Java-Container für CORBA-Komponenten am Beispiel des CFD-Simulationssystems TENT

Thomas Breitfeld
thomas.breitfeld@gmd.de

GMD/SCAI, Sankt Augustin, Germany

Zusammenfassung Dieser Artikel beschreibt die Benutzung von Java in einem CFD-Simulationssystem[1], welches die gegenwärtig bestehenden Probleme bezüglich der Kopplung und Konfiguration der in einem solchen System notwendigen Komponenten überwindet. Dieses System namens TENT[2] dient der Unterstützung des Entwurfs und Designs von Flugzeugen und deren Teilen. TENT ist ein Komponentensystem. Die zentralen Algorithmen zur Steuerung und Konfiguration der an einer Simulation beteiligten Anwendungskomponenten sind in Java mit Hilfe der JavaBeans Technologie implementiert. Die Anwendungskomponenten selbst sind jedoch sehr heterogener Natur, so daß für ihre Schnittstellendefinition und -implementierung CORBA benutzt wird. Um beide Welten auf Komponentenebene zu verbinden, bedarf es einer Komponentenarchitektur für CORBA, die von TENT definiert wird. Neben einer Einführung in die Problematik dieser Simulationssysteme wird im einzelnen erläutert, welchen Design Patterns die Komponentenarchitektur genügt und wie diese von Java aus für die Kopplung und Überwachung des Systems und für ein generisches Konfigurations-GUI[3] benutzt wird. Es wird gezeigt, daß sich Java hervorragend für diese Kombination eignet und Vorteile gegenüber anderen Implementierungssprachen an diesen Stellen aufweist. Damit wird die vorteilhafte Benutzung von Java in einem industriellen und praxisrelevanten Anwendungsszenario unter Beweis gestellt.

1 Einführung

1.1 Problem

CFD Simulationen werden intensiv bei der Entwicklung und dem Design von Flugzeugen und ihren Teilsystemen benutzt. In den letzten Jahren konnten die Simulationszeiten von derartigen Systemen durch den Einsatz von Parallelrechnern und verbesserten Algorithmen auf ein in der Praxis akzeptierbares Maß reduziert werden. Allerdings ist es immer noch ein Problem, die verschiedenen Applikationen und Werkzeuge, die für eine derartige Simulation benötigt werden,

[1] CFD - Computational Fluid Dynamics
[2] TENT - TEstbed for Numerical Turbines
[3] GUI - Graphical User Interface

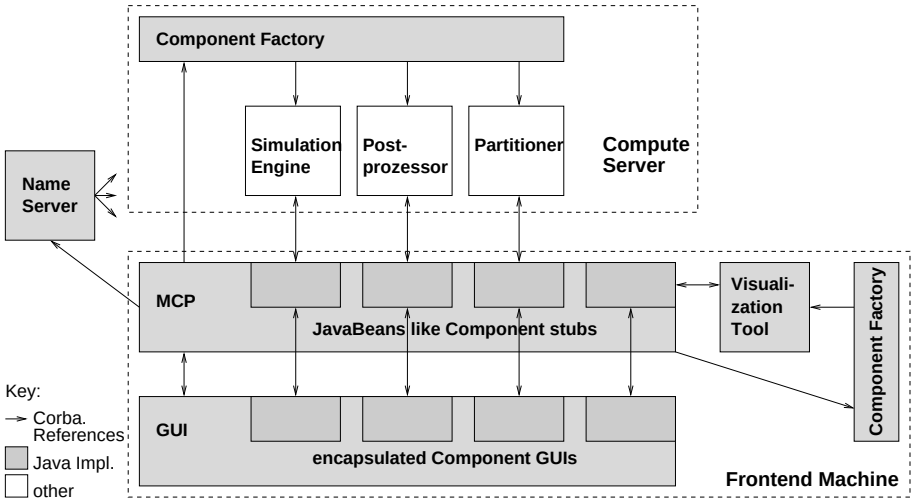

Abbildung1. TENT Systemarchitektur

miteinander zu koppeln und ihre Benutzung einfach zu gestalten. Wir sehen uns heute der paradoxen Situation gegenüber, daß der Ingenieur wesentlich mehr Zeit für das Konfigurieren und Zusammenstellen der Werkzeuge für eine Simulation benötigt, als die Simulation selbst an Zeit beansprucht. Auf der anderen Seite wurden in den letzten Jahren große Fortschritte im Bereich der verteilten, heterogenen und objektorientierten Komponentensysteme gemacht. Das momentan in der Entwicklung befindliche System TENT [1] überträgt diese Fortschritte auf das Gebiet des 'traditionellen' HPC[4]-Ansatzes um die genannten Probleme zu lösen.

