VConfig: Eine Java basierte Oberfläche zur Systemadministration

Rüdiger Schuster, Peter Kalthoff, Raimund Klute

Siemens Nixdorf Informationssysteme AG
33094 Paderborn
schuster.pad@sni.de, kalthoff.pad@sni.de, klute.pad@sni.de

Abstract. In dem vorliegenden Papier wird eine vollständig in Java implementierte Oberfläche zum System Management von gekoppelten Unix Systemen beschrieben. Es wird sowohl die Architektur des Programms diskutiert als auch auf die während der Implementierungsphase aufgetretenden Probleme eingegangen.

1 Einleitung

Unix Systeme im High-End Bereich erreichen eine Größenordnung, die weit in das Gebiet herkömmlicher Mainframes hineinreicht. Mit High-End Unix Systemen können 1000 und mehr Anwender gleichzeitig bedient werden, die Kapazität externer Speichermedien liegt im Terabyte Bereich. Dadurch bedingt ist der Hardwareaufbau dieser Systeme ausgesprochen komplex. Es kommen eine Vielzahl verschiedener System- und Peripherieschränke zum Einsatz, die jeweils mit unterschiedlichsten Hardwarekomponenten bestückt sind. Sowohl Aspekte des Datendurchsatzes als auch der Ausfallsicherheit lassen es als notwendig erscheinen, daß High-End Installationen häufig aus Kopplungen (Clustern) von Unix-Systemen bestehen.

An das Systemmanagement solcher gekoppelten Systeme werden sowohl aus funktionaler als auch aus ergonomischer Sicht die höchsten Anforderungen gestellt. Funktional muß es möglich sein, nahezu alle Managementtätigkeiten ohne Systemausfall durchführen zu können. Die Reduktion von Ausfallzeiten motiviert insbesondere die Anforderung, sich abzeichnende Hardwaredefekte frühzeitig zu erkennen und über geeignete Alarmmechanismen zu melden. Fehlerhafte Hardwarekomponenten sollen während des laufenden Betriebes aus der Konfiguration genommen und eventuell auch ausgetauscht werden können. Heutige Managementsysteme müssen natürlich mit einer modernen, objektorientierten, graphischen Oberfläche ausgestattet sein, die dem Systemadministrator alle notwendigen Informationen auf einen Blick zugänglich macht. Administrationstätigkeiten müssen einfach und ohne die Kenntnis komplexer interner Strukturen durchführbar sein.

In dem vorgestellten Projekt wurde eine Lösung basierend auf einer in Java realisierten Benutzerschnittstelle für das oben skizzierte Szenario entworfen. Mit dem realisierten Programm können sowohl Konfigurations- als auch Überwachungstätigkeiten ausgeführt werden. Eine in das System integrierte Schwellwertüberwachung aller signifikanten Hardwarebausteine gestattet es, sich abzeichnende Defekte frühzeitig zu erkennen und an der Oberfläche zur Anzeige zu bringen. Der Operator erhält über eine detaillierte graphische Ausgabe des Hardwareaufbaus gewissermaßen eine Landkarte des von ihm zu administrierenden Clusters von Unix-Systemen. In dieser Landkarte werden defekte Komponenten über eine Einfärbung der entsprechenden Bereiche kenntlich gemacht. Eine direkte Anwahl der dargestellten Bestandteile (Point- und Select-Mechanismen) versetzt den Operator in die Lage, Konfigurationsänderungen des Systems (Stillegen oder Austausch defekter Komponenten, Hinzufügen neuer Hardware) vorzunehmen. Alle von dem Administrator durchgeführten Konfigurations- und Administrationstätigkeiten wirken „clusterweit“, d.h. das Konfigurationssystem stellt für den Administrator ein Single-System Image des gesamten Clusters her.

