrenzte Zahl an „Safe Places“ in Obhut geben), möchte ihre Position aber noch weiter ausbauen. Laut Angaben der KB decken die 20 größten Verlage weltweit über 90% der elektronischen STM Literatur ab und die KB möchte dieses Level ebenfalls in ihrem *e*-Depot erreichen [11].
Hierzu muss die KB ihren Partnern natürlich ein dauerhaftes Archiv bieten. Dauerhaft bezüglich der Speicherung der Daten aber vor allem auch dauerhaft bezüglich des Zugriffs auf die Daten. Es wird also ein Tool gebraucht, dass den permanenten Zugriff auf die im *e*-Depot gespeicherten Daten ermöglicht.
Ein spezieller Emulator mit entsprechenden Funktionen könnte diese Aufgabe erfüllen. Allerdings stellt sich hier nun (zusammen mit den im vorherigen Abschnitt besprochenen Problemen) folgende Fragestellung, auf die der UVC eine mögliche Antwort darstellt:

> „Wie können wir sicherstellen, dass ein Emulator, der imstande ist, archivierte digitale Objekte in ihrer (virtuellen) Originalumgebung auszuführen, korrekt auf zukünftigen Computern laufen wird, ohne zu Wissen, wie diese zukünftigen Plattformen aussehen werden?“ [4]

3 Was ist der UVC?

Es muss an dieser Stelle zwischen dem UVC an sich und dem Konzept zur LZA, welches der UVC ermöglicht, unterschieden werden. Zu diesem Konzept gehören außer dem UVC noch einige weitere Elemente. Es wird nun zunächst das Konzept und dann der Aufbau des UVC erläutert.

3.1 Konzept der UVC-basierten LZA

Für das Format einer zu archivierenden Datei wird eine logische Beschreibung seiner Struktur, das sogenannte „Logical Data Scheme (LDS)“, erstellt. Zusätzlich wird ein Programm geschrieben, welches auf dem UVC läuft und die Struktur der Datei anhand des LDS analysieren und in eine logische Form bringen kann (der Decoder). Um es den zukünftigen Entwicklern zu ermöglichen, wieder einen UVC zu programmieren, auf dem der Decoder läuft, muss die Spezifikation des verwendeten UVC ebenfalls archiviert werden.
Soll nun in Zukunft eine Datei mit einem nicht mehr gängigen Dateiformat betrachtet werden, muss zunächst ein UVC anhand der überlieferten Spezifikation entwickelt werden. Nun kann der Decoder auf diesem UVC ausgeführt werden. Der UVC kann dann die logische Sicht der archivierten Datei, die „Logical Data View (LDV)“, als Bitstream ausgeben. Die Form des Streams wird von der LDS bestimmt. Er beinhaltet die Nummern sämtlicher Elemente sowie die jeweils zu den Elementen gehörenden Werte. Mit Hilfe des LDS lässt sich die LDV wieder in eine leicht verständliche Fassung in Form einer Auszeichnungssprache (z.B. XML) bringen. Abbildung 1 zeigt das LDS und die bereits in XML umgewandelte Version der LDV einer Bilddatei.

```
ELEMENT 01 [image](10,24+)

ELEMENT 10 [imageSize](11,12)
ELEMENT 11 [numScanLines]
ELEMENT 12 [pixelsPerScanLine]

ELEMENT 24 [pixel](25,29)

ELEMENT 25 [colour](26,27,28)
ELEMENT 26 [red]
ELEMENT 27 [green]
ELEMENT 28 [blue]

ELEMENT 29 [pixelPosition](30,31)
ELEMENT 30 [scanLine]
ELEMENT 31 [pixelNum]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<image>
    <imageSize>
        <numScanLines>38</numScanLines>
        <pixelsPerScanLine>39</pixelsPerScanLine>
    </imageSize>
    <pixel>
        <colour>
            <red>255</red>
            <green>255</green>
            <blue>255</blue>
        </colour>
        <pixelPosition>
            <scanLine>0</scanLine>
            <pixelNum>0</pixelNum>
        </pixelPosition>
    </pixel>
    ...
```

Abbildung 1: Ausschnitt aus einem LDS und LDV einer Bilddatei (LDS aus [4], LDV entstanden durch eigenen Probelauf des Prototyps[1]).