1.2 Ansatz

Der verfolgte Komponentenansatz wurde aus folgenden applikationsspezifischen Zielen und Anforderungen heraus entwickelt:

- Ein CFD Simulationssystem besteht aus mehreren Teilen, die für eine kompletten Simulationsprozeß anwendungsspezifisch zusammengeschaltet werden müssen. Solche Teile sind zum Beispiel Preprozessoren, Postprozessoren, Gitterpartitionierer, Simulationscodes (Löser) und Visualisierungswerkzeuge, welche im Abschnitt 3 näher beschrieben werden. Diese Komponenten müssen verallgemeinerte Schnittstellen erhalten, die ein implementierungsunabhängiges Zusammenarbeiten ermöglichen.
- Die benutzten Prozeßketten zur Simulation eines bestimmten Problems sind anwendungsbezogen und sehr dynamisch, TENT muß eine Systemkomponente zur Verfügung stellen, welche das interaktive Verschalten von Kompo-

[4] HPC - High Performance Computing

nenten mittels einer zentralen Kontrollinstanz realisiert. Diese Steuerungskomponente soll eine GUI besitzen, und so das Steuern von Komponenten von einem zentralen Punkt aus ermöglichen, auch dann, wenn die beteiligten Komponenten auf entfernten Parallelrechnern laufen, die nur über eine WAN[5]-Verbindung zugreifbar sind. Diese Steuerungskomponente wird in Abschnitt 3.2 näher beschrieben.

- Ein CFD Simulationssystem besitzt eine große Anzahl von Konfigurationsparamtern welche sowohl zur Installation als auch zur Laufzeit veränderbar sind und vor allem mit denen anderer Komponenten des Systems konsistent gehalten werden müssen. TENT muß einen Konfigurationsdatenmechanismus und eine einheitliche Schnittstelle für den Zugriff und damit für die Verteilung über das System zur Verfügung stellen. Desweiteren soll die Möglichkeit existieren diese Konfigurationsparamter in einer GUI zu visualisieren und zu verändern, ohne das für jede entsprechende Komponente spezieller Code dafür zu entwickeln ist.

Komponentenmodelle können diese Anforderungen sehr gut erfüllen, allerdings ist es für TENT sinnvoll, zwei verschiedene Technologien für die beiden notwendigen unterschiedlichen Komponentenarten zu verwenden.

Die *Anwendungskomponenten* von TENT sind plattformabhängig implementiert, da sie einerseits in den meisten Fällen bereits existieren und andererseits sehr leistungskritisch sind. Sie werden in C, C++ und FORTRAN codiert. Es liegt nahe, diese Komponenten auf basis der Common Object Request Broker Architecture (CORBA [3]) zu beschreiben und zu koppeln. CORBA realisiert die Integration von verteilten Objekten zu einem Gesamtsystem im wesentlichen durch das Beschreiben von Schnittstellen mittels einer sprach- und plattformunabhängigen Interface Description Language (IDL [3]). Die Interfacebeschreibungen werden in eine der spezifizierten Implementierungssprachen übersetzt. Der generierte Programmcode enthält Aufrufe zum Object Request Broker (ORB), welcher die Funktionen für das Aufrufen von Methoden entfernter Objekte kapselt.

Demgegenüber sollen die zu entwickelnden *Systemkomponenten* möglichst unabhängig von der gerade verwendeten Plattform funktionieren und innerhalb von TENT solche Aufgaben wie die Lebenszyklusüberwachung, die Administration und die Konfiguration von Komponenten übernehmen. Da die Systemkomponenten leistungsunkritisch sind, bietet sich Java und sein Komponentenmodell JavaBeans [10] für eine Implementierung an. Java ist nicht nur wegen der graphischen Fähigkeiten und Plattformunabhängigkeit in diesem Bereich vorteilhaft, sondern auch wegen der Möglichkeiten die Runtime Type Information, Introspection und das dynamischen Nachladen von Implementierungsklassen bieten. Eine Kombination von Java-Welt und CORBA-Welt ist möglich, da für Java ein IDL Language Binding[6] definiert ist.

[5] WAN - Wide Area Network

[6] IDL Language Binding - In der CORBA Spezifikation [3] definierte Abbildung von IDL zu einer Implementierungssprache

Die zentrale Systemkomponente, der MCP[7], stellt im System dabei den Container für die Anwendungskomponenten dar (siehe Abbildung 1). Momentan ist jedoch keine Komponentenarchitektur für CORBA definiert oder in Form von Produkten erhältlich. TENT definiert deshalb eine Komponentenarchitektur auf basis der heute verfügbaren CORBA 2.0 konformen Implementierungen. Um die softwaretechnische Integration von Java-Container und CORBA-Komponenten möglichst einfach und intuitiv zu gestalten sowie vorhandene Entwicklungswerkzeuge als auch Code benutzen zu können, ist die TENT-Komponentenarchitektur stark an das JavaBeans-Modell angelehnt und nur dort wo Applikations- oder Architekturbeschränkungen es sinnvoll erscheinen lassen, mit Erweiterungen versehen.

Im weiteren werden deshalb zuerst die Kernaspekte der TENT-Komponentenarchitektur und darauf aufbauend die beispielhaft realisierte Applikation sowie spezielle Aspekte der zentralen Systemkomponenten beschrieben.

