inkl. CD!

Ausgabe 2.10 Februar | März www.phpmagazin.de S&S

PHPmagazin

Deutschland 9,80 € Österreich 10,80 € | Schweiz 19,20 sFr
Niederlande 11,25 € | Luxemburg 11,25 €

PostgreSQL
Schnell mal den eigenen Cluster aufgesetzt

Formular-Styling
Die Dekorierer der Zend_Form-Komponente

Flexibel mit Doctrine
Der ORM-Mapper und das Zend Framework

Charts mit PHP
Interaktive Diagramme dank Open Flash Chart 2

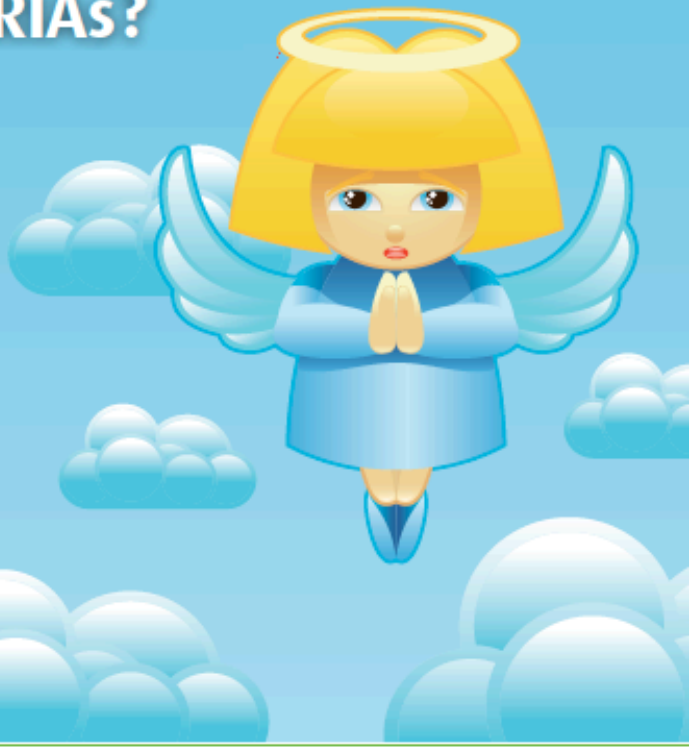
web2test (Trial)
Die automatisierte Capture & Replay-Testsoftware für das funktionale, GUI-basierte Testen webbasierter Anwendungen und Portale.

Open Limbas
Webbasiertes Framework zur Umsetzung von Unternehmenslösungen. Inklusive Basisdienste und Module für CRM, CM, Ticketsystem, Workflow und Groupware.

CruiseControl
Das CI-Tool und Framework für individuelle Build-Prozesse.

Zusätzlich die aktuellen **CM-Systeme**, **PHP-Frameworks** und **Quellcodes** zu den Artikeln auf CD!

Silverlight 3
Alternative zu Flash-RIAs?



UTF-8 für alle
Website-Encoding einfach anpassen

Einfach mehr MySQL
Schaufeln Logik in die Datenbank: Foreign Keys, Trigger und Stored Procedures

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

Außerdem mit auf der CD
PostgreSQL, Doctrine, Zend Framework, CodeIgniter, WordPress, Serendipity, Drupal, TYPO3, ImpressCMS, Open Flash Chart, alle Codes zu den PHP-Artikeln und vieles mehr!

4 195649 609809 02

LIMBAS und die Wahl der Datenbank

Kontaktfreudig

LIMBAS als Datenbank-Frontend hat bislang nur die Adabas-D- bzw. SAP-DB- oder MaxDB-Datenbank unterstützt. Seit Version 2.0 bietet LIMBAS die Möglichkeit, sich die Datenbank auszusuchen. Voraussetzung dafür ist ein ODBC-Treiber und eine Reihe von datenbankspezifischen Anpassungen. Dieser Artikel soll anhand von MaxDB, PostgreSQL und Ingres die Unterschiede der Datenbanken beleuchten sowie eine Vorlage für die Migration anderer Datenbanken in LIMBAS bieten.