In dem vorliegenden Papier wird zunächst ein kurzer Überblick über die Architektur und Funktionsweise des gesamten Systems gegeben. Daran anschliessend wird der Aufbau der in Java erstellten Oberflächenkomponenten detailliert erläutert. Die von uns während des Projektverlaufs geschaffenen Werkzeuge werden skizziert. Eine kurze Beschreibung der aufgetretenen Probleme sowie ihrer Lösungen (soweit verfügbar) ergänzen die Ausführungen.

2 Architektur und Funktionsweise des Systems

Die folgende Skizze gibt einen kurzen Überblick über die gewählte Architektur:

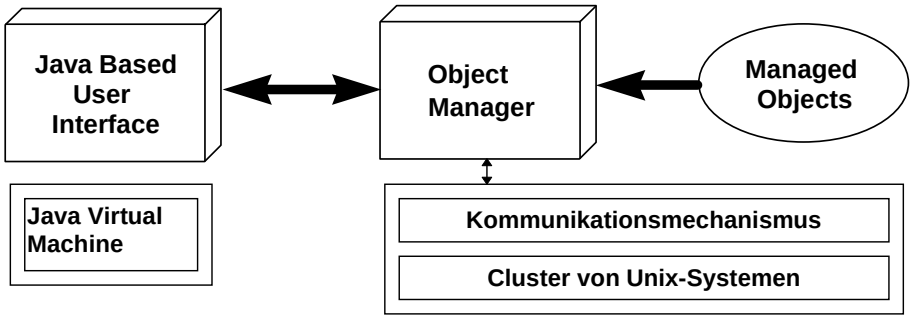

Fig. 1. Architektur des Systems

Der Administrator kommuniziert mit dem System über ein Java basiertes User Interface. Realisiert wurde das Interface mit JDK V1.1 und läuft somit auf allen Administrationsarbeitsplätzen, auf denen eine entsprechende Java Virtual Machine

verfügbar ist. Die GUI (Graphical User Interface)-Komponente kommuniziert mit Hilfe eines speziellen Management Protokolls über eine TCP/IP Verbindung mit einem Object Manager, der mit Hilfe der Managed Objects die Funktionalität des Systems realisiert. Sowohl Object Manager als auch die Managed Objects sind aus Performancegründen in C++ realisiert. Die Semantik der Managed Objects ist kompatibel mit den sich abzeichnenden Standards der Desktop Management Task Force (DMTF). Konfigurationsdaten werden über einen clusterweiten Kommunikationsmechanismus aus den gekoppelten Unix Systemen gelesen bzw. in die Systeme übernommen.

Gestartet wird das Programm durch Eingabe einer URL (Uniform Resource Locator) in einem Web-Browser bzw. durch einen Java Application Launcher. Zunächst wird von dem HTTP-Server die Java Oberfläche auf den Arbeitsplatz geladen. Die erste Aktion der Oberfläche ist der Aufbau der TCP/IP Verbindung mit dem Object Manager.

Wir wollen nun zunächst den Aufbau der Oberfläche genauer erläutern:

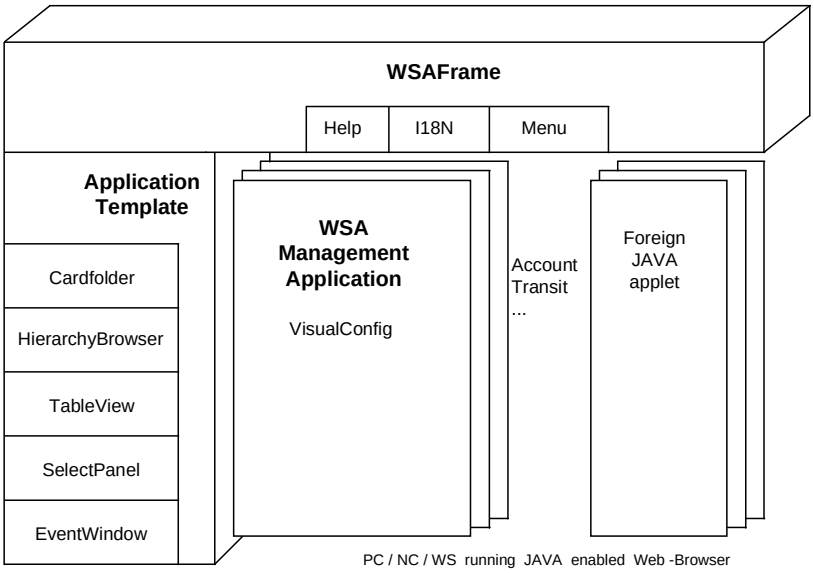

Fig. 2. Architektur der Oberfläche

Die Oberfläche besteht aus einem Rahmen (hier WSA Frame, WSA steht dabei für Web based System Administration), in den sich die verschiedenen System Management Funktionalitäten einhängen können. Man unterscheidet dabei zwei verschiedene Integrationslevel: Die einfachste Integration ist das direkte Einhängen von Java Applikationen. Der WSA-Frame übernimmt in diesem Fall lediglich die Funktion eines Application Launchers. Die Funktionsweise des eingehängten Java Applets ist für den WSA Frame transparent. Nutzt eine Applikation diese Methode, hat sie keinen Zugang zur Infrastruktur des Administrationssystems. Zu dieser Infrastruktur gehören u.a. User Interface Komponenten, das Protokoll und die

Managed Objects. Diese Möglichkeit der Integration wird angeboten, um beispielsweise Third Party Applikationen einzuhängen.

Neben dieser einfachen Methode bietet der Frame eine ganze Reihe von Schnittstellen, über die eine zu erstellende Management-Applikation integriert werden kann. Dieses umfaßt zum einen Zugänge zu User-Interface Komponenten, wie etwa dem Menübalken, dem Hilfesystem oder zu Internationalisierungskomponenten. Zum anderen kann eine derartig integrierte Applikation die gesamte Infrastruktur (Protokollzugang, Zugriff auf Managed Objects) des Systems verwenden.

3 Layout des User-Interfaces

Das Design der Oberfläche orientierte sich an dem Paradigma, dem Administrator mit möglichst wenig Mausaktionen Zugang zu der von ihm gewünschten Information zu verschaffen. Es werden, die am Cluster beteiligten Systeme als Hardware-Hierarchie (Cluster, Schrank, Einschübe, Boards, ...) dargestellt, in der man navigieren kann. In einer Arbeitsfläche werden die Objekte der jeweiligen Hierarchiestufe angezeigt:

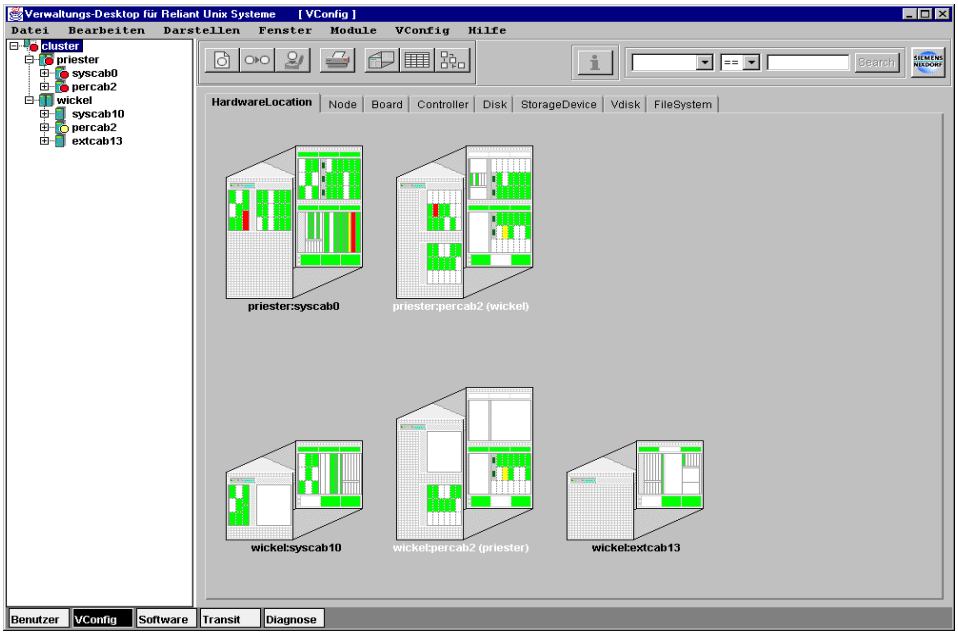

Fig. 3. Graphische Hardwaredarstellung

Neben dieser *graphischen* Sicht des Systems wird eine *objektorientierte* Sicht angeboten, etwa: alle Platten, Controller oder Benutzer des Clusters. Die Menge der

angezeigten Objekte kann über die Angabe von Selektionskriterien syntaktisch eingeschränkt werden.

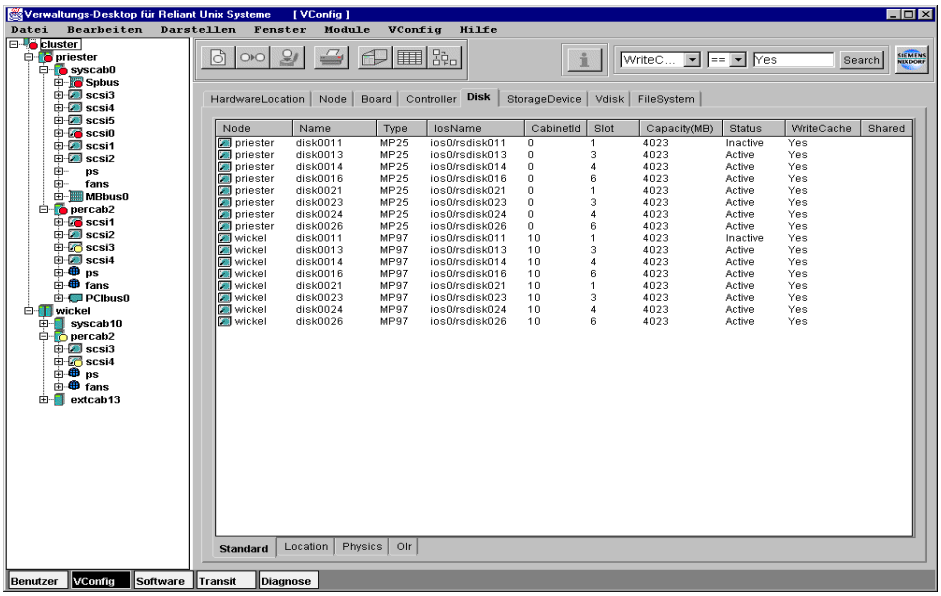

Fig. 4. Tabellarische Objektdarstellung

Durch Anwahl eines Objekts kann dessen Inhalt über eine objektspezifische Dialogbox angezeigt und manipuliert werden.

4 Verwendete Werkzeuge

In der Anfangsphase des Projekts wurden verschiedene Engineering-Tools, z.B. VisualJ++ (Microsoft) und Java WorkShop (Sun) evaluiert. Die Entscheidung fiel letztendlich für Symantec's Visual Cafe 1.1, einerseits wegen der besseren Bedienbarkeit, andererseits aber auch, da so dem Java-Standardisierungs-Zwist zwischen Sun und Microsoft aus dem Weg gegangen wurde.

Zum Profiling wurden Produkte verschiedener Anbieter evaluiert. Letztendlich konnte jedoch keines dieser Produkte überzeugen. Der Nachweis der Java-Kompatibilität sollte mit JavaPureCheck (JavaSoft) durchgeführt werden.