2 TENT-Komponentenarchitektur

2.1 Überblick

Komponentenmodelle bestehen aus zwei Hauptbestandteilen: dem *Objektmodell* und der *Komponentenarchitektur*. CORBA definiert ein konkretes *Objektmodell* was garantiert, daß Komponenten unterschiedlicher Entwickler, die unterschiedliche Programmiersprachen benutzen, problemlos auf binärer Ebene sowohl innerhalb eines Systems als auch in verteilten Systemen zusammenarbeiten. Die *Komponentenarchitektur* ist ein Prinzip und ein Set von APIs, häufig auch als "Design Patterns" bezeichnet, welche es einem Entwickler erlauben, Softwarekomponenten zu entwickeln und diese dynamisch (d.h. erst zur Laufzeit) zu einer fertigen Applikation zu kombinieren. Eine Komponentenarchitektur basiert auf einem bestimmten Objektmodell. Folgende Designziele waren für die in diesem Kapitel beschrieben Komponentenarchitektur maßgebend:

1. Ausrichtung an den Erfordernissen der Applikation und nicht an einer größtmöglichen Generalität.
2. Übereinstimmung mit dem JavaBeans-Modell soweit dies mit dem gegebenen CORBA-Objektmodell realisierbar ist.
3. Anlehnung an die gegenwärtigen Vorschläge für eine CORBA-Komponentenarchitektur um kompatibel werden zu können, wenn selbige standardisiert sein wird.

2.2 Allgemeine Interfaces

Neben den bereits erwähnten Design Patterns definiert die TENT-Komponentenarchitektur eine Hierarchie von allgemeingültigen Interfaces (siehe Abbildung 2).

[7] MCP - Master Control Process

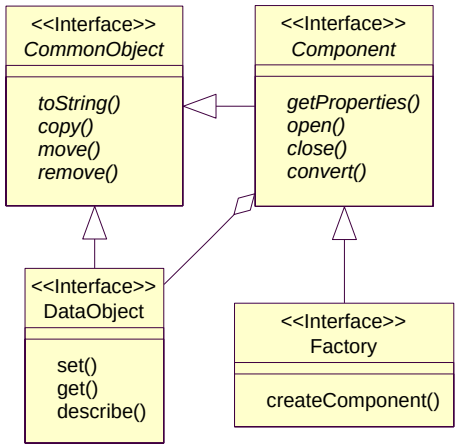

Abbildung2. TENT Allgemeine Interface Hierarchie (UML Notation)

Das `CommonObject`- und `Factory`-Interface realisieren das Kreieren und Zerstören von Komponenten in der Art, wie es im *CosLifeCycle-Service* [7] definiert ist. Das `DataObject`-Interface ist eine allgemeine Datenschnittstelle, welche im Zusammenhang mit dem Property-Modell in Abschnitt 2.4 beschrieben wird. Das `Component`-Interface ist die Basisklasse aller Komponenten. Die beiden Methoden `open` und `close` realisieren dabei eine Sessionverwaltung, mit Hilfe derer die konkurrierende Benutzung einer Komponente in mehreren TENT-Projekten gesteuert werden kann. Dies ist notwendig, da TENT-Komponenten häufig ihre lokalen (Parallel-)Rechnerresourcen selbst verwalten und als eine Art Service für TENT-Projekte fungieren. Über die `convert` Methode kann jede Komponente Konvertierungsdienste anbieten. Wir gehen davon aus, daß insbesondere jede Applikationskomponente bereits mehrere Input- und Output-Formate verarbeiten kann und die damit verbundene Fähigkeit der Formatkonvertierung dem System zur Verfügung stellt. Die `getProperties` Methode gibt eine Referenz auf das Property-Datenobjekt der Komponente zurück (siehe Abschnitt 2.4). Zusätzlich erbt jede Komponente das in der *OMG Externalization Service Specification* [7] standardisierte `CosStream::Streamable`-Interface, um seine Konfiguration zu laden oder zu sichern.

2.3 Event-Modell

Das Event-Modell entspricht den von JavaBeans bekannten Konventionen und Mechanismen, bis auf die Definition des Events selbst. Da wir in einem verteilten System arbeiten, ist es wünschenswert, das Event als Wert und nicht als Referenz, wie in Java immer der Fall, zu übergeben. Dies verhindert entfernte Methodenaufrufe beim Zugriff auf die Elemente des Events. Allerdings definiert das CORBA-Objektmodell keine 'Call by value'-Argumentübergabe für Objekte, so daß wir anstatt eines Event-Objektes eine Event-Struktur übertragen:

```
module Tent {
   struct Event {
      CommonObject source;
   }
}
struct <Xx>Event {
   Tent::Event baseEvent;
   ...
}
```

Das Event-Modell wird außerdem um die Definition von asynchronen Events erweitert, da die Event-Bearbeitung, bedingt durch die verteilte Lokation der Komponenten, häufig parallel ausgeführt werden kann. Das Senden eines asynchronen Events blockiert den Sender nicht bis zur vollständigen Abarbeitung durch die Empfänger. Realisiert wird dies durch eine Ergänzung in den `EventSupport`-Interfaces. Zu jeder dort vorhandenen Methode:

```
void fire<action-name>(<Xx>Listener l);
```

wird eine zusätzliche Methode:

```
void fire<action-name>_async(<XxListener l, CallbackListener l);
```

definiert, der optional ein `CallbackListener`-Interface übergeben werden kann, dessen `callback` Methode dann aufgerufen wird, wenn die Event-Abarbeitung vollständig abgeschlossen ist. Die Implementierung der Asynchronität kann entweder mittels Threads oder der im DII[8] definierten Methoden `send_deferred` und `get_response` erfolgen.

