Humboldt University Berlin
Computer Science Department
Systems Architecture Group

Rudower Chaussee 25
D-12489 Berlin-Adlershof
Germany

Phone: +49 30 2093-3400
Fax: +40 30 2093-3112
http://sar.informatik.hu-berlin.de

DistSim
Eine verteilte Umgebung zur Durchführung von parametrisierten Simulationen

HU Berlin Public Report
SAR-PR-2007-03

February 2007

Author(s):
Ulf Hermann

Humboldt-Universität zu Berlin,
Institut für Informatik,
Lehrstuhl für Systemarchitektur

Betreuer:	Mathias Kurth, Anatolij Zubow
Gutachter:	Prof. Jens-Peter Redlich
Abgabedatum:	1. Februar 2007

DistSim

Eine verteilte Umgebung zur Durchführung von parametrisierten Simulationen

Studienarbeit vorgelegt von

Name:	Ulf Hermann
Matrikelnummer:	185523
E-Mail:	uhermann@informatik.hu-berlin.de

Zusammenfassung

DistSim ist ein System zur Verteilung von Simulationsaufgaben auf mehrere Rechner in einem Netzwerk. Es besteht aus einer Client-Anwendung zur Definition und Überwachung der Aufgaben, einem Server ("Wrapper"), der die eigentliche Simulation ausführt und einer Java-Bibliothek, die eine objekt-relationale Abbildung zur Speicherung von Resultaten in einer Datenbank vornimmt.

Inhaltsverzeichnis

1 Einleitung

1.1 Simulation

Computergestützte Simulation ist eine Methode, die in vielen Bereichen der Wissenschaften intensiv eingesetzt wird. Insbesondere die Simulation von Computer-Netzwerken bietet eine effiziente Möglichkeit, neue Protokolle zu testen und Abhängigkeiten zwischen äußeren Faktoren, der Netzwerkstruktur und Parametern des Protokolls empirisch zu untersuchen, ohne dass ein physisches Netzwerk aufgebaut werden muss. Es existieren verschiedene Simulatoren, die solche Experimente in unterschiedlichen Detailgraden ermöglichen, zum Beispiel NS2 (`http://www.isi.edu/nsnam/ns/`, Breslau u. a. (2000); Riley u. a. (2000)), JNS (`http://jns.sourceforge.net`) oder JiST (`http://jist.ece.cornell.edu/`, Haas und Barr (2005); Barr u. a. (2005a, b); Barr (2004); Barr u. a. (2004)). Üblicherweise ist es nötig eine große Zahl von Simulationsexperimenten auszuführen, um einen bestimmten Sachverhalt abschließend zu untersuchen. Die genannten Simulatoren sind jedoch zunächst - und zurecht - nur auf die Bewältigung je einer einzelnen Simulationsaufgabe ausgerichtet.

1.2 Verteilung

Um mehrere Experimente durchzuführen, ist es also nötig, den Simulator wiederholt und mit unterschiedlichen Parametern zu starten. Dabei ist die Nutzerin darauf angewiesen, den Überblick über die verwendeten Parameter und erzeugten Resultate zu behalten, um bei der anschließenden Analyse eine Zuordnung vorzunehmen. Zudem sind Simulationen oft von langen Laufzeiten geprägt. Es ist nicht selten, dass es Tage und Wochen dauert, bis verwertbare Ergebnisse vorliegen. Sinnvoll ist also eine Verteilung der Simulationsaufgaben auf mehrere Rechner, so dass mehrere Simulationen parallel ausgeführt werden können. Eine Automatisierung dieser Organisationstätigkeiten lässt einen beträchtlichen Zuwachs in der Produktivität der Anwender erwarten. Dieser Gedanke führte zu dem vorliegenden System zur Verteilung von parametrisierten Simulationen.

