Die LabVIEW-DIM Schnittstelle: Das Tor zur standardisierten Kommunikation zwischen LabVIEW und einer Vielfalt von Programmiersprachen und Betriebssystemen

Dietrich Beck, Holger Brand und Nikolaus Kurz
GSI-Darmstadt, DVEE, Planckstr. 1, D-64291 Darmstadt

Kurzfassung

DIM (Distributed Information Management) [1,2] ist eine Kommunikationsschnittstelle für verteilte Systeme und wurde bereits Anfang der 90er Jahre am CERN, dem europäischen Zentrum für Hochenergiephysik, in Genf entwickelt. DIM basiert auf einer Client-Server Architektur. Ein Server publiziert so genannte "Services", deren Typ ein elementarer Datentyp, ein Feld beliebiger Größe oder auch ein benutzerdefinierter Datentyp sein kann. Analog zu den "write items" bei OPC[1] kann ein DIM Server auch Kommandos entgegennehmen. Selbstverständlich ist DIM ereignisgesteuert, es wird nicht "gepollt".
Vom technischen Standpunkt gesehen gibt es zwei wesentliche Unterschiede zu OPC. Zum einen wurde DIM entwickelt, um die riesigen Datenströme der Hochenergiephysik handhaben zu können. Aufgrund seines schlankes Protokolls ist der Datendurchsatz bei DIM im Vergleich zum reinen TCP/IP nur um wenige Prozent reduziert [3]. Zum anderen kann DIM auf allen Plattformen eingesetzt werden, auf denen TCP/IP implementiert ist. So wird DIM neben den Betriebssystemen Windows und Linux auch auf VMS, Unix sowie den Echtzeit Plattformen OS9, LynxOs und VxWorks eingesetzt. Es werden C, C++, Java und Fortran unterstützt. Selbst für Mikrocontroller im Scheckkartenformat gibt es DIM Schnittstellen. Auch das kommerzielle SCADA System PVSSII von ETM hat eine DIM Schnittstelle. Die überfällige Anbindung von LabVIEW an DIM wurde im Rahmen dieser Arbeit vollzogen.

Abstract

DIM (Distributed Information Management) [1,2] is a communication interface for distributed systems and has been developed at CERN, the European centre for high energy physics, during the early nineties. DIM is based on a client-server architecture. A server publishes so-called "services", which can be of an elementary data type, an array of arbitrary size or a user defined data type. As OPC "write items", a DIM server can receive and process commands. Of course, DIM is fully event driven.
There are two main differences to OPC. First, DIM has been developed to transfer the huge amount of data produced in modern high energy physics experiments. Thanks to its light weight protocol the data throughput of DIM is reduced by a few percent only compared to pure TCP/IP [3]. Second, DIM can in principle be used on all platforms that have TCP/IP implemented. DIM is implemented for the operating systems Windows, Linux, VMS, Unix as well as the real-time platforms OS9, LynxOs and VxWorks. Supported languages are C, C++, Java und Fortran. There are DIM interfaces for credit card sized micro controllers as well as for the commercial SCADA product like PVSSII from ETM. This work provides the link from LabVIEW to DIM which is in use since May 2004.

Einleitung

Für den Austausch von Prozessdaten in der PC-basierten Automatisierungstechnik ist OPC eine einheitliche und hardwareherstellerunabhängige Softwareschnittstelle. Sie ermöglicht den Zugriff auf Prozessdaten beliebiger Hersteller in einheitlicher Art und Weise. OPC basiert auf dem DCOM (Distributed Component Object Model) Standard von Microsoft. Daraus resultiert, dass die Mehrzahl der angebotenen OPC-Produkte ein Betriebssystem von

[1] OLE for Process Control. OPC basiert auf dem DCOM Standard.

Microsoft voraussetzen – eine wichtige Einschränkung. Dagegen bietet DIM die Freiheit, Knoten mit nahezu beliebigen Architekturen und Betriebssystemen miteinander zu verbinden. Ein großer Nachteil von DIM war, dass es bis jetzt keine LabVIEW Anbindung gab.
DIM wird vom CERN gewartet und ist nach dem Modell der "GNU Public Licence" auch in Form von lauffähigen Bibliotheken (DLL) verfügbar. Der im Rahmen dieser Arbeit erstellte Kommunikationstreiber setzt auf dieser DLL auf. Entscheidend war hier die Speicherverwaltung, denn DIM arbeitet der besseren Performanz wegen direkt im Speicherbereich der jeweiligen Applikation. Dies ist in LabVIEW jedoch nicht erlaubt und so wurde eine zusätzliche DLL als globaler Speicher erstellt, auf den sowohl DIM als auch LabVIEW zugreifen kann. Die hier vorgestellte LabVIEW-DIM Schnittstelle ist seit Mai 2004 im Einsatz und erlaubt die Implementierung von DIM Servern und Clients in LabVIEW.