2.4 Property-Modell

Die Definition des Property-Modells erfolgt unter Verwendung der von TENT definierten generischen Datenschnittstelle. Dieses `DataObject`-Interface wird zwischen den Komponenten auch für die den Datenaustausch benutzt und ist an einen Vorschlag von Mowbray [13] angelehnt.

Strukturell beschreibt das `DataObject`-Interface eine Liste von Name-Wert-Paaren, die beliebig ineinander verschachtelt sein können. Einzelne Property Werte sind über die `get()` und `set()` Methoden des `DataObject`-Interfaces zugreifbar. Zur Verdeutlichung ist in Abbildung 3 ein Ausschnitt der entsprechenden IDL-Definition dargestellt. Der Wert einer Property wird immer in Form eines `any` übergeben. Dies ist ein von CORBA definierter generischer Datentyp, der es erlaubt, beliebige andere Typen durch das Hinzufügen eines Typecodes typsicher zu speichern.

Das DataObject selbst unterstützt die von JavaBeans bekannten Event-Unterstützung für bound und constraint Properties, welche auf gleiche Weise funktioniert. Um die Anzahl der entfernten Methodenaufrufe beim Auslösen eines

[8] DII - CORBA Dynamic Invocation Interface

```
module Tent {
   interface DataObject : Tent::CommonObject, CosStream::Streamable {
      exception PropertyNotFound { string propertyName; };
      exception WrongValueType { string propertyName; };
      exception ReadOnlyProperty { string propertyName; };

      any get(in string propertyName) raises(PropertyNotFound);

      void set(in string propertyName,in any value)
               raises(PropertyNotFound, WrongValueType,
                      ReadOnlyProperty, PropertyVetoException);

      typedef sequence<string> NameList;
      typedef sequence<TypeCode> TypeList;
      typedef sequence<long> AttributeList;
      const long ATTR_READ_ONLY = 1;
      void describe(out unsigned long nrOfProps, out NameList names,
                    out TypeList types, out AttributeList attributes);
      ...
   }
}
```

Abbildung3. Ausschnitt der IDL Definition des TentData Modules (Event Unterstützung nicht dargestellt.)

Events zu minimieren, ist es jedoch notwendig, ein propertyselektives Registrieren eines `PropertyChangeListener` oder `VetoableChangeListener` zu ermöglichen. Deshalb unterstützt das `DataObject` zusätzlich vier Methoden, wovon im folgenden nur beispielhaft die beiden für bound Properties gezeigt sind:

```
void addPropertyChangeListener(in NameList propertyNames,
                               in PropertyChangeListener l)
     raises(TooManyListeners);
void removePropertyChangeListener(in NameList propertyName,
                                  in PropertyChangeListener l);
```

Ebenso wie die `Component` unterstützt das `DataObject` das `CosStream::Streamable`-Interface zum Serialisieren seiner Properties.

Im Vergleich zu JavaBeans unterscheidet sich die Property-Definition einer TENT-Komponente im wesentlichen in 2 Punkten:

1. Alle Properties sind über ein generisches Interface zugreifbar, indem der Typ `any` verwendet wird.
2. Das Property-Interface wird nicht innerhalb des Interfaces einer Komponente definiert, sondern stellt ein eigenständiges Objektinterface dar.

Durch diese Veränderungen ist die Möglichkeit gegeben hierarchische Property-Objekte aufzubauen und Properties zur Laufzeit hinzuzufügen oder zu entfernen.

Weiterhin läßt sich dadurch eine Art Offline-Konfigurationsmodus implementieren, was bei der Anwendung von knappen Ressourcen, wie zum Beispiel Hochleistungsrechnern notwendig ist.

3 TENT Applikationsarchitektur

Aufbauend auf der im vorangegangenen Kapitel beschriebenen Komponentenarchitektur wurden alle TENT-Anwendungs- als auch Systemkomponenten implementiert. Sie sind in Abbildung 1 zusammen mit ihren gegenseitigen Assoziationen und ihrer Implementierungssprache gezeigt. Die wesentliche Erweiterung, welche diese Komponenten zum allgemeinen `Component`-Interface hinzufügen, sind konkrete `EventListener`-Interfaces, die das Verschalten der Komponenten zu einem Gesamtsystem erlauben (siehe Abbildung 4). Mögliche Anwendungsszenarien sind damit im Rahmen der Event-Kompatibilität in einer genau spezifizierten Weise zusammenstellbar.

