Qualitäts-Managementsystem für Finite-Elemente-Analysen

Die Bundesanstalt für Geowissenschaften und Rohstoffe (BGR) [5], Hannover, ist als Fachbehörde des Bundesministeriums für Wirtschaft und Arbeit die zentrale wissenschaftlich-technische Institution zur Beratung der Bundesregierung in allen georelevanten Fragestellungen. Die Abteilung B2 ist in der BGR die Organisationseinheit für den Bereich „Geotechnische Sicherheit/Endlagerung" und befasst sich hauptsächlich mit der Endlagerung radioaktiver Abfälle. Neben den Endlagerarbeiten gehören die geotechnische Bewertung untertägiger Bauwerke (z.B. Salzkavernen) sowie Georisiken (z.B. Rutschungen, Erdbeben) zu den Tätigkeiten dieser Abteilung, die auch auf internationaler Ebene stattfinden.

Im Referat B2.6 (Modellberechnungen, Numerische Verfahren) der Abteilung B2 werden in einem heterogenen Netzwerk (PCs, Unix/Linux Workstations, Linux-Server, Sun-Server) Finite-Elemente-Analysen durchgeführt. Hierzu erstellen Wissenschaftler auf dem Desktop (PC/Workstation) Berechnungsmodelle (Pre-Processing), die sie anschließend auf einem leistungsfähigen Compute-Server analysieren lassen. Die Ergebnisse einer solchen Berechnung werden auf dem Desktop visualisiert und ausgewertet (Post-Processing). Schließlich archivieren die Benutzer sämtliche Daten der Modellberechnungen auf einem zentralen File-Server.

Bisher mussten die Benutzer sämtliche Programmaufrufe und Datentransfers von Hand ausführen und die Berechnungen selbst starten und überwachen. Um diese Abläufe zu automatisieren und potenzielle Fehlerquellen zu minimieren, sollte eine Software entwickelt werden, die auf der vorhandenen Infrastruktur sämtliche Remote-Logins sowie manuelle File-Transfers überflüssig macht und komplett vom Desktop zu bedienen ist. Zusätzlich sollte eine Benutzerverwaltung mit einem mehrstufigen Berechtigungsmodell implementiert werden.

Als generelles Konzept wurde dafür eine zweistufige Client-Server-Architektur angestrebt, mit einem zentralen Server, der sämtliche Daten vorhält, Benutzer verwaltet sowie Berechnungen steuert und überwacht.

Das Projekt sollte in akzeptabler Zeit mit möglichst geringem Kostenaufwand realisiert werden. Zusätzlich sollte es auf einer heterogenen Infrastruktur möglichst wenig Portabilitätsprobleme mit sich bringen und sowohl unter Unix/Linux als auch unter Windows eine grafische Benutzeroberfläche anbieten. Unter Berücksichtigung dieser Randbedingungen wurde als Programmiersprache Java ausgewählt.

Um den Programmieraufwand und damit die Kosten zu minimieren, wurde nach Modulen recherchiert, die in diesem Projekt genutzt werden können. Als Datenbank-Management-System wurde PointBase von DataMirror [6] ausgewählt, da es SQL beherrscht, eine JDBC-Schnittstelle besitzt, keine langwierige Installations- oder Administrationarbeit mit sich bringt und gut lokal oder auf dem zentralen Server läuft.

Als eine Art Baukasten für Client-Server-Lösungen wurde der EsprIT-Server von TnTsoft azusgewählt. Dieser sollte als Basis dienen und bietet Unterstützung für die Persistenz-Schicht DBOSuite, ebenfalls von TnTsoft, mit der ein Zugriff auf Datenbanken einfach zu realisieren ist.

Am Anfang der Programmierarbeit stand ein SQL Script, das die PointBase-Datenbank aufbaut. Mithilfe des DBOCompilers aus der DBOSuite wurde dann auf Basis dieser Datenbank für jede Tabelle eine Java-Zugriffsklasse (DBObject) im Quellcode erstellt. Der Vorteil dabei liegt darin, dass die Schnittstelle zur Datenbank fehlerfrei ist und das zeitaufwendige Erstellen eigener DataAccess-Objekte entfällt. Nach Modifikationen beziehungsweise Erweiterungen des Datenmodells wird der DBOCompiler erneut gestartet, wodurch sämtliche Änderungen im Quellcode der DBObjects vorgenommen werden.