Aufgabenstellung

Das folgende C Programm stellt die vollständige Implementierung eines DIM Servers dar.

```
1 : #include <dis.h>
2 :
3 : int buffer[] = {0,1,2,3,4,5,6,7,8,9};
4 : int service_id;
5 :
6 : void execute_command(int *tag, char *cmnd_buffer, int *size)
7 : {
8 :         if (*tag == 1)
9 :         {
10:                 buffer[0]++;
11:                 dis_update_service(service_id);
12:         }
13: }
14:
15: void main()
16: {
17:         service_id = dis_add_service("SERV_BY_BUFFER","L:10",buffer,40,0,0);
18:         dis_add_cmnd ("SERV_CMND", "C", execute_command, 1);
19:         dis_start_serving("DIS_TEST");
20:         while (1) sleep(10);
21: }
```

Zunächst wird ein DIM Service "SERV_BY_BUFFER" deklariert, der aus einem Feld von 10 Zahlen besteht. Dabei wird nur die Adresse "buffer" auf dieses Feld an DIM übergeben (Zeile 17). Dann wird ein DIM Kommando "SERV_CMND" definiert. Dabei wird DIM die Adresse der aufzurufenden Funktion "execute_command" übergeben (Zeile 18). Anschließend wird der DIM Server "DIS_TEST" gestartet (Zeile 19). Empfängt der DIM Server nun das Kommando "SERV_CMND", so ruft DIM direkt die aufzurufende Funktion "execute_command" (Zeile 6) auf. In dieser Funktion wird das erste Element des Feldes "buffer" inkrementiert und der Service "SERV_BY_BUFFER" aktualisiert (Zeilen 10 und 11).
Eigentlich ist die LabVIEW Implementierung eines DIM Servers einfach, denn es brauchen nur die Funktionen der DIM DLL, wie z.B. "dis_add_service", über das "Call Library Function Node" VI der "Advanced" Funktionspalette benutzt zu werden. Die Schwierigkeit besteht in der direkten Adressierung des Speichers der Applikation durch DIM. Dies ist in LabVIEW offiziell nicht möglich. Zwar könnte man eine direkte Adressierung durch trickreiche Benutzung von Schieberegistern erreichen. Das ist jedoch nicht dokumentiert, umständlich und würde schlecht skalieren, wenn ein DIM Server viele DIM Services zur Verfügung stellt.

Lösung

Globales Memory

Als Lösung wurde der Weg über eine zusätzliche DLL gewählt, die globalen Speicher für Applikationen bereitstellt [4]. Dieser Speicher kann innerhalb einer Applikation frei genutzt werden und zwar entweder durch explizites Schreiben oder Lesen, oder auch durch direkte Adressierung über Zeiger. Abbildung 1 gibt einen Überblick der Funktionen, die von der *GlobalMem* DLL exportiert werden. Diese DLL basiert auf C++ Code. Ein Speicherbereich kann mit den Funktionen "CreateBuffer" und "DestroyBuffer" erzeugt bzw. freigegeben werden. "SetBuffer" und "GetBuffer" erlauben das Schreiben und Lesen eines Speicherbereiches, der mit "InitBuffer" initialisiert werden kann. Die Funktionen "GetBufferPointer", "GetBufferSize", "GetBufferName" sowie "GetBufferID" dienen zur sicheren Handhabung eines Speicherbereiches in einer Applikation. Der Vollständigkeit halber sei erwähnt, dass die *GlobalMem* nur globalen Speicher für eine Applikation bereitstellt, aber kein "shared Memory", auf das von verschiedenen Applikationen aus zugegriffen werden kann.

GlobalMemory dll Export
+GetProcCounter(:void):int
+CreateBuffer(ID:long *,Size:unsigned long,Name:char *):unsigned long
+DestroyBuffer(ID:long):long
+SetBuffer(ID:long,Offset:unsigned long,Length:unsigned long,Buffer:unsigned char *):unsigned long
+GetBuffer(ID:long,Offset:unsigned long,Length:unsigned long,Buffer:unsigned char *):unsigned long
+GetBufferPointer(ID:long):unsigned long
+InitBuffer(ID:long,Value:unsigned char):unsigned long
+GetBufferSize(ID:long):unsigned long
+GetBufferName(ID:long,Name:char *):long
+GetBufferID(Name:char *):long

Abbildung 1: Funktionen, die von der GlobalMem DLL exportiert werden.