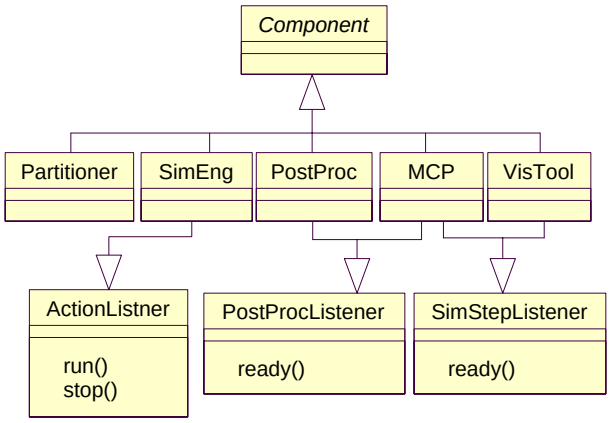

Abbildung4. TENT Komponenten- und Listener-Interfaces (UML Notation)

3.1 Applikationskomponenten

Die einzelnen Applikationskomponenten erfüllen folgende Funktionen:

Partitionierer: agiert als Service für die Simulation-Engine und bereitet die Eingangsnetzdaten so auf, daß parallel oder verteilt arbeitende Simulationscodes ein bereits in parallel abarbeitbare Teile zerlegtes Netz als Eingangsdaten erhalten. In Abhängigkeit von der Art der Simulation-Engine kann der Partitionierer auch entfallen.

Simulation-Engine: kapselt den Simulationscode und führt die eigentliche Simulation, d.h. die Berechnung von physikalischen Größen auf dem Eingangsnetz, aus.

Postprocessor: dient der Aufbereitung und Reduktion der Daten für ein bestimmtes Visualisierungstool.

Visualisierungstool: stellt die Simulationsergebnisse interaktiv und online (d.h. noch während die Simulation läuft) in Form von 3D-Szenen und Diagrammen graphisch dar.

3.2 MCP

Der MCP ist vollständig in Java implementiert und stellt softwaretechnisch einen Container für die Anwendungskomponenten dar. Allerdings arbeitet der MCP nicht mit den Komponenten direkt, sondern lediglich mit den in Java implementierten Stubs der Anwendungskomponenten. Dies ist jedoch aus Sicht des MCP bis auf die Besonderheiten beim Erzeugen und Zerstören der Komponenten transparent, so daß er in gewohnter Weise und mit teilweiser Unterstützung von vorhandenen Tools implementiert werden kann.

Der MCP ist für das projektspezifische Verwalten und Instanziieren der Komponenten sowie das Zuordnen von GUI-Komponenten verantwortlich. Dazu besitzt er selbst ein GUI-Interface, welches momentan von einer ebenfalls in Java implementierten GUI benutzt wird.

Ein TENT Projekt besteht aus zwei Hauptbestandteilen

1. Der Simulationsszene, welche das geometrische Aussehen des zu simulierenden Teils als auch notwendige Anfangszustände beschreibt. Die Simulationsszene wird üblicherweise mit einem CAD-Programm erzeugt und mittels eines Gittergenerators in ein Netz von diskreten Punkten und deren Verbindungen umgewandelt.
2. Der Simulationsumgebung, d.h. der spezifischen Auswahl von Komponenten und deren Zusammenschaltung, mit Hilfe derer die Simulationsszene simuliert werden kann.

Die Projektverwaltung des MCP erlaubt das Erzeugen und Laden derartiger Projekte. Beim Initialisierungsvorgang, d.h. beim Laden eines Projektes werden folgende Aktionen ausgeführt:

- Der MCP lokalisiert oder erzeugt die im Projekt benötigten Komponenten. Dazu bedient er sich des CORBA Naming Service oder der TENT Factory Server, welche nicht bereits im System vorhandene Komponenten erzeugen können. Im Ergebnis dieser Phase besitzt der MCP Referenzen auf alle benötigten Komponenten der Simulationsumgebung
- Der MCP deserialisiert die projektspezifische Konfigurationsinformation und konfiguriert damit die Komponenten.
- Der MCP versucht jedem der instanziierten Komponenten je nach Bedarf eine Steuer- und Konfigurations-GUI zur Verfügung zu stellen und sie in die System-GUI einzubetten. Dazu gibt es zwei Möglichkeiten. Einerseits kann eine Komponente den Einsatz einer speziellen von ihr selbst gelieferten Bean favorisieren, indem sie mittels der Property "`DefaultGUIBean`" den Klassennamen der entsprechenden Bean propagiert. Andererseits, falls diese Property fehlt oder die angegebene Klasse nicht geladen werden kann, instanziiert