1.3 Sprachliche Hinweise

Männliche und weibliche Formen von Substantiven werden in dieser Arbeit weitgehend synonym verwendet. Sollte nur ein bestimmtes Geschlecht gemeint sein, so wird das speziell kenntlich gemacht. Damit entfallen Konstruktionen wie “der Nutzer/die Nutze-

rin" oder "der/die NutzerIn". Stattdessen werden männliche und weibliche Formen in ungefähr gleicher Zahl benutzt.

2 Anforderungen

Im Folgenden wird die Konzeption und Realisierung eines verteilten Systems für Netzwerksimulationen beschrieben. Zunächst sind die Anforderungen an ein solches System zu bestimmen. Insbesondere soll dabei JiST als primäre Simulationsplattform berücksichtigt werden. Diese Aufgabe lässt sich in folgende Teilbereiche untergliedern.

2.1 Umgebung

Die primäre Simulationsumgebung ist der Simulator JiST. Andere Simulatoren müssen jedoch ebenfalls einsetzbar sein und klare Schnittstelle zwischen dem Simulator und anderen Teilen des Systems sind erforderlich. Das Konzept der Studie mit parametrisierten Simulationen muss umgesetzt werden. Eine Studie ist eine Menge von ähnlichen Simulationen, die mit der selben Code-Basis aber mit unterschiedlichen Parametern ausgeführt werden. Die Speicherung der Parameter und Resultate soll in einer Datenbank erfolgen. Konkret wird hier MySQL verwendet.

2.2 Verteilte Ausführung von Simulationen

Das System soll effiziente Möglichkeiten bieten, Simulationen auf verschiedene Rechner zu verteilen. Dabei müssen Flaschenhälse, zum Beispiel bei der Speicherung der Resultate, nach Möglichkeit vermieden werden. Daneben wird eine zentrale Steuerungsinstanz entwickelt benötigt, mit der Simulationen konfiguriert und überwacht werden können. Synchronisationsmechanismen sollten sich, soweit wie möglich möglich, auf die Datenbank als zentrale Stelle stützen.

2.3 Parameterdefinition

Als Parameter sind einfache Paare aus Namen und Werten ausreichend. Eine grafische Oberfläche zur Spezifikation von Simulations-Parametern und deren Abhängigkeiten ist notwendig. Diese soll ein grafisches Verfahren zur Spezifikation abhängiger und unabhängiger Parameter bieten. Durch Kombination der Parameter soll pro Studie eine Menge von Simulationskonfigurationen erzeugt werden. Die Simulationskonfiguratio-

nen sowie weitere Konfigurationen für den eigentlichen Simulator (beispielsweise log4j-Properties) und Referenzen auf den verwendeten Code sollten für spätere Nachvollziehbarkeit in der Datenbank abgelegt werden. Der Simulator muss seine gesamte Konfiguration aus der Datenbank beziehen und keinen persistenten Zustand, beispielsweise in Dateien, halten.

2.4 Persistentes Speichern der Ergebnisse

Ein generisches und effizientes Verfahren, um Simulationsergebnisse direkt von den Simulationen in die Datenbank zu schreiben, war zu entwickeln. Das Format der Resultate soll der Nutzerin möglichst wenig Beschränkungen auferlegen. Optimalerweise sollen jegliche Datentypen, die die Programmiersprache bietet, akzeptiert werden und die nötigen Tabellen ohne Nutzerinteraktion generiert werden. Auf diese Art sollte eine direkte Zuordnung von Parametern und Resultaten in der Datenbank ermöglicht werden.

2.5 Überwachung der Simulationen

Zur Überwachung des Fortschritts der Simulationen wird ein grafisches Werkzeug benötigt. Dies soll die Einsichtnahme in schon eingetragene Ergebnisse, sowie die Anzeige des jeweiligen Fortschritts der Simulationen ermöglichen.