Das Letzte, was noch getan werden muss um die Datei anzuzeigen, ist ein entsprechendes Betrachtungsprogramm zu entwickeln. Dieser sogenannte „Viewer“ kann nun schließlich die Datei anhand der LDV und des LDS auf dem Bildschirm (oder sonstigen zukünftigen Anzeigegeräten) ausgeben. Abbildung 2 stellt diese Vorgänge schematisch dar. Auf der linken Seite sind alle Elemente zu sehen, die archiviert werden müssen. Die rechte Seite zeigt den Prozess zum Anzeigen der archivierten Datei.

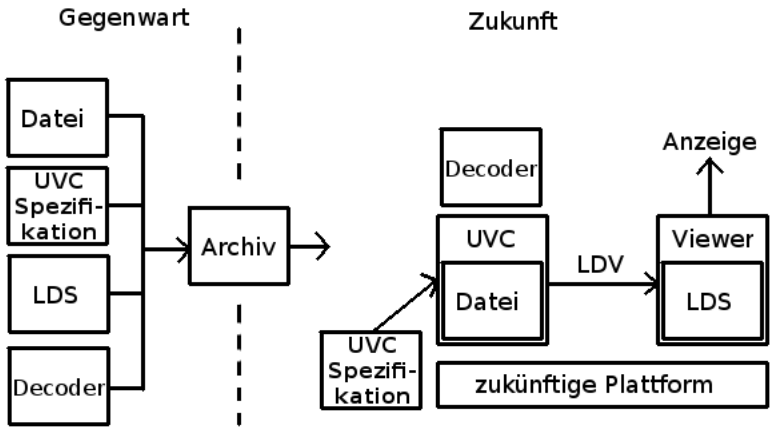

Abbildung 2: Schema des UVC-basierten Archivierungsprozesses (nach [3])

Im Vergleich zur Migration, wo jede Datei eines Archivs einzeln transformiert werden müsste, ist der Aufwand dieses Ansatzes deutlich geringer. Es muss auch nicht zwingend für jedes Dateiformat ein LDS entwickelt werden. Oft reicht ein Schema für eine Kategorie von Dateitypen aus. So verwendet z.B. auch der von IBM und der KB entwickelte Prototyp das gleiche Schema für GIF- und JPEG-Dateien [4].

3.2 Funktionsweise

Der UVC ist ein virtueller Allzweck-Computer, der so einfach gehalten ist, dass er auch in Zukunft noch leicht verstanden und rekonstruiert werden kann. Die Grundlage ist sein segmentiertes Speichermodell. Hier wird jedes Programm bzw. jeder Teil eines Programms in einem eigenen Speichersegment abgelegt. Dies soll verhindern, dass sich Programme versehentlich gegenseitig überschreiben.

[1] http://www.alphaworks.IBM.com/tech/uvc

3.2.1 Architektur

Die Spezifikation des UVC ist bewusst einfach gehalten. Grund hierfür ist, dass man so davon ausgehen kann, dass die Architektur des UVC auch in der Zukunft noch verstanden und implementiert werden kann. Sein Speichermodell und seine verfügbaren Instruktionen sind so grundlegend, dass zumindest für eine sehr lange Zeit davon ausgegangen werden kann, dass zukünftige Programmierer damit umzugehen wissen. Um zu zeigen, wie einfach der UVC wirklich ist, werden im Folgenden sein segmentiertes Speichermodell und die Instruktionen auf Basis von [8] genauer erläutert.