LabVIEW-DIM Schnittstelle

Abbildung 2 zeigt die gewählte Architektur einer DIM Applikation.

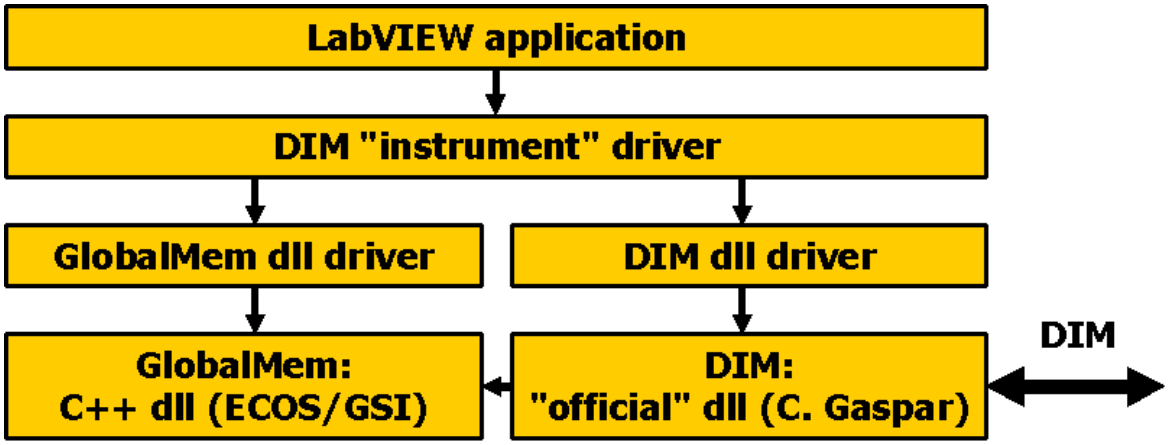

Abbildung 2: Architektur eines typischen DIM Servers in LabVIEW, Die DLLs basieren auf C/C++ Code und werden durch in LabVIEW geschriebene DLL-Treiber gekapselt. Applikationen benutzen den DIM "Instrumenten"-Treiber, der auf den DLL-Treibern basiert.

Die auf C/C++ Code basierenden DLLs werden durch in LabVIEW geschriebene DLL-Treiber gekapselt. Die Applikationen benutzen den DIM "Instrumenten"-Treiber, der auf den beiden DLL-Treibern basiert. Soll z.B. ein DIM Server mit einem DIM Service implementiert werden, so wird zunächst ein Speicherbereich in der *GlobalMem* DLL erzeugt. Anschließend kann über die DIM DLL der Service erzeugt werden, wobei die Adresse des zuvor erzeugten Speicherbereiches übergeben wird. Das Publizieren eines neuen Wertes erfolgt in zwei Schritten. Zum einen muss der neue Wert in den Speicherbereich der DLL geschrieben werden. Erst dann kann der DIM Service durch den Aufruf der entsprechenden Funktion der DIM DLL aktualisiert werden. Die DIM DLL greift dabei über die Speicheradresse direkt auf den Speicherbereich in der *GlobalMem* DLL zu. Diese Details sind jedoch im DIM "Instrumenten" Treiber gekapselt und bleiben dem LabVIEW Entwickler verborgen.

Beispiel

Abbildung 3 zeigt die Implementierung eines einfachen DIM Servers mit LabVIEW.