▶ von Axel Westhagen

Die Anbindung der Datenbank erfolgt in PHP über die ODBC-Erweiterung. Dafür kann der jeweilige ODBC-Treiber direkt in PHP einkompiliert werden oder man nutzt unixODBC als Zwischenschicht. unixODBC hat den Vorteil, dass es über PHP-Module leicht installiert werden kann und nur noch seine Konfigurationsdateien angepasst werden müssen. Der gemessene Geschwindigkeitsverlust gegenüber der direkten Verbindung kann bei LIMBAS als minimal eingestuft werden und wirkt sich erst bei hohen Lasten negativ aus. Ein späterer Wechsel der Verbindungsart ist jedoch ohne Probleme möglich. Hier die Treiberdefinition von unixODBC in der *odbcinst.ini*:

```
[PSQL]
Description=PostgreSQL
Driver=/usr/lib/psqlodbcw.so
Setup=/usr/lib/unixODBC/libodbcpsqlS.so
```

Einbindung

Welche Datenbank LIMBAS nutzen soll, ist in der Datei */dependent/inc/include_db.lib* definiert. Diese Datei beinhaltet nicht nur die Zugangsinformationen zur jeweiligen Datenbank, sondern definiert auch den Pfad zu den Integrationsdateien. Der Pfad setzt sich aus der Globalen *$DBA['DB']* und dem Verzeichnispfad */lib/db* zusammen. Pro Datenbank gibt es jeweils zwei Dateien. Die erste ist für die Verbindung der Datenbank, für die allgemeine Beschreibung der Datentypen und für ein paar grundsätzliche Funktionen zuständig. Die zweite behandelt Admin-Funktionen wie Trigger, Foreign Keys oder Indizes und steht ausschließlich LIMBAS-Benutzern zur Verfügung, die Admin-Rechte besitzen. Für PostgreSQL wären das die */lib/db/postgres.lib* bzw. */lib/db/postgres_admin.lib*.

Für das Einbinden einer neuen Datenbank wie MySQL müssen wir nur die zwei Integrationsdateien einer vorhan-

denen Datenbank kopieren und in *db_mysql.lib* und *db_mysql_admin.lib* umbenennen sowie die Globale *$DB['DB']* in der *include_db.lib* anpassen. Listing 1 zeigt den Inhalt der */inc/include_db.lib*.

LIMBAS Core

Für den Fall von MaxDB, PostgreSQL und Ingres ist außer der Anpassung der zwei Dateien kein Eingriff in den LIMBAS-Core mehr notwendig. Für weitere Datenbanken ergeben sich aber möglicherweise notwendige Erweiterungen. Ein Beispiel zeigt die MySQL. LIMBAS geht aktuell davon aus, dass mehrere *TIMESTAMP*-Felder mit den *DEFAULT CURRENT_TIMESTAMP, NOW()* etc. Definitionen pro Tabelle möglich sind, was MySQL so nicht unterstützt (es wird nur die erste Definition nicht ignoriert). Eine Änderung des LIMBAS-Cores wäre somit notwendig. Eine schon integrierte spezifische Anpassung ist hingegen für die Ingres-Datenbank bereits vorhanden, die den Datentyp *BOOLEAN* nicht kennt. Der LIMBAS-Core musste in diesem Fall die Möglichkeit bieten, anstatt *BOOLEAN* ein Zahlenfeld mit den Werten *0* und *1* zu nutzen. Eine Abwägung von Nutzen und Aufwand sollte also vor jeder Entscheidung für eine Integration einer Datenbank stehen.

Anpassung

Nun wenden wir uns der eigentlichen Integration zu. In unserem Beispiel beschreiben wir die Einbindung der PostgreSQL-Datenbank. Dafür schauen wir uns zuerst die allgemeine Integrationsdatei *db_postgres.lib* etwas genauer an. Im ersten Teil der Datei finden wir eine Reihe von Konstanten. Diese Konstanten sind für die unterschiedliche Behandlung der Datentypen und der SQL-Interpretation zuständig. Dabei werden die Konstanten durch ihren Namensanfang nach ihrer Aufgabe unterschieden.

*LMB_DBTYPE_**: Diese Konstanten definieren den entsprechenden Datentyp der Datenbank. Als Referenz wurden aus historischen Gründen die Datentypen der MaxDB genommen: *define("LMB_DBTYPE_LONG","TEXT");* bedeutet, dass der entsprechende Datentyp für *LONG* für die PostgreSQL-Datenbank *TEXT* ist.