Segmente Jedes einzelne Speichersegment hat eine theoretisch unbegrenzte Anzahl an Registern und ebenso unbegrenzten Speicher. Jedes Register und damit jedes Segment kann auf beliebige Größe anwachsen. Eindeutig identifiziert werden die Segmente über physikalische Segmentnummern.
Programme für den UVC bestehen aus mehreren Teilen, die miteinander agieren. Die einzelnen Teile eines Programms werden „Sections“ genannt und in separaten Segmenten gespeichert. Jede einzelne *Section* kann alle Segmente ansprechen, die in ihrem Adressraum liegen (alle verwendeten Segmente werden am Anfang einer *Section* angegeben). Die Segmente eines Adressraums sind eindeutig identifiziert über die logische Segmentnummer. Während der Ausführung weist der UVC die physischen Segmente den logischen zu und speichert die Zuweisungen (in der Implementierung des Prototyps z.B. in einer „Dispatch Table“; siehe unten). Hierdurch muss z.B. Code, der mehrmals ausgeführt wird, nur einmal gespeichert werden. (Segmentnummern im Folgenden beziehen sich, sofern nicht anders angegeben, immer auf logische Segmente.)
Der Adressraum einer *Section* beinhaltet mindestens die reservierten Segmente 0, 1, 2 plus ein Segment mit dem Code der *Section*. Es folgt nun eine Beschreibung der einzelnen Segmente [8].

- Segment 0 kann von allen *Sections* adressiert werden. Es beinhaltet globale Konstanten und Variablen und kann somit z.B. die Daten der zu decodierenden Datei aufnehmen.
- Segment 1 ist von der *Section* adressierbar, zu der es gehört. Wenn die *Section* rekursiv aufgerufen wird, kann das Segment von jeder Instanz gelesen werden. Es dient als gemeinsamer Speicher für mehrere Instanzen einer *Section*.
- Segment 2 wird zur Übergabe von Parametern zwischen den *Sections* benutzt. Wenn *Section* A *Section* B aufruft, wird A das Ergebnis von B in A's Segment 2 sehen können.
- Segmente 3 bis 999 sind global (gehören zum Adressraum aller *Sections*). Rufen z.B. zwei *Sections* das Segment 4 auf, so werden sie den

selben Inhalt erhalten.
- Segmente über 1000 sind privat. Rufen zwei *Sections* das gleiche Segment auf, werden sie unterschiedliche Inhalte erhalten, denn sie werden auf verschiedene physikalische Segmente verwiesen.

Instruktionen UVC Programme bestehen aus einem vordefiniertem Satz an Instruktionen. Es gibt insgesamt 25 verschiedene Befehle. Sie lassen sich in verschiedene Kategorien einteilen, die im Folgenden erläutert werden. Für eine vollständige Liste und Beschreibung aller Instruktionen siehe [8, S.8-12].

- Logische Bit Manipulation
  Hiermit werden bitweise Operationen (wie „not“, „and“, „or“) auf Register angewendet.
- Arithmetische Operationen
  Hierunter fallen die Befehle zum Addieren, Subtrahieren, Multiplizieren und Dividieren der Werte zweier Register nach den Regeln der binären Arithmetik.
- Vergleiche
  Vergleicht zwei Register auf größer oder gleich.
- Speicheroperationen
  Instruktionen zum Bewegen von Daten zwischen Registern und dem Speicher („load“ und „store“).
- Registeroperationen
  Das Laden des Inhalts eines Registers in ein weiteres, Setzen des Vorzeichens auf positiv oder negativ, Einfügen einer bestimmten Anzahl Bits von einem gegebenen String und Anzeigen der Länge eines Registers.
- Instruktionen zum Ändern des Ausführungsflusses
  Hierunter fallen alle Befehle zum Springen zwischen den *Sections* und innerhalb des Codes der aktiven *Section* sowie das Unterbrechen und Stoppen der Ausführung.

3.2.2 Realisierung

In [4] wird eine Möglichkeit der Implementierung vorgestellt, welche zeigt, wie die Spezifikation eines UVC leicht mit gängigen Methoden und Mustern der Informatik umgesetzt werden kann. In dieser Implementierung wird die Verwaltung und Zuweisung der Segmente und *Sections* durch eine *Dispatch Table* und den „Activation Stack“ übernommen. In der Spezifikation des UVC sind diese Elemente nicht namentlich erwähnt, was zeigt, dass auch hier den Programmierern der Zukunft freie Hand bei der Art der Umsetzung gelassen wird. Im Folgenden wird die erwähnte Implementierung vorgestellt

um zu zeigen, wie sich die Vorgaben des UVC mit relativ einfachen und altbekannten Methoden umsetzen lassen.