5 Komponentenbaukasten für Dialogboxen

Erste Abschätzungen während der Projektplanungsphase zeigten schon, daß die Realisierung der objektspezifischen Dialogboxen einen signifikanten Teil des Gesamtaufwands in Anspruch nehmen würde. Dialogboxen müssen für jedes Objekt entworfen (graphisches Layout) und mit einer objektspezifischen Logik versehen werden (das Tauschen einer Platte hat andere Aktionen auf dem zu verwaltenden System zur Folge als das Anlegen eines Benutzers).

Die funktionale Schwäche der von Sun im AWT bzw. Swing mitgelieferten Komponenten wurde durch eine eigens entwickelte Toolbox behoben, die die AWT- oder Swing-Klassen (z.B. TextArea) um sinnvolle Eigenschaften erweitert oder aus Basis-Komponenten komplexere Konstrukte (z.B. ButtonPanel, BrowsePanel oder PieCharts) bildet.

Gerade Standardprobleme wie Font- und Layoutbehandlung, Plausibilitätschecks, Internationalisierung, Hilfesystem, usw. können somit auch von Programmierern ohne Swing- oder AWT-Erfahrung schnell gelöst werden.

Darüber hinaus erreicht man mit einer solchen Toolbox natürlich eine gewisse Vereinheitlichung der Präsentation von Informationen, bei einem so großen Projekt ein nicht zu vernachlässigendes Detail.

Ein weiterer positiver Effekt ist das Information-Hiding der zugrunde liegenden Basis-Komponenten, ein Aspekt, der sich beim Umstieg von den Symantec-Klassen auf Swing bezahlt gemacht hat.

6 Imagebuilder

Die graphische Anzeige der Systemkomponenten ist ein hoch dynamischer Prozess. So können z.B. jederzeit neue Hardwarekomponenten (neue Schränke, weitere Controller, ...) hinzugefügt oder der vorhandene Aufbau verändert werden. Darüber hinaus muß die graphische Anzeige natürlich Zustandsänderungen der Hardware unmittelbar anzeigen. Die Oberflächenkomponente Imagebuilder wurde entworfen, um diese Probleme zu lösen.

Der Imagebuilder (s. Fig. 5) hat die Aufgabe, aus Bild-Beschreibungsdateien (sog. CFD-Dateien) der Objekte dynamisch Images für die Objektpräsentation zu erzeugen. Natürlich werden verschieden Auflösungen und Objektzustände hierbei berücksichtigt.

Die ImageFactory liest die eingeparsten Beschreibungen aus dem CFDFile-Cache, der sich wiederum die Daten über den CFDFileParser/-Reader vom HTTP-Server besorgt. Enthält die Beschreibungsinformation dynamisch zu ermittelnde Attributwerte, so werden diese zur Laufzeit vom Objektmanager gelesen. Die CFD Beschreibungen beinhalten einen „Subbild-Mechanismus“, so lassen sich komplexe Bilder aus einzelnen Bausteinen zusammensetzen. Außerdem werden Standard-Graphik-Funktion wie DrawLine, DrawText, Fill, usw. unterstützt. Somit lassen sich alle notwendigen Bilder dynamisch erzeugen.