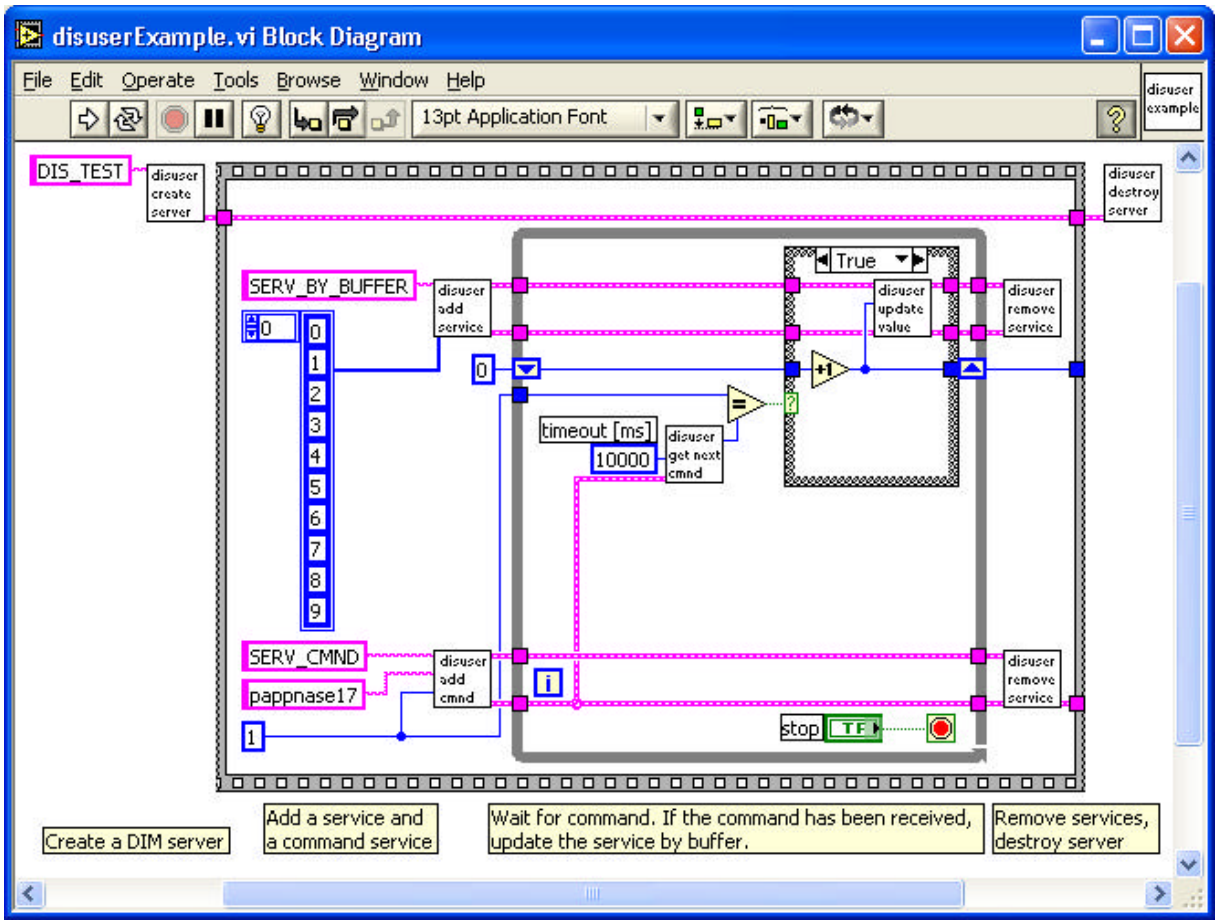

Abbildung 3: Beispiel für einen einfachen DIM Server mit LabVIEW, Erklärung im Text.

Dieses Beispiel stellt das LabVIEW Gegenstück zum C Programm dar, das oben im Text beschrieben wurde. Auch hier wird ein DIM Service "SERV_BY_BUFFER" publiziert und ein Kommando "SERV_CMND" definiert. Es gibt zwei wesentliche Unterschiede zum C Programm. Zum einen wird explizit auf ein neues Kommando gewartet. Ein direkter Aufruf eines LabVIEW VIs aus dem DIM "Instrumenten" Treiber ist (noch) nicht implementiert. Zum anderen werden hier alle Services und Kommandos ordentlich entfernt und der DIM Server zerstört. Dies ist notwendig, da andere Programmteile einer LabVIEW Applikation eventuell auch nach Beenden des DIM Servers weiter ausgeführt werden sollen.

Eigenschaften

In einer Applikation kann nur ein DIM Server oder ein DIM Client implementiert werden. Applikationen die sowohl DIM Server als auch DIM Client sind, sind unter Umständen instabil. Gemessen an Implementierungen mit C oder C++, erreicht die Transferrate der LabVIEW-DIM Schnittstelle etwa 50%. Bei einer Ethernet Verbindung mit 100 Mbit/s wurde ein Datendurchsatz von etwa 5 Mbyte/s gemessen. Dies hat zwei Ursachen. Zum einen müssen die Daten beim Schreiben bzw. Lesen der *GlobalMem* DLL kopiert werden. Zum anderen liegt es an der Repräsentation der Daten in LabVIEW. Da LabVIEW unabhängig vom Prozessor intern das Format der Motorola Architektur nutzt, muss für die LabVIEW-DIM Schnittstelle auf einem Intel Prozessor bei allen Daten die Reihenfolge der Bytes getauscht werden.