Dispatch Table Um den Überblick über alle geladenen *Sections* zu behalten wird eine Zuordnungstabelle, die *Dispatch Table*, angelegt. Jede *Section* bekommt eine eindeutige ID, die als Schlüssel zur Identifizierung der einzelnen *Sections* innerhalb des UVC dient. Erhält der UVC eine neue *Section*, so wird seine ID ausgelesen und zwei neue Segmente angelegt. Ein Segment für den Code der *Section* und ein zweites für ihr Segment 1 (siehe vorheriger Abschnitt). Nun werden noch alle logischen Segmente erfasst, die von dieser *Section* verwendet werden. All diese Informationen werden dann in der *Dispatch Table* gespeichert. Sie kann nun zur Navigation während der Ausführung eines Programms benutzt werden. Abbildung 3 zeigt die schematische Darstellung einer *Dispatch Table* nachdem zwei *Sections* geladen wurden.
Die Einträge der benutzten logischen Segmente werden später vom *Activation Stack* zum Abarbeiten der Programmschritte benutzt (siehe unten).

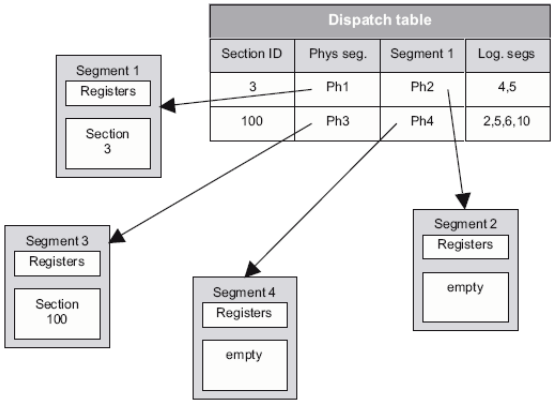

Abbildung 3: *Dispatch Table* nach dem Laden zweier *Sections* [4]

Activation Stack Der Stapel des UVC arbeitet nach dem „Last-In-First-Out (LIFO)“ Prinzip. Elemente können über eine „push“ Operation zum Stapel hinzugefügt und über eine „pop“ Operation vom Stapel entfernt werden.
Sobald also eine *Section* instanziiert wird, wird ein sogenannter „Activation Record“ erzeugt und mittels dem *push* Befehl an die oberste Stelle des *Activation Stack* gestellt. Jeder Aufruf einer *Section* wird einen neuen *Activation Record* erzeugen und an die oberste Stelle des Stapels platzieren. Es ist auch möglich, dass sich eine *Section* selbst aufruft. Hierdurch werden rekursive Funktionen ermöglicht.

Ist der Code der aktiven *Section* zu Ende ausgeführt, so wird der entsprechende *Activation Record* mittels *pop* Befehl vom Stapel entfernt und der darunter liegende Eintrag aktiv. Dieser Prozess wird so lange wiederholt, bis keine Einträge mehr vorhanden sind. Tabelle 1 zeigt den schematischen Aufbau des *Activation Stack*.

Section ID	Physical segment	Physical segment 1	Logical to physical mapping	Return adress (section, adress)

Tabelle 1: Schematische Darstellung des *Activation Stack* [4]

Informationen wie z.B. Parameter werden zwischen den einzelnen *Sections* des Stapels über ihr Segment 2 ausgetauscht (siehe Abschnitt „Segmente“).

Weitere Elemente Ferner wurden für die Umsetzung des Prototyps folgende Aufgabenbereiche identifiziert:

- Memory Manager
  Der „Memory Manager“ organisiert das Speichen und Laden von Daten innerhalb des UVC. Er kontrolliert die Zuordnung sowie die Benutzung der einzelnen Segmente anhand der *Dispatch Table*.
- Execution Manager
  Der „Execution Manager“ stellt das Herz des UVC dar. Er kontrolliert die Ausführung der Instruktionen (sequenziell und nur eine zur gleichen Zeit [4]). Er überwacht dazu die Instruktionen, den *Activation Stack* und die Einträge der *Dispatch Table*.
- I/O Manager
  Dieser Manager implementiert einen abstrakten Kommunikationskanal zwischen dem *Viewer* (siehe Abschnitt 3.1) und dem UVC. Über diesen Kanal können Daten, Konstanten und *Sections* [4] in den UVC geladen werden und im Gegenzug die verarbeiteten Daten als Elemente des LDV wieder ausgegeben werden.