*LMB_DBDEF_**: Hiermit werden die Default-Werte definiert: *define("LMB_DBDEF_TIME","CURRENT_TIME");*, was soviel bedeutet wie „nutze als Default-Wert für den Zeitwert *CURRENT_TIME*."

*LMB_DBRETYPE_**: Diese Konstante wird genau umgekehrt wie *LMB_DBTYPE_** definiert. Hier ist ihr Name die Definition und ihr Wert die Referenz: *define("LMB_DBRETYPE_TEXT","LONG");*. Diese Konstante wird für die Auflösung der Datentypen aus einem Export der Datenbank benötigt. Somit ist es möglich, in LIMBAS einen allgemeingültigen Export zu generieren, der wieder in eine andere LIMBAS-Installation mit einer ganz anderen Datenbank importiert werden kann.

*LMB_DBREDEF_**: Auch für *LMB_DBDEF_** gibt es eine Konstante, die durch ihren Namen die Definition des Default-Werts beschreibt: *define("LMB_DBREDEF_NOW()","DEFAULT TIMESTAMP");*. In diesem Fall bringt PostgreSQL als Default-Wert für einen *TIMESTAMP* den Wert *NOW()*.

*LMB_DBFUNC_**: Als Letztes werden funktionale Konstanten beschrieben: *define("LMB_DBFUNC_YEAR","EXTRACT(YEAR FROM ");*. Die SQL-Abfrage nach dem Jahr eines Felds des Typs *TIMESTAMP* würde für PostgreSQL *SELECT EXTRACT(YEAR FROM ZEITWERT)* lauten. Für MaxDB und Ingres hingegen genügt ein einfaches *SELECT YEAR(ZEITWERT)*. Ein kompletter SQL-Aufruf zum Erstellen einer Tabelle könnte somit folgendermaßen aussehen:

```
CREATE TABLE TEST (ID [LMB_DBTYPE_INTEGER]
 [LMB_DBFUNC_UNIQUE], NAME [LMB_DBTYPE_VARCHAR]
  (150), ERSTELLT [LMB_DBTYPE_TIMESTAMP] DEFAULT
   [LMB_DBDEF_TIMESTAMP] PRIMARY KEY
    [LMB_DBFUNC_PRIMARY_KEY] ID)
```

Sind die Unterschiede der SQL-Aufrufe zu unterschiedlich und die Anforderung durch einfache Konstanten nicht mehr umsetzbar, werden Funktionen eingesetzt, die im zweiten Teil der *db_postgres.lib* definiert sind: Die erste Funktion ist die Funktion *dbq_0*. Sie definiert die eigentliche ODBC-Verbindung. Optional ist der Gebrauch von *CURSOR*-Einstellungen.

Die Funktionen *dbf_1* und *dbf_2* konvertieren das Format des Datums für und von der Datenbank. In der Regel haben Datumsfelder ohne besondere lokale Einstellungen der Datenbank das Format *YYYY-MM-DD*. Zu beachten ist, dass sich ausschließlich LIMBAS um die Konvertierung des Datums kümmert und nicht von der Datenbank behandelt werden sollte. Ein gutes Beispiel ist das standardmäßige Lokalisierungsverlangen von PostgreSQL. Bei der Installation der Datenbank sollte man deshalb auch die Option *initdb –locale=C*, die einen nicht lokalisierten PostgreSQL-Datenbankcluster erzeugt, nutzen. Die Funktionen *dbf_6* und *dbf_7* kümmern sich um die Maskierung von Sonderzeichen. Meistens genügt eine Behandlung des Hochkommas *(')* durch Voranstellung des selbigen *('')*.

PostgreSQL akzeptiert auch einen Backslash *(\)*, was mit der PHP-Funktion *addslashes()* einfach bewerkstelligt werden kann. Durch die Sonderstellung des Fragezeichens *(?)* können damit aber auch Probleme auftreten. Der SQL-Aufruf *UPDATE TEST SET TEXTFELD = 'der \'Hund?\' Hat Flöhe'*, was soviel be-