Applikationsbeispiel

Als Beispiel soll hier das Werkzeug *DomainConsole* vorgestellt werden [4]. *DomainConsole* wird als Prozess auf jedem Rechner eines verteilten Kontrollsystems lokal gestartet und hat zwei Aufgaben. Erstens ermittelt *DomainConsole* über eine Datenbankabfrage die Namen und Pfade der Prozesse, die auf dem jeweiligen Knoten benötigt werden und startet diese. Sollte ein Prozess unerwartet beendet werden ("abstürzen"), so wird er wieder neu gestartet. Hierauf soll jedoch nicht weiter eingegangen werden. Zweitens kann *DomainConsole* wichtige Informationen über Prozesse auf allen (!) beteiligten Rechnern einer verteilten Applikation darstellen. Die Verteilung der Prozessinformation erfolgt mit dem DIM Protokoll.
Formal gesehen ist *DomainConsole* ein DIM Client. Als DIM Server dient jeweils ein kleines schlankes C/C++ Programm *NodeMon* [4], das auf jedem der beteiligten Rechner ausgeführt werden muss. Dadurch können auch Prozesse auf Plattformen überwacht werden, auf denen LabVIEW nicht unterstützt wird.
Abbildung 4 zeigt auf der linken Seite die Benutzeroberfläche der *DomainConsole*.

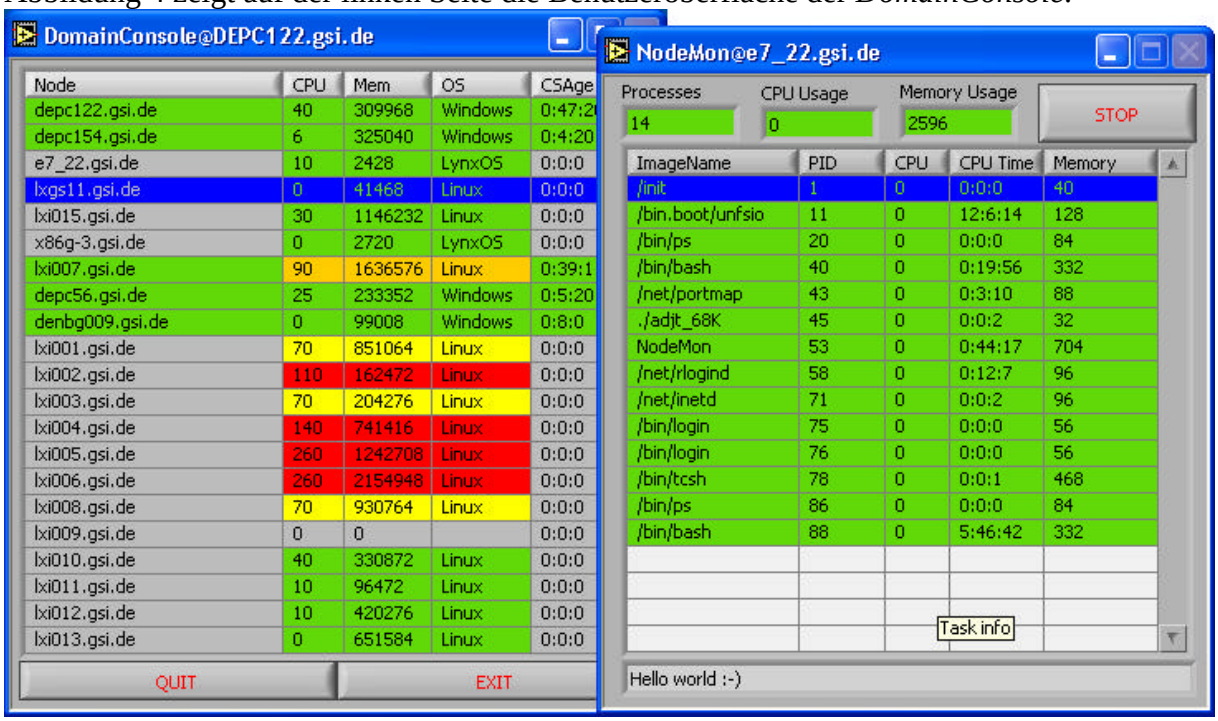

Abbildung 4: DomainConsole. Das Hauptfenster (links) gibt einen Überblick über die Rechner eines verteilten Systems. Durch Doppelklick wird ein Pop-up-Fenster geöffnet (rechts), das detaillierte Information zu den Prozessen eines Knotens gibt. Erklärung im Text.