4 Geplanter Einsatz in der KB

Im Rahmen der Erprobung des UVC-Konzepts wurde der bereits erwähnte Prototyp erfolgreich zu Testzwecken im elektronischen Archiv der KB (dem sogenannte *e*-Depot) zur Archivierung und Betrachtung von Bilddateien eingesetzt. Das *e*-Depot ist bereits jetzt durch verschiedene Maßnahmen (siehe z.B. [10]) auf einen langen Zugang zu den archivierten Publikationen ausgelegt. Mit dem Einsatz des UVC wollte die KB diesen bereits auf lange Zeit gesicherten Zugang erweitern. Im Idealfall sollte ein permanenter Zugriff auf die Dateien des Archivs ermöglicht werden.

Aufgrund der in Abschnitt 5.1 gezeigten Probleme dieses Ansatzes kam es bis heute jedoch noch nicht zur Anwendung in der täglichen Praxis.

5 Schlussfolgerungen

Es folgen nun einige abschließende Betrachtungen über die Grenzen des UVC, aktuelle und zukünftige Forschungsschwerpunkte sowie eine Einschätzung seiner Praxistauglichkeit.

5.1 Grenzen des Ansatzes

Auch das Konzept des UVC hat mit einigen Nachteilen bzw. Problemen zu kämpfen. Einer dieser Nachteile ist, dass die LDV einer Datei sehr ineffizient in Bezug auf die Verarbeitungsgeschwindigkeit ist.
Dies führt direkt zum Problem: die Performance. Will man z.B. mit dem Demo-UVC von IBM ein JPEG mit 595 mal 842 Pixeln (entspricht etwa einer DIN A4 Seite eines PDFs bei 72 DPI) anzeigen, so dauert dies auf einem durchschnittlich schnellen PC (Intel Core2Duo E7300 2x2,67GHz, 3GB RAM, Kubuntu 9.04, Sun Java 6) gut 10 Minuten. Stellen wir uns nun die Wiederherstellung (im Sinne von Anzeigen einer archivierten Datei) einer Publikation mit 300 Seiten vor, so kommen wir auf eine Wiederherstellungsdauer von etwa 2 Tagen und 2 Stunden.
Sicherlich würde die KB sehr viel leistungsstärkere Großrechner für diese Aufgabe einsetzen. Allerdings könnten diese Großrechner aufgrund der (durchaus gewollten) Einfachheit des UVC ihre Leistungsfähigkeit bei seiner Ausführung zum größten Teil nicht einsetzen. So ist es beispielsweise unerheblich mit wie vielen Prozessoren ein Großrechner ausgestattet ist, da der UVC aufgrund seiner linearen Ausführung der Programme keinen Nutzen aus mehreren parallelen CPUs ziehen kann.
Nun ist die LDV von einzelnen Dateien aus logischer Sicht noch nicht sehr komplex. Soll aber die gesamte Logik eines Programms archiviert und wiederhergestellt werden, so wird die LDV sehr komplex und damit die Wiederherstellung bzw. Ausführung um ein Vielfaches langsamer.
Dieses sind wohl auch einige der Gründe, weshalb die KB den UVC noch nicht in der Praxis einsetzt. Stattdessen konzentriert sie sich zur Zeit mehr auf die Entwicklung eines modularen Emulators („Dioscuri“[2]). Der UVC selbst wird im Rahmen des Projekts „Planets“[3] weiterentwickelt. Im Folgenden Abschnitt wird erläutert, welche Forschungsschwerpunkte dort zur Zeit unter anderem bearbeitet werden.