Die generierten DBObjects kommunizieren mit der Datenbank in Verbindung mit dem EsprIT-Server auch von Clients aus, ohne die Datenbank direkt zu kontaktieren. Dazu verbindet sich der Client mit dem EsprIT Server, der als Tunnel zur Datenbank agiert und Netzwerk- und Datenbank-Last reduziert. Da die DBOSuite ein eigenes konfigurierbares Connection-Pooling enthält, wird außerdem ein Teil der Lizenzkosten für die Datenbank eingespart.

Im zweiten Schritt wurden erste Agenten programmiert, welche die Schnittstelle zwischen einem Client und dem EsprIT-Server bilden. Diese enthalten sowohl den Programmcode des Clients als auch den des Servers. Dadurch steht der Quellcode kompakt in einer einzigen Klasse. Im Ergebnis ist der Code einfach wartbar und man hat eine gute Übersicht über die Abläufe. Den Transport und das Handling dieser Agenten und der enthaltenen Daten übernimmt der EsprIT-Server. Synchrone oder asynchrone Ausführung, mit oder ohne Rückmeldung, oder die Rückmeldung des Ergebnisses als Response oder Message … all das kann man in seinen eigenen Agenten konfigurieren.

Als Drittes wurde ein erstes einfaches Client-GUI für Tests erstellt, das diese Agenten an den EsprIT-Server sendet. Dabei funktionierte der Server auf Anhieb und arbeitete die Agenten sauber und robust ab. Dadurch war der Grundstein für die Realisierung des Projektes gelegt.

Die nächste Aufgabe bestand darin, einen spezialisierten EsprIT-Server, den Zentralknoten-Server (ZK-Server, Abb.6) zu erstellen. Besondere Merkmale sind hier neben der Konfiguration eines eigenen *UserAuthenticator* der *ProjectManager* und der *PermissionManager*. Diese regeln die Zugriffe auf die Datenbank und ins Dateisystem. Auf diese Weise ist sichergestellt, dass die Aktionen auf dem ZK-Server geprüft und protokolliert ablaufen. Die ZK-Agenten können auf Serverseite direkt den *ProjectManager* ansprechen und z.B. für einen bestimmten Berechnungslauf die Dokumentation abrufen. Stammt der Agent von einem nicht autorisierten Benutzer, so wird die Anfrage mit einer Fehlermeldung abgelehnt.

Um auch große Datenmengen zwischen dem Zentralknoten und den Clients austauschen zu können, wurde der Transfermechanismus des EsprIT-Servers getestet. Dazu erstellt man eine *FileTransferList*, die detailliert konfigurierbar ist. Die Angaben von Quelle und Ziel, ASCII oder binär, mit oder ohne Erzeugung fehlender Ziel-Verzeichnisse und sogar ein gemischter Transfer von Up- und Downloads sind machbar. Für diesen Austausch nutzt der Server eine separate Verbindung (Transfer-Kanal), um die Serverlast gering zu halten. Für dieses besondere Feature war ser-