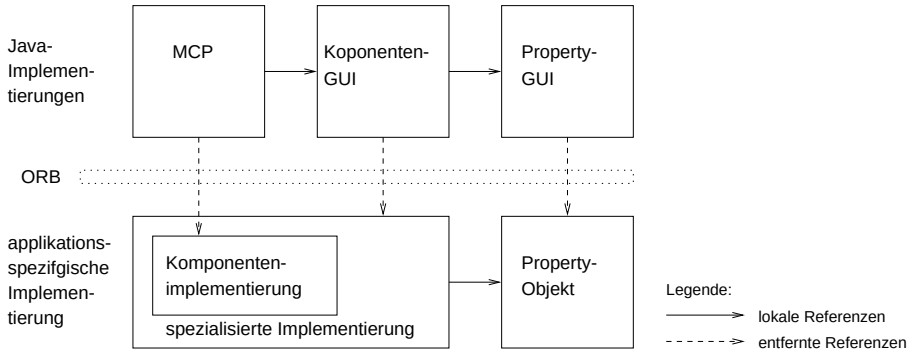

Abbildung5. Referenzenübersicht zwischen GUI- und Anwendungskomponenten

der MCP eine Standard-Bean, welche keine applikationsspezifischen Steuerelemente sondern lediglich eine spezifische oder allgemeine Property-Bean nachlädt und anzeigt (siehe Abschnitt 3.3).
- Der MCP nimmt die 'Verdrahtung' der Komponenten vor, d.h heißt er registriert entsprechende Event-Listener bei Event-Sendern. Mit diesem Schritt ist die Initialisierung der Simulationsumgebung abgeschlossen.

Die beschriebenen Schritte müssen natürlich beim Anlegen eines neuen Projektes 'von Hand', d.h. vom Benutzer ausgeführt werden. Dies geschieht gewöhnlich über das GUI des MCP.

Die Referenzbeziehungen, welche in einem geladenen Projekt zwischen MCP, einer speziellen Applikationskomponente und der entsprechenden GUI-Beans entstanden sind, schematisiert Abbildung 5.

3.3 Komponenten-GUI

Eine Komponenten-GUI besteht aus zwei Teilen: der GUI für die Komponente selbst und darin eingebettet die Property-GUI. Beide Teile sind als JavaBean implementiert, so daß sie im folgenden als Komponenten-Bean und Property-Bean bezeichnet werden. Die Komponenten-Bean hält eine Referenz auf ihre entsprechende Komponente, während die Property-Bean mit der Referenz des entsprechenden Property-Objektes verknüpft ist. Das Property-Object ist Grundlage sowohl für die allgemeine oder komponentenspezifische Initialisierung der GUIs als als auch für die Standard-GUI-Eigenschaften von TENT.

Abbildung 5 zeigt die Assoziationen zwischen Komponenten-Bean, Property-Bean und Property-Objekt. Im allgemeinen Fall dient die Komponenten-Bean lediglich als Container für die Property-Bean, und kann damit für alle TENT-Komponenten durch die selbe Klasse implementiert werden. Im speziellen Fall ist es möglich, auf der Komponenten-Bean zusätzliche Steuerelemente anzuordnen, die auf der Interfacedefinition einer speziellen Komponentenimplementierung beruhen können.

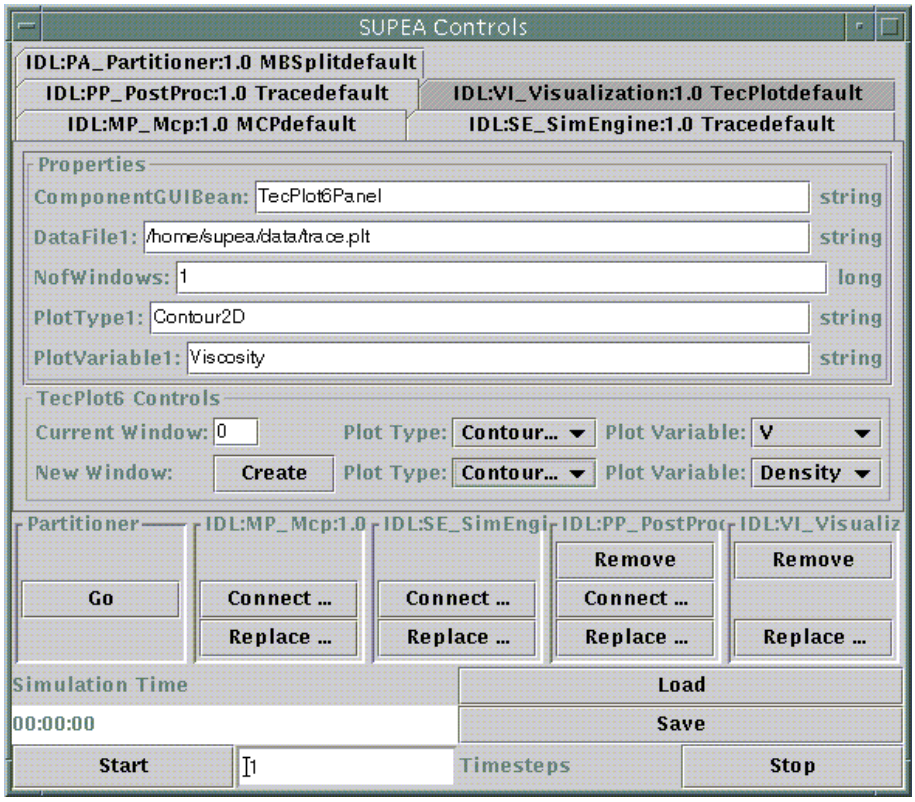

Abbildung6. Bildschirmdump der TENT-Control-GUI mit Visualization-Tool Properties und Steuerelementen