LISTING 1

/inc/include_db.lib

```
$DBA["DB"] = 'postgres'; /* maxdb76 | postgres |
                                        ingres */
$DBA["DBCUSER"] = 'CONTROLUSER';
                    /* DB control user (only MaxDB)*/
$DBA["DBCPASS"] = 'CONTROLPASS';
                /* DB control password (only MaxDB)*/
$DBA["DBUSER"] = "limbasuser"; /* DB username */
$DBA["DBPASS"] = "limbaspass"; /* DB password */
$DBA["DBNAME"] = "limbas"; /* DB instance
                                        name */
$DBA["DBSCHEMA"] = "public"; /* DB schema */
$DBA["DBHOST"] = "localhost"; /* DB hostname
                                        or IP */
$DBA["LMHOST"] = "localhost";
                      /* LIMBAS hostname or IP */
$DBA["DBPATH"] = "";
                /* Path to database (only MaxDB)*/
$DBA["LMPATH"] = "/usr/local/httpd/htdocs/
        limbas_2/dependent"; /* Path to LIMBAS */
$DBA["ODBCDRIVER"] = "PSQL";
                              /* unixODBC Driver */
require_once("{$DBA['LMPATH']}/
                    lib/db/db_{$DBA['DB']}.lib");
```

deutet wie „der 'Hund?' hat Flöhe“ verursacht in PostgreSQL folgenden Fehler: *The # of binded parameters < the # of parameter markers*.

Besser ist der Aufruf: *UPDATE TEST SET TEXTFELD = 'der ''Hund?'' Hat Flöhe'* – so wird der Parametermarker '*?*' als Text behandelt. Da sich LIMBAS um die Maskierung von SQL-Aufrufen kümmert und nicht zuletzt, da das *magic_quotes*-Feature in kommenden PHP-Versionen entfernt werden soll, sollte die serverseitige Maskierung von PHP via *magic_quotes* ausgeschaltet sein.

Weit größere Unterschiede finden sich in der Behandlung von *LONG*-Feldern. Wie bei den meisten Datenbanken wurden *LONG*- oder *BLOB*-Felder erst im Nachhinein implementiert. Dadurch entstanden einige Einschränkungen, was die Nutzung dieses Feldtyps anbelangt.

LISTING 2

```
SELECT c.oid,
  n.nspname,
  c.relname AS TABLENAME,
  c.relhasindex, c.relkind, c.relchecks, c.relhasrules,
                          c.relhasoids, c.reltablespace,
  a.attname AS COLUMNNAME,
   pg_catalog.format_type(a.atttypid,
                                  a.atttypmod) AS TYPE,
  (SELECT substring(pg_catalog.pg_get_expr
                           (d.adbin, d.adrelid) for 128)
    FROM pg_catalog.pg_attrdef AS d
    WHERE d.adrelid = a.attrelid AND d.adnum =
                 a.attnum AND a.atthasdef) AS attrdef,
   a.attnotnull, a.attnum,
   i.indexrelid,
   i.indrelid,
   i.indisunique, i.indisprimary, i.indisclustered,
                          i.indisvalid AS INDEX_VALID,
                          i.indisready AS INDEX_USED,
   c1.relname AS INDEXNAME
FROM pg_catalog.pg_class AS c
  LEFT JOIN pg_catalog.pg_namespace AS n ON
                                 n.oid = c.relnamespace
  LEFT JOIN pg_catalog.pg_index
                                 AS i ON i.indrelid = c.oid
  LEFT JOIN pg_attribute AS a ON a.attrelid =
                                              i.indexrelid
  LEFT JOIN pg_catalog.pg_class
                                 AS c1 ON i.indexrelid = c1.oid
WHERE c.relhasindex = true
  AND c.relname not like '%pg_%'
  AND a.attnum > 0 AND NOT a.attisdropped
  AND pg_catalog.pg_table_is_visible(c.oid)
```

MaxDB erlaubt beispielsweise keine *LONG*-Felder im Zusammenhang mit *DISTINCT* oder *ORDER BY*-Aufrufen. Ebenso kann man *LONG*-Felder nicht per SQL-Aufrufe durchsuchen. Ein *UPDATE* eines *LONG*-Felds ist nur über den Umweg eines *odbc_prepare*-Statements möglich. Da LIMBAS historisch über die Adabas D bzw. SAP DB entstanden ist, existiert für diese Felder eine eigene Indizierungsfunktion, die das Durchsuchen dieser Felder wieder möglich macht.