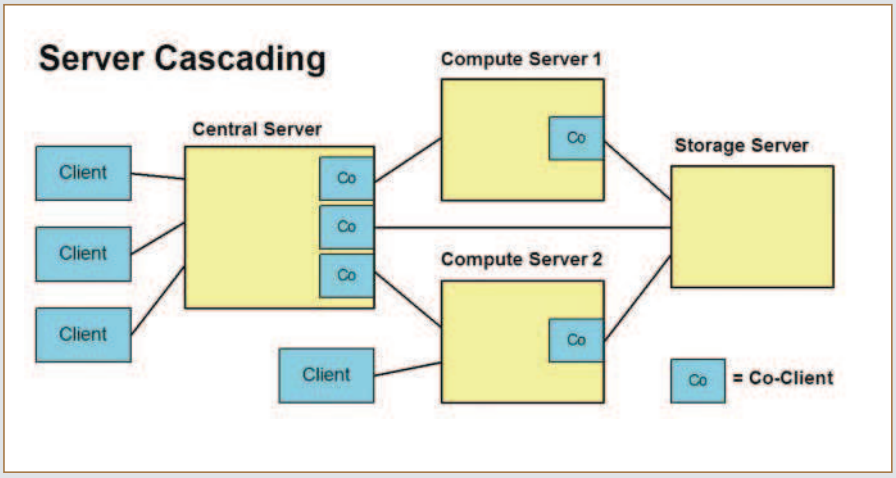

Abb. 5: Genereller Aufbau des Inca-Systems

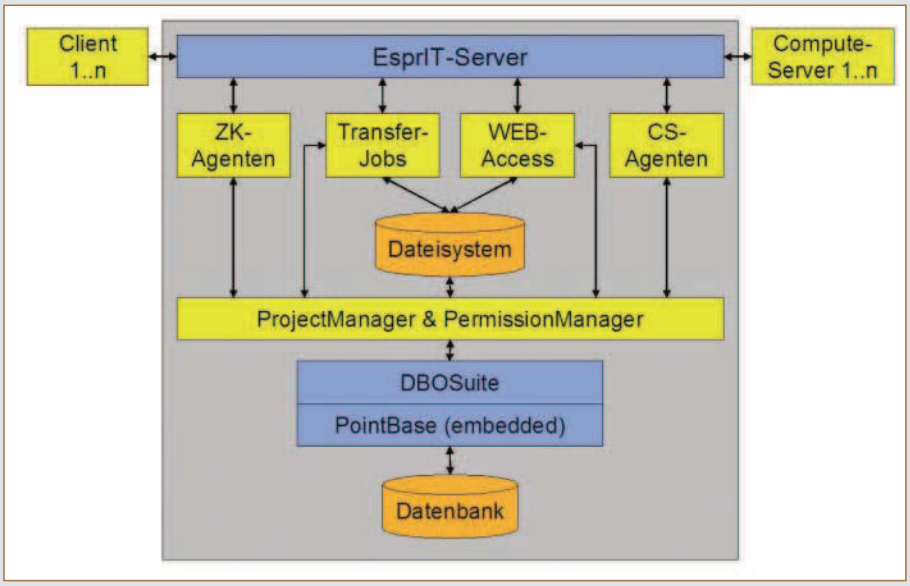

Abb. 6: Struktur des ZK-Servers

verseitig kein zusätzlicher Quellcode notwendig, da das Verfahren letztendlich identisch mit dem der Agenten ist.

Für den produktiven Betrieb wurde ein zweigeteiltes GUI entwickelt, das eine effiziente Navigation durch die vielen Modellberechnungen bietet (Abb. 7). Links sind diejenigen Teile des Projektbaumes zu sehen, für die man Zugriffberechtigungen hat und rechts erhält man zu dem jeweils selektierten Knoten einen HTML-Report.

Dessen HTML-Code wird jedoch nicht komplett vom Server abgerufen, sondern vom Client selbst dynamisch generiert. Dies geschieht teilweise aus den im Projektbaum präsentierten DBObjects und zum anderen aus Informationen, die per Agenten vom Server geholt werden. Lediglich die Bilder kommen per HTTP Request direkt vom Web-Kanal des EsprIT-Servers. Auch diese Zugriffe werden geprüft beziehungsweise gegebenenfalls verweigert.

Die letzte Komponente des Systems bildet der Compute-Server (CS). Diesen kann es in mehreren Ausführungen auf beliebig vielen Rechnern geben, je nachdem welche Analyse-Software dort vorhanden ist. Ein Compute-Server ist nichts weiter als ein unmodifizierter EsprIT-Server, der seine speziellen Fähigkeiten aus den Agenten bekommt, die der ZK-Server ihm aus dem *ProjectManager* heraus schickt. Zur Verbindung mit dem ZK-Server wird der eingebaute Server-Kaskadierungsmechanismus genutzt.