Die Komponenten-Bean wird vom MCP kreiert (siehe Abschnitt 3.2) und bekommt als Initialisierungparameter die CORBA-Referenz auf die durch sie repräsentierte Komponente übergeben. Handelt es sich um eine spezielle Komponenten-GUI, so castet sie die Referenz mittels der `CORBA::Object::_narrow` in den tatsächlich erwarteten Typ um und erhält so Zugriff zu den implementierungsspezifischen Interfacemethoden. Im zweiten Schritt fragt die Komponenten-GUI das Property-Objekt nach einer Eigenschaft "`PropertyBean`" ab. Diese Eigenschaft kann den Namen einer speziellen GUI zur Darstellung des Property-Objektes dieser Komponente enthalten. Falls diese Eigenschaft nicht existiert oder das Instanziieren der darin angegebenen Property-Bean-Klasse fehlschlägt, wird eine Standard-Property-Bean instanziiert, welche die Referenz auf das Property-Objekt als Initialparamter übergeben bekommt. Die Property-Bean kann nun die Meta-Daten des Property-Objektes mittels `describe` abfragen und entsprechende Editoren für die Darstellung und Veränderung der einzelnen Properties instanziieren. Auch dabei hat die Komponente wiederum die Möglichkeit über das Vorhandensein spezieller Properties

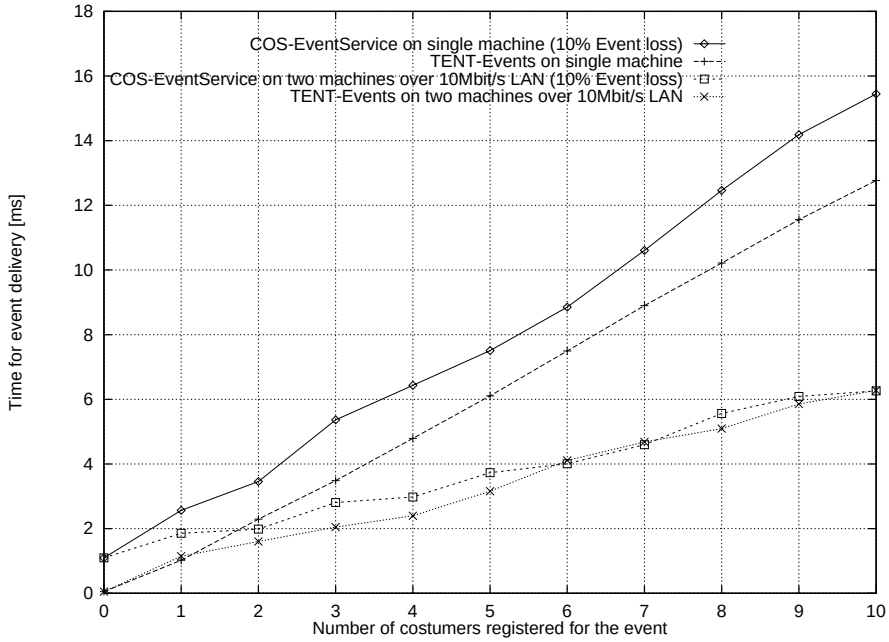

Abbildung7. Performancevergleich von COS-Event-Service und TENT-Eventmodell (gemessen auf Sun UltraSparc 1 für die Einzelmaschinenmessungen und zusätzlich einer Sun Ultra Enterprise für die Event-Consumer der verteilten Messungen

für bestimmte Typen spezielle Editoren zu registrieren. Um die Netzbelastung möglichst gering zu halten, greifen die Editoren initial nicht selbständig auf die ihnen zugeordnete Property zu, sondern werden von der Property-Bean initialisiert. Die Property-Bean registriert sich abschließend beim Property-Object als `PropertyChangeListener`. Damit ist die Initialisierung der Property GUI beendet.

Dieser Mechanismus ermöglicht die sehr flexible Anbindung beliebiger TENT-Komponenten in das zentrale Steuerungs- und Konfigurations-GUI. Im einfachsten Fall muß die Komponente keinerlei zusätzliche Funktionalität für diese Anbindung bereitstellen. Dies ist einer der Schlüsselmechanismen für das Integrieren von neuen TENT-Komponenten.

4 Anwendungserfahrungen

Momentan existiert eine erste prototypische Implementierung von TENT auf Basis des Visibroker der Inprise Inc. Eine Portierung auf GNU-Software unter GPL[9] ist in Arbeit. Das Konzept erwies sich als implementierbar und leistungsfähig.

[9] GPL - GNU General Public License

Eine wesentliche Abweichung von den durch die OMG standardisierten Diensten stellt das Benutzen des TENT-Eventmodells im Gegensatz zum OMG-Event-Service dar. Abgesehen von der wesentlichen einfacheren Logik zeigt ein direkter Vergleich mit der Event-Service-Implementierung von Visigenic, daß die direktere Kopplung auch Performancevorteile aufweist (siehe Abbildung 7). Die andererseits verlorengegangene Entkoppelung von Event-Sender und -Listener ist durch die Aufnahme der Definition von asynchronen Events wieder hergestellt worden. Ein wesentlicher und systemnotwendiger Vorteil ist die garantierte Zuverlässigkeit des TENT-Eventmodells. Die durchgeführten Leistungstest mit der COS-Implementierung ergaben bei sehr hohen Eventaufkommen eine Event-Verlustrate von durchschnittlich 10%, was auf eine Implementierung mittels der durch CORBA definierten 'oneway' Methoden schließen läßt. Für diese Methoden ist die sichere Zustellung zum entfernten Objekt nicht garantiert.