Bei PostgreSQL gestaltet sich das wiederum deutlich einfacher. *TEXT*-Felder können wie normale *VARCHAR*-Felder behandelt werden. Trotzdem kann auch hier die Indizierungsfunktion von LIMBAS genutzt werden, falls es gewünscht ist. Je nach Größe und Menge der Inhalte kann die eine oder andere Weise ihre Vorteile haben. Die gleiche Indizierungsfunktion nutzt LIMBAS übrigens auch, um das integrierte Dateisystem zu verschlagworten.

Ingres hat ähnlich zu MaxDB einige Einschränkungen, was den Datentyp *LONG* (in Ingres *LONG VARCHAR*) anbelangt, bietet aber dafür extrem lange *VARCHAR*-Felder (bis zu 32 000 Zeichen), die keiner Einschränkung unterliegen. Zusätzliche Problematik bei Ingres ist, dass *LONG VARCHAR*-Felder nicht per ODBC-Cursor angesteuert werden können. Eine differenzierte Behandlung ist so für eine Nutzung notwendig. Nicht zu unterschlagen ist auch die Nutzung von *FLOAT*-Feldern. Sie sind, wie der Name schon sagt, Fließkommazahlen und somit „ungenau“. Im Allgemeinen ist dieser Feldtyp also sicher nicht für Währungen geeignet und sollte durch den Datentyp *NUMERIC* ersetzt werden.

Interessanterweise konnte diese Ungenauigkeit bei MaxDB so nicht beobachtet werden, wo hingegen in PostgreSQL die interessantesten Rundungen entstanden sind. Welche Feldtypen intern verwendet werden sollen, kann direkt in der LIMBAS-Systemtabelle *lmb_field_types* oder über die LIMBAS-Administrationstools per Webbrowser angepasst werden.

Die letzte Funktion ist für die Groß-/Kleinschreibung der Tabellen oder Feldnamen in der Datenbank zuständig. Obwohl fast alle Datenbanken das frei definierbar lassen, aber manche eine Großschreibung bevorzugen, sind in PostgreSQL, MySQL und Ingres die Namen standardmäßig klein geschrieben. Diese Funktion erspart dem Administrator notwendige Anpassungen an der Datenbank.

Nach dieser kleinen Aufwärmung kommen wir zur zweiten Integrationsdatei *db_postgres_admin.lib*, die wir allerdings nur oberflächlich behandeln werden. Diese Datei beinhaltet alle Funktionen, die für die Admin-Werkzeuge, wie Trigger verwalten, Foreign Keys erstellen, Backup erzeugen usw., notwendig sind. Wo in der Definition der Feldtypen oder der Umsetzung von SQL-Standard noch einigermaßen Konformität herrschte, ist dieser Bereich deutlich spezifischer gehalten. Ein gutes Beispiel ist die Abfrage nach allen Indizes inklusive ihrer Verweise, Primary Keys und Informationen, die zum Nachbau in einer anderen Datenbank notwendig sind. Für MaxDB genügt in diesem Fall ein einfacher SQL-Aufruf: *SELECT * FROM DOMAIN.INDEXCOLUMNS*, wo hingegen für PostgreSQL geradezu ein Feuerwerk einer Systemtabellenabfrage notwendig ist (Listing 2). Insgesamt werden in dieser Datei Foreign Keys, Primary Keys, Indizes, Trigger, Views, Tables, Columns, Stored Procedures und Backups behandelt.

Geschwindigkeit

Ein Vergleich der Geschwindigkeit von LIMBAS in Verbindung der eingesetzten Datenbanken ist ein Auf und Ab der Gefühle. Eine Festlegung, welches die schnellere Alternative ist, wäre in meinen Augen schwer zu erklären. Jede Datenbank hat in bestimmten Situationen ein ganz eigenes Leistungsverhalten. Es ist ein Zusammenspiel von Komplexität des SQL-Aufrufs und Größe der Datenbank, was sehr unterschiedliche Ergebnisse liefern kann. Ebenso tragen die von Haus aus notwendigen Hardwareressourcen stark zur Leistungsfähigkeit bei. Im Einsatz für LIMBAS konnten wir feststellen, dass PostgreSQL für kleine oder mittlere Installationen nicht zu schlagen ist. MaxDB und Ingres gewin-

nen bei hohen Lasten oder großen Datenmengen die Oberhand.