[2] http://dioscuri.sourceforge.net/
[3] http://www.planets-project.eu

5.2 Forschungsschwerpunkte

Das gesamte Geschwindigkeitsproblem relativiert sich, wenn man bedenkt, dass zukünftige Plattformen um ein Vielfaches schneller arbeiten werden, als es die heutigen tun. Außerdem läuft der Prototyp auf einer JVM, was in diesem Fall im Vergleich zu einer compilierten Version langsamer ist (siehe unten). Allerdings hatte bei der Entwicklung des Prototyps die Performance keinen Vorrang, sondern die Flexibilität. Es ist daher davon auszugehen, dass im Bereich der Performance noch einiges an Verbesserungspotenzial vorhanden ist.
So wurde im Rahmen von *Planets* erfolgreich getestet, ob sich tatsächlich nur anhand der Spezifikationen ein neuer UVC konstruieren lässt. In dieser Implementierung wurde C++ statt Java als Programmiersprache gewählt, was die JVM überflüssig macht. Weitere Verbesserungen der Performance wurden unter anderem erreicht durch: Verwendung eigener Funktionen zur Speicherverwaltung statt Hashmaps oder ähnliche Strukturen, Verwendung von Sprungtabellen statt Case- oder Select-Anweisungen und effizientere Berechnung von kleinen Zahlen[4]. Hierdurch konnte die Performance des UVC von 17.000 Instruktionen pro Sekunde [4] auf 1 MIPS erhöht werden[5].
Für obiges Berechnungsbeispiel würde dies eine deutliche Reduzierung des Zeitaufwands für eine Seite auf etwa 10,2 Sekunden und für ein ganzes Dokument mit 300 Seiten auf etwa 51 Minuten bedeuten.
Eine weitere Restriktion des Prototyps ist die Beschränkung auf Archivierung und Wiederherstellung von einzelnen Dateien. Hier wird zur Zeit jedoch bereits an der Erweiterung auf komplette Programme gearbeitet. Es wurde bereits erfolgreich die Logik eines Spreadsheet Programms (Sharp Tools Spreadsheet[6]) archiviert und wiederhergestellt[5]. Auch hier bleibt die weitere Entwicklung abzuwarten.

5.3 Einschätzung der Praxistauglichkeit

Zur Zeit kann man wohl behaupten, dass der UVC noch nicht für den Einsatz in der alltäglichen Praxis geeignet ist. Hierzu müssen zunächst noch oben genannten Probleme gelöst bzw. zumindest vermindert werden. Danach können dann alltagstaugliche Versionen des Prototyps entwickelt werden, welche außer den gewollten Funktionen auch eine einfache Bedienung bieten.
Nichtsdestotrotz ist das Konzept des UVC sehr interessant und wird vermutlich in Zukunft eine wichtige Rolle im Bereich der LZA spielen. In welcher Form er dies tun wird bleibt allerdings abzuwarten.

[4] siehe Mail von Jasper Schroder im Anhang auf Seite 18
[5] siehe Mail von Jasper Schroder im Anhang auf Seite 17
[6] http://www1.cs.columbia.edu/sharptools/

Anhang

Re: UVC

Von: Jasper Schroder <jasper@nl.ibm.com>
An: Gerrit Kahn <jahudi@gmx.net>
Datum: 2009-04-30 19:11

Hi Gerrit,

One of the research projects was to preserve the logic of a spreadsheet program. This was done with the Sharp tools spreadsheet as it completely written in Java. The logic of the spreadsheet was recreated in an UVC program and and a front-end application was created for the input/output of data.

The performance of the UVC has also been improved. On a current machine the UVC runs with just over 1Mips.

I hope this helps and please feel free it you have more questions.

Note: next week I'm not in the office.
Jasper Schröder
IBM Nederland B.V
IBM Center for Advanced Studies Amsterdam
Amsterdam - HDK 2U
+31 (0) 20 513 8069
IBM Amsterdam Hoofdkantoor on Google maps
Running the IBM Ubuntu Desktop (IBM intranet link)

From:
Gerrit Kahn <jahudi@gmx.net>
To:
Jasper Schroder/Netherlands/IBM@IBMNL
Date:
04/30/2009 11:14 AM
Subject:
Re: UVC

Hello Japser,

first of all thank you very much for your response.

I'm studying information systems and I'm currently participating in a seminar
about long term preservation. As part of this, I'm writing a short term paper
about the UVC.

Since the latest information about the UVC from the KB are from 2005, I'm very
interested in the current progress:
 - Has it been tested for other file formats than gif and jpeg?
 - Has it been used for archiving und running a whole program already or still
just for restoring files?
 - What about the performance? Especially when emulating whole programs...