5 Andere Arbeiten

Die Arbeit, die in diesem Artikel präsentiert wurde, ist eine Synthese der Ergebnisse verschiedener Forschungsgebiete und Software-Engineering-Methoden. Wesentliche Beiträge bei der Benutzung von Komponentensoftware auf dem Gebiet der verteilten integrierten Produktentwicklung und -simulation wurden von den Systemen COVISE [12], CEASAR [2] und SPINEware [11] geleistet. Alle diese Projekte demonstrierten mit Hilfe von wirklichen Applikationen, das die heute verfügbare Computer- und Netzwerkinfrastruktur es möglich macht, die Entwicklung von Simulationssystemen im HPCN Bereich anzugehen. Die Projekte benutzten dabei nicht CORBA sondern eine jeweils speziell entwickelte Middleware, über die auch die nicht javabasierten GUIs angekoppelt wurden. Umfangreiche Forschungen wurden in den letzten Jahren auf dem Gebiet der generellen Organisation von Komponenten-Frameworks durchgeführt. Ein bedeutetender Beitrag wurde von GLOBUS [9] geleistet. GLOBUS ist eine komplette Metacomputing-Infrastruktur die auf einer Middleware namens NEXUS [8] basiert. Im besonderen wurden auch Probleme bezüglich Directory Services und Data Access Services untersucht, die gegenwärtig nicht Bestandteil von der TENT Spezifikation sind, jedoch in der weiteren Entwicklung mit einfließen werden. Das Design einer CORBA-Komponentenarchitektur wird momentan intensiv von der OMG diskutiert [4–6].

6 Zusammenfassung

Dieser Artikel präsentierte die Anwendung von Java im verteilten Simulationssystem TENT. Ausgehend von den Problemen, die CFD-Simulationssysteme heute aufweisen, wurde gezeigt, das der komponentenbasierte Ansatz von TENT diese Probleme lösen kann. Der Einsatz von Java ist dabei besonders für die Implementierung von leistungsunkritischen Systemkomponenten interessant, da sie aus den speziellen Eigenschaften wie Plattformunabhängigkeit, Run Time Type Information und einfach GUI-Realisierbarkeit Vorteile ziehen können.

Im speziellen wurde beschrieben, wie die Komponentenarchitektur von TENT organisiert ist und wie die in Java implementierte MCP-Komponente und die GUI-Controls der Applikationskomponenten umgesetzt wurden und über CORBA mit dem System kommunizieren.

TENT wird gegenwärtig im Rahmen des SUPEA Projektes des BMBF entwickelt. Der industrielle Hintergrund dieser Anwendung im Bereich des Flugzeugbaus, welcher durch die Projektpartner BMW/RR, MTU und DASA in das Projekt hineingetragen wird, stellt die Praxistauglichkeit von Java auch in diesem Bereich unter Beweis.

Die zukünftigen Ziele mit TENT sind die Erweiterung auf interdisziplinäre Simulationen, bei denen mehrere Simulation-Engines in einer Simulationsumgebung verschieden physikalische Phänomene gekoppelt simulieren. Unsere Vision ist ein System, welches den kompletter Entwurf eines Flugzeugteiles inklusive seines virtuellen Test beinhaltet.

Literatur

1. Thomas Breitfeld and Sven Kolibal. Tent: A corba based component architecture for mpi-parallel cfd simulation systems and their supporting tools. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*. C.S.R.E.A., Athens, Georgia, 1998.
2. CAESAR. Clusters of computational intensive applications for engineering, design, and simulation on scalable parallel architectures. *http://www.telecall.co.uk/ srcbae*, 1998.
3. The common object request broker: Architecture and specification. revision 2.1. OMG, July 1997.
4. Corba components, joint initial submission. OMG TC Document orbos/97-11-24, November 1997.
5. Corba components model, multiple interfaces and composition, inline software corporation. OMG TC Document orbos/97-12-08, December 1997.
6. Corba components model, rogue wave software, inc. OMG TC Document orbos/97-11-35, November 1997.
7. Corbaservices: Common object services specification. OMG, July 1997.
8. I. Foster, C. Kesselman, and S. Tuecke. The nexus approach to integreting muktithreading and communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1997.
9. Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. http://www-fp.globus.org/documentation/papers.html.
10. Graham Hamilton, editor. *JavaBeans API Specification*. Sun Microsystems Inc., Mountain View, CA., 1.01 edition, July 1997.
11. NLR. Network middleware for product engineering. *http://www.nlr.nl/public/-facilities/c867a*, 1997.
12. RUS. Introduction to covise. *http://www.hlrs.de/structure/organisation/vis/-covise/index.html*, 1998.
13. Ron Zahavi and Thomas J. Mowbray. *The essential CORBA: Systems Integration Using Distributed Objects*. John Wiley & Sons, New York, August 1997.