Backup

Je nach Einsatzgebiet sind mehr oder weniger professionelle Backup-Möglichkeiten gewünscht. Hierfür stehen eine Reihe unterschiedlicher Backup-Konzepte der einzelnen Datenbanken bereit. Die MaxDB unterstützt ein Voll- und inkrementelles Backup im laufenden Betrieb und auch verschiedene Medien wie auch Tapes. Das Backup und das Wiedereinspielen gehen ebenso einfach wie schnell. Zusätzlich sichern Transaktions-Logs alle Änderungen zwischen den Backups.

Die PostgreSQL bietet zur Datensicherung ein SQL-Dump, das die Daten konsistent in eine Textdatei schreibt. Während des Dumps werden die normalen Operationen nicht blockiert, womit ein Dump im laufenden Betrieb ermöglicht wird. Der Dump gibt Objekte allerdings nicht unbedingt in der Reihenfolge aus, in der sie in der Datenbank vorhanden sind, was zu Problemen beim wieder Einspielen führen kann. Eine weitere Möglichkeit (und das gilt für alle Datenbanken) ist das einfache Sichern auf Dateisystemebene. Diese Möglichkeit speichert zwar wirklich alles, setzt aber voraus, dass für eine konsistente Sicherung die Datenbank während des Backups gestoppt wird.

Ingres hat ebenfalls die Möglichkeit eines Voll-Backups im laufenden Betrieb und kann zusätzlich über seine Journaling-Dateien ein inkrementelles Backup erzeugen sowie ältere Zustände wiederherstellen. Zusätzlich bietet auch sie die Möglichkeit eines SQL-Dumps.

Fazit

Die in LIMBAS integrierten Datenbanken MaxDB, PostgreSQL und Ingres sind alle eine gute Wahl. Das betrifft die Leistungsfähigkeit sowie Ihre Kompatibilität zu LIMBAS. MaxDB zeichnen seine professionellen Managementtools, sowie ihr ausgezeichnetes Backup aus. Ebenso war sie im bisherigen Einsatz mit LIMBAS immer sehr stabil und nur selten aus der Fassung zu bringen. Mankos sind die magere *LONG*-Unterstützung und die ungewisse Zukunft der Datenbank.

PostgreSQL hat seinen Vorteil klar in der schlankeren Struktur. Das zahlt sich in einer einfachen Installation sowie einer hohen Geschwindigkeit auch bei schwachen Rechnern aus. Größtes Manko aus meiner Sicht ist das Backup-Konzept, wenn man mal Fremdprogramme außer Acht lässt.

Die Ingres-Datenbank behauptet sich in ihrem professionellen Support sowie der guten Performance im Lastbetrieb. Sie ist zwar bei kleineren Installationen etwas langsamer als PostgreSQL, entfaltet aber ihre volle Leistungsfähigkeit bei höherer Last und größeren Datenmengen, was sie ohne gravierenden Leistungseinbruch wegsteckt.

Es bleibt also dem Nutzer zu entscheiden, welche Datenbank für seine Zwecke die bessere Wahl ist. Durch die grundsätzlich einfache Portierung der Datenbank in LIMBAS bleibt es aber immer frei, die Datenbank zu wechseln.

Links & Literatur

[1] LIMBAS: http://www.limbas.org/

[2] MaxDB: http://maxdb.sap.com/

[3] PostgreSQL: http://www.postgresql.org/

[4] Ingres Database: http://www.ingres.com/products/ingres-database.php

Axel Westhagen

Bereits seit seinem Physikstudium arbeitet Axel Westhagen (axel.westhagen@limbas.com) als Softwareentwickler und Berater. Schnell wurde ihm klar, dass die Anforderungen an die Verwaltungsprojekte auf ähnlichen Anforderungen basieren und so begann er mit dem Aufbau eines Anwendungsframeworks für diese Lösungen. Durch die Kundenprojekte reifte das Projekt immer weiter und wurde schließlich als Open-Source-Softwarelösung LIMBAS veröffentlicht. Gemeinsam mit seinen Partnern hat Axel Westhagen 2006 die LIMBAS GmbH gegründet, um für das Open-Source-Projekt professionelle Dienstleistungen anbieten zu können. Sowohl die Plattformunabhängigkeit wie auch die ganz klare Fokussierung auf webbasierte Lösungen sind die wesentlichen Merkmale des Produkts LIMBAS. In Zukunft soll sich die Plattform zu einer Open-Source-basierten Alternative im Bereich webbasierter Anwendungsumgebungen entwickeln und individuelle Kundenlösungen unterstützen.