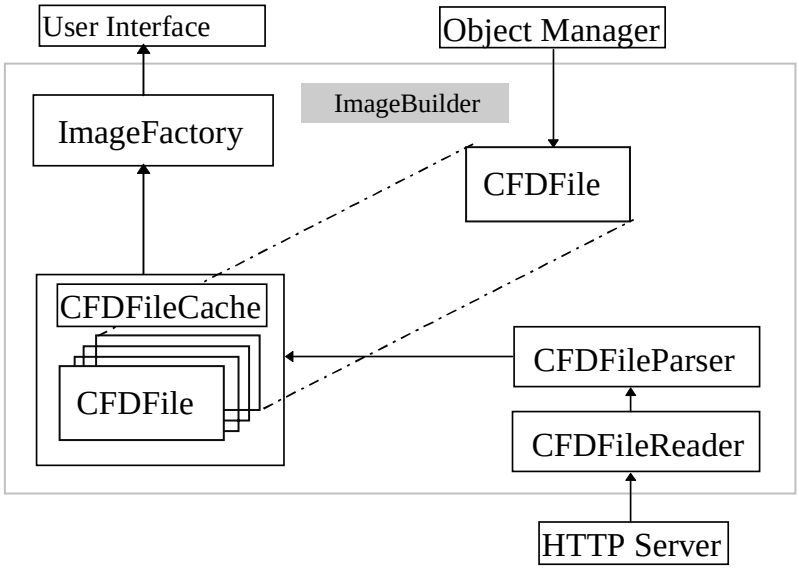

Fig. 5. Architektur des ImageBuilders

Insbesondere bei Container-Objekten wie z.B. Datenbus-Käfigen wird der Vorteil dieses Ansatzes klar. Auf dem zugrundeliegenden Image des Hardware-Busses können dann nämlich die Bilder der einzelnen Komponenten dieses Busses gezeichnet werden, und zwar mit Hilfe derselben Beschreibung, die auch bei der exklusiven Anzeige der Komponente verwendet wird, da sämtliche Positionierungen innerhalb der Beschreibung als relative Koordinaten zum Bildursprung interpretiert werden. Man kann also mit nur einer Beschreibungsdatei pro Objektklasse eine Vielzahl verschiedener Images zusammenstellen. Ein besonderer Clou ist der HotSpot-Mechanismus, d.h. der Benutzer kann auch bei zusammengesetzten Images Tooltips oder objektspezifische Aktionen zu einem der enthaltenen Objekte anfordern bzw. ausführen.

7 Die Protokollschnittstelle des User Interfaces

Das User Interface in der Rolle des Client sendet Anfragen an den Objectmanager und erhält entsprechende Antworten. Darüber hinaus können beide Seiten auch asynchrone Nachrichten, z.B. zur Übermittlung von Events, senden. Diese freilaufenden Nachrichten, und gewisse Anforderungen bezüglich der Performance, bedingen eine Übertragung im Vollduplex-Modus. Dies wird durch ein 3-Thread-Konstukt erreicht, das über eine Queue synchronisiert wird.

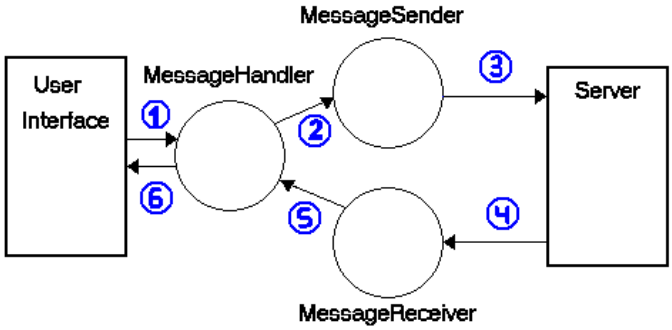

Fig. 6. Thread-Architektur

Das User-Interface übergibt Aufträge an den MessageHandler. Dieser stellt Methoden zum Senden(1) bzw. Empfangen(6) der Nachrichten bereit. Der MessageHandler seinerseits bedient sich des MessageSender-Threads(2), um die Nachrichten an den Server(3) zu senden, bzw. des MessageReceiver-Threads(5), um Nachrichten vom Server(4) zu empfangen.

Sowohl auf Sender- als auch auf Receiver-Seite werden die Messages „gequeued“. Die Daten werden, um Plattformunabhängigkeit zu gewährleisten, im XDR-Format übertragen.