Best regards,

Gerrit Kahn

Re: UVC

Von: Jasper Schroder <jasper@nl.ibm.com>
An: Gerrit Kahn <jahudi@gmx.net>
Datum: 2009-05-11 12:05

Hello Gerrit,

After the version by Jeffrey, we developed a C++ version. The goal of that development work was to check that with only the specification a new implementation could be developed on another platform (move from Win32 Java, to Linux C++).

The latest implementation was done to get more performance out of the system. A major improvement was achieved by doing the memory operations self coded and not rely on the structures like hashmaps, linked listed in C++. Most of the code is now plain C. Other improvements were based on:
using jumptables instead of case/select statements.
using OS semaphores for waiting in the commucation channel (instead of polling).
allow to compile without any debug information
smarter way of calculation with smaller numbers (calc with 16 bit on 32-bit machine to detect overflow)

Hope that this answer your question.

Jasper Schröder
IBM Nederland B.V
IBM Center for Advanced Studies Amsterdam
Amsterdam - HDK 2U
+31 (0) 20 513 8069
IBM Amsterdam Hoofdkantoor on Google maps
Running the IBM Ubuntu Desktop (IBM intranet link)

From:
Gerrit Kahn <jahudi@gmx.net>
To:
Jasper Schroder/Netherlands/IBM@IBMNL
Date:
05/11/2009 10:12 AM
Subject:
Re: UVC

Hello Jasper,

I have one more question about the performance of the UVC:

in his article "Development of a Universal Virtual Computer for long-term preservation of digital objects" Jeffrey van der Hoeven indicated the performance of the UVC with about 17.000 instructions per second (on IBM JRE 1.3). So its increase to 1 MIPS seems very considerable.

How did you achieve this enhancement?
Of course I dont want to know your secrets ;-) some keywords would be sufficient (e.g. do you still use Java or did you switch to another language?)

Best regards,

Gerrit Kahn

Abkürzungsverzeichnis

I/O Input/Output

JVM Java Virtual Machine

KB Koninklijke Bibliotheek

LDS Logical Data Scheme

LDV Logical Data View

LIFO Last-In-First-Out

LZA Langzeitarchivierung

MIPS Millionen Instruktionen pro Sekunde

STM Science-Technology-Medicine

UVC Universal Virtual Computer

Literaturverzeichnis

[1] *http://www.kb.nl/dnp/e-depot/factsandfigures-en.html*. 28.04.2009

[2] Borghoff, Uwe M. ; Rödig, Peter ; Scheffczyk, Jan: *Long-Term Preservation of Digital Documents: Principles and Practices*. Springer Verlag, 2006

[3] Hoeven, Jeffrey van d.: *UVC and emulation as preservation strategies*. 2005. – Presantation for workshop on electronic publishing, Lund, Zweden

[4] Hoeven, Jeffrey van d. ; Diessen, Raymond J. ; Meer, K. van d.: Development of a Universal Virtual Computer (UVC) for long-term preservation of digital objects. In: *Journal of Information Science* 31 (2005), Nr. 3, S. 196–208

[5] Hoeven, Jeffrey van d. ; Lohman, Bram ; Verdegem, Remco: Emulation for Digital Preservation in Practice: The Results. In: *The Journal of Digital Curation* 2 (2007), Nr. 2, S. 196–208

[6] Lorie, Raymond A.: *Long Term Preservation of Digital Information*. 2001

[7] Lorie, Raymond A.: The UVC: a method for preserving digital documents - proof of concept. In: *LTP reports series* 4 (2002)

[8] Lorie, Raymond A. ; Diessen, Raymond J.: UVC: A Universal Virtual Computer for Long-term Preservation of Digital Information. In: *IBM Research Report* (2005)

[9] Oltmans, Erik ; Diessen, Raymond J. ; Wijngaarden, Hilde van: Preservation Functionality in a Digital Archive. In: *4th ACM/IEEE-CS joint conference on Digital libraries* (2004), S. 279–286

[10] Oltmans, Erik ; Wijngaarden, Hilde van: Digital preservation in practice: the e-Depot at the Koninklijke Bibliotheek. In: *The Journal of Information and Knowledge Management Systems* 34 (2004), Nr. 1, S. 21–26

[11] Oltmans, Erik ; Wijngaarden, Hilde van: The KB e-Depot digital archiving policy. In: *Library Hi Tech* 24 (2006), Nr. 4, S. 604–613