Hervorzuheben ist noch, dass ein EsprIT-Server als ein einziger Prozess im Betriebssystem läuft, der allerdings „multithreaded“ arbeitet. Für jeden verbundenen Client werden mindestens zwei Threads gestartet: Der erste empfängt Requests und arbeitet sie ab und der zweite sendet asynchron Messages an den Client. Messages können allerdings vielfachen Ursprungs sein. Welcher Client welche Message bekommt, entscheidet der *MessageDispatcher*. Auf diesem Wege kann man Nachrichten entweder an eine einzelne Session, einen User, an alle User oder ausgewählte Sessions schicken. Der Inhalt einer solchen Message ist frei programmierbar.

Dies kann ein Text, ein *DBObject* oder sogar ein *ClientCommand* sein, das auf Client-Seite ausgeführt wird. Mit diesem Mechanismus werden z.B. vom ZK-Server alle eingeloggten Clients über Änderungen im Projektbaum oder über neue Berechnungsergebnisse dynamisch informiert. Diese Aktualisierung erfolgt natürlich nur dann, wenn der betroffene User auch Rechte an den jeweiligen Informationen hat. Dies steuert ein spezialisierter *MessageDispatcher*.

Abschließend formuliert, versetzt der EsprIT-Server den Entwickler in die Lage, sich viel mehr um das Was als um das Wie zu kümmern. Die Netzwerkproblematik ist ausgeblendet und die Hauptaufgabe besteht in der Entwicklung einer guten Benutzeroberfläche und der dazugehörigen Agenten. *Thomas Stecher, BGR Hannover*

Thomas Stecher ist staatlich geprüfter Informatikassistent Fachrichtung Wirtschaft. Seit 1992 arbeitet er bei der Bundesanstalt für Geowissenschaften und Rohstoffe im Referat B2.6 (Modellberechnungen, Numerische Verfahren) als Systemadministrator und Softwareentwickler.

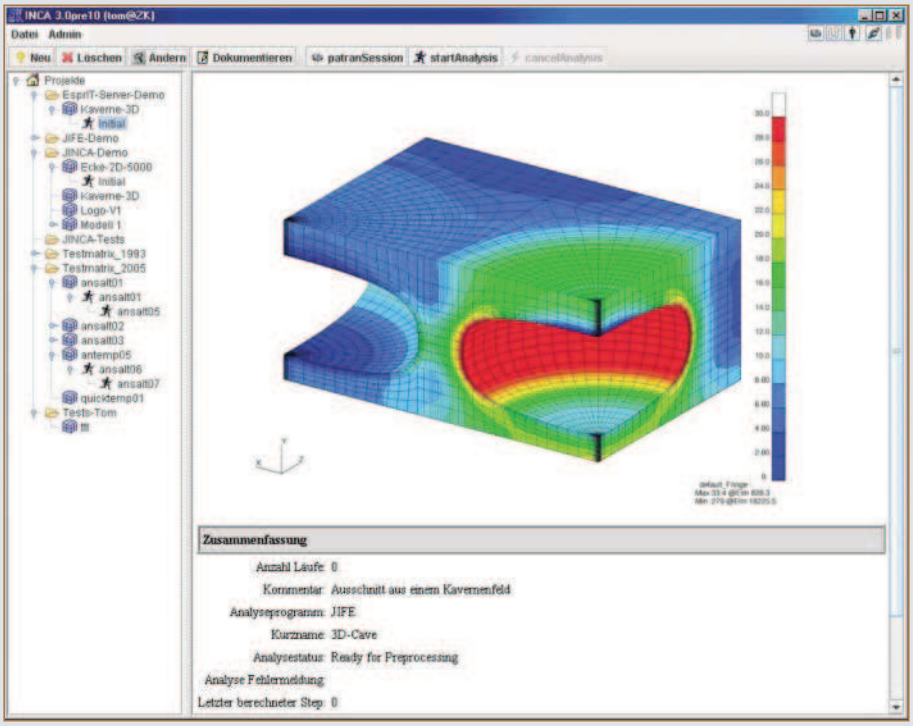

Abb. 7: Screenshot der grafischen Benutzeroberfläche des Inca-Systems