8 Aufgetretene Probleme

Die unten beschriebenen Probleme erzeugten zum Teil erheblichen zusätzlichen Aufwand. Ein großer Teil der Probleme ist auf die Tatsache zurüchzuführen, daß die Dynamik im Java-Umfeld und die damit verbundenen Instabilitäten sowohl in bezug auf die Schnittstellenkompatibilität als auch auf die Qualität der vorhandenen Standard-Komponenten recht hoch sind. Die „Problem-Highlights“ waren:

1. „Instabile Java-Versionen“

   Praktisch jeder JDK- und Swing-Stand vor der freigegebenen Version hatte eine für einen Projekteinsatz ungenügende Qualität. Neue Releases behoben bekannte Fehler nicht, oder brachten stattdessen neue mit. Der Aufwand für Workarounds war erheblich.

2. „Fehlende Browserunterstützung“

   Browser liefern bis heute, von Sun’s *HotJava* abgesehen, trotz anderslautender Versprechen, keinen vollständigen JDK1.1-Support. Ein kleiner Lichtblick am Horizont ist Netscapes Ankündigung zum Sommer ‘98, zu JDK1.2 kompatibel zu sein. Das würde auch einen Performance-Gewinn bedeuten, da JDK1.2 bereits die

verwendeten Swing-Klassen beinhaltet und diese somit nicht nachgeladen werden müßten.

3. „Einarbeitung in objektorientierte Programmierung“
   Java als Programmiersprache ist sehr schnell erlernbar, die Umstellung auf die objektorientierte Programmierung (Generalisierung, Polymorphismus) ist jedoch für den Java-Neuling eine wesentlich höhere Hürde. Auch das neue Entwicklungsumfeld (PC-Tools, Browser, Online-Dokumentation per JavaDocs) brachte einen höheren Einarbeitungsaufwand als geplant.

4. „Instabile Schnittstellen in der Anfangsphase“
   Schnittstellenänderungen beim Wechsel von JDK1.0 auf 1.1 verursachten zusätzlichen Aufwand. Einer der Hauptgründe war die Umstellung der Event-Schnittstelle auf das Event-Delegation-Modell.

5. „Mangelnde Tool-Unterstützung“
   Ein derart großes Java-Projekt überfordert quasi alle vorhandene Tools. So stürzt z.B. *javadocs* aus dem JDK1.1.5-Paket mit der Fehlermeldung „Memory overflow“ ebenso ab wie *JavaPure Check* oder die eingesetzten Performance-Profiler.

6. „Schwache Klassenbibliothek“
   Für eine derart verbreitete Sprache stehen immer noch sehr wenig „High-Level“-Klassen zur Unterstützung bei Standardproblemen (Beispiel: *editierbare Tabellen*, *Quicksort*, *Internationalisierung*, etc.) zur Verfügung. Viele dieser Dinge werden heute sicherlich noch redundant gelöst.

9 Projekthistorie

Der Startschuß fiel im März 1997. Zwei „Java-erfahrene“ Studenten der Uni Enschede schrieben im Rahmen eines sechsmonatigen Praxissemesters die erste Implementierung des GUIs auf Basis von JDK1.0. Parallel hierzu wurde der Objektmanager implementiert.

Im Juli 1997 wurde dann, wegen der verbesserten Features im AWT-Package, auf JDK1.1 umgestellt. Für die sichtbaren Komponenten wie TableView und TreeView wurden zuerst entsprechenden Klassen von Symantec verwendet.

Mit zunehmender Stabilität der Swing-Klassen wurden dann alle sichtbaren Komponenten hierauf rebasiert. Heute fundiert die Implementierung auf JDK1.1.5 und Swing1.0.1, welches sich sowohl dem Standard-AWT als auch den Symantec-Klassen gegenüber in bezug auf Funktionalität und Stabilität als überlegen erwiesen hat.
Das hier vorgestellte Projekt wird als Bestandteil der nächsten ReliantUNIX Version zur Administration der RM-Server eingesetzt.