Dargestellt werden die Namen der beteiligten Knoten, und jeweils die CPU-Last, der Speicherverbrauch und das Betriebssystem. Als spezielle Information wird noch die Laufzeit der Kontrollsystemsoftware *CS* [5,6] dargestellt, darauf soll hier aber nicht näher eingegangen werden. Detaillierte Information zu einem Knoten kann durch Doppelklick auf die entsprechende Zeile im in einem separaten Pop-Up-Fenster dargestellt werden. Als Beispiel ist ein solches Fenster auf der rechten Seite von Abbildung 4 für einen Knoten mit dem Betriebsystem LynxOS gezeigt. In der Kopfzeile werden die Zahl der Prozesse, die gesamte CPU-Last in % und der Speicherverbrauch in kbyte dargestellt. Die darunter angeordnete Tabelle gibt jeweils für jeden Prozess den Namen, die Prozess-ID, die CPU-Last die verbrauchte CPU-Zeit im Format hh:mm:ss sowie den Speicherverbrauch an. Die CPU-Last der Prozesse wird immer auf einen Prozessor normiert. Bei Multiprozessormaschinen kann sich daher rechnerisch ein Wert von deutlich über 100% ergeben, wie im linken Teil der Abbildung 4 gezeigt.
An der GSI ist LynxOS weit verbreitet, da eine Vielzahl der heute existierenden und zukünftigen GSI-Experimente zur Datenaufnahme das so genannte Multi Branch System (MBS) einsetzen [7,8], welches auf dem Echtzeit-Betriebsystem LynxOS aufbaut. Die hier gezeigte Applikation demonstriert, dass die neue LabVIEW-DIM Schnittstelle die Möglichkeit bietet, LabVIEW über das bewährte DIM-Protokoll an verschiedenste Hardware-Plattformen und Betriebsysteme anzuschließen.

Zusammenfassung

Die hier vorgestellte LabVIEW-DIM Schnittstelle bietet die Möglichkeit, LabVIEW über das standardisierte DIM-Protokoll mit allen Plattformen zu verbinden, die von DIM unterstützt werden. Damit kann LabVIEW auch in verteilten Systemen eingesetzt werden, in denen unterschiedlichste Hardware-Architekturen und Betriebsysteme zum Einsatz kommen. Einerseits ermöglicht dies den Einsatz von LabVIEW Applikation als "Frontends" in bereits bestehenden Systemen, die nicht auf LabVIEW basieren. Zum anderen kann LabVIEW nun auch genutzt werden, um verschiedenartigste Subsysteme über das DIM Protokoll zu einem großen verteilten System zusammenzuschließen. Selbstverständlich wurde die LabVIEW-DIM Schnittstelle auch auf Linux portiert. Die LabVIEW-DIM Schnittstelle ist an mehreren Experimenten der GSI seit Mai 2004 im Einsatz.

Acknowledgements

We would like to thank CERN, especially Clara Gaspar, for providing DIM. Your advice and suggestions have been extremely helpful.

Literatur

[1] C. Gaspar und M. Dönszelmann, Konferenzbeitrag *IEEE Eight Conference REAL TIME '93 on Computer Applications in Nuclear, Particle and Plasma Physics*, Vancouver, Kanada, 8.-11. Juni 1993.
[2] http://dim.web.cern.ch/dim/.
[3] C. Gaspar et al., Konferenzbeitrag *International Conference on Computing in High Energy and Nuclear Physics*, Padova, Italien, 1.-11. Februar 2000.
[4] http://www-w2k.gsi.de/controls/.
[5] D. Beck et al., Nucl. Instr. Meth. A 527 (2004) 567-579.
[6] D. Beck et al., Proc. "Virtuelle Instrumente in der Praxis 2004", VIP 2004, Fürstenfeldbruck, Germany, Editors R. Jamal and H. Jaschinski, ISBN 3-7785-2932-3, 250-254.
[7] H. G. Essel et al., IEEE Trans. of Nucl. Science, Vol.43, No.1 (1996) 132-135.
[8] H. G. Essel und N. Kurz, IEEE Trans. of Nucl. Science, Vol.47, No.2 (2000) 337-339.