3 Bisherige Ansätze

3.1 Manuelle Lösungen

Die nächstliegende Lösung für das Problem der Organisation und Verteilung von Simulationen ist eine Sammlung von Shell-Skripten, mit denen Parameter für Simulationen erzeugt und die Simulationen ausgeführt werden konnten. Der Vorteil dieses Ansatzes liegt in seiner Flexibilität - Parameterkombinationen können ohne große Mühe und ganz nach den jeweiligen Anforderungen erzeugt werden. Als Nachteil stellte sich jedoch heraus, dass es keine zentrale Stelle gibt, an der die Ergebnisse der Simulation gesammelt und verglichen werden können. Durch die wiederholten Experimente ergibt sich vielmehr eine große Zahl an Dateien in unterschiedlichen Formaten, die zunächst mit verschiedenen Methoden konvertiert, zusammengeführt und visualisiert werden müssen. Zudem ist es schwierig, zusammengehörige Parameter, Resultate und Code-Bestandteile nach mehrmaliger Simulation zu identifizieren. Schließlich ist dieses Verfahren auch nur schwer

dazu geeignet, eine automatische Verteilung der Simulationen auf mehrere Rechner im Netzwerk zu gewährleisten.

3.2 DUCKS

Aus diesen Gründen wurde DUCKS (Kargl 2006) evaluiert. DUCKS verwendet die in JiST eingebaute Server-Funktion, was leider dazu führt, dass sich DUCKS ausschließlich mit JiST als Simulator betreiben lässt. DUCKS verteilt die Simulationen allerdings auf verschiedene Rechner und sammelt später die Ergebnisse wieder an zentraler Stelle. Dabei wird eine zentrale Kontrollinstanz - der "Controller - verwendet, der Simulationsparameter aus Konfigurationsdateien einliest und Aufgaben an die JiST-Server verteilt. Nach Abschluss der Berechnungen geben die JiST-Server die Ergebnisse der Simulation als Java-Properties an den Controller zurück, der diese filtert und in eine Datenbank-Tabelle einträgt.

Dieses Vorgehen bringt jedoch bei der gewählten Implementation auch einige Probleme mit sich. Es fällt schnell auf, dass der zentrale Controller die Ausführung jeder einzelnen Simulation anstoßen und somit den JiST-Servern faktisch Rechenzeit zuweisen muss. Um dies effizient zu gestalten, wurde die Implementation des Controllers sehr komplex. Die Parameter für Simulationen werden zudem nicht in der Datenbank abgelegt, sondern in Dateien. Das Format der Parameterdateien bildet eine eigens definierte Sprache, die wiederum in Java-Properties eingebettet ist. Diese Sprache ist sehr mächtig was die Möglichkeiten der Parameterdefinition angeht, die Handhabung ist jedoch umständlich. Bei Verwendung einer Datenbank könnte hier die mächtigere Tabellen- und Tupel-Struktur statt der einfachen Name-Wert-Struktur der Properties genutzt werden, um eine erheblich weniger komplexe Sprache mit der gleichen Funktion zu definieren. Zudem werden die Parameter und Resultate der Simulationen bei Verwendung von DUCKS an getrennten Stellen gehalten. Dies entspricht nicht dem Ziel, eine feste Zuordnung von Parametern und Resultaten zu erreichen. Weiterhin werden die Resultate am Ende jeder Simulation als Java-Properties aus deren Standardausgabe gelesen, die auch noch anderen Text enthalten kann. Die eingeschränkte Syntax der Java-Properties führt dabei zu Problemen, indem sich komplex strukturierte Ergebnisse nur schwer in flachen Name/Wert-Paaren ausdrücken lassen, und die Übertragung stellt sich als fehleranfällig und langsam heraus.

3.3 LINDA Tuple Space

LINDA Tuple Space (Gelernter 1985) ist ein abstraktes Verteilungsmodell. LINDA definiert einen gemeinsamen Tupel-Raum für eine Menge von Prozessen. Dieser kann mit den Operationen “in”, “out” und “read” gelesen und modifiziert werden. Tupel sollen dabei elementare Daten darstellen. Es gibt eine Reihe von Implementierungen der LINDA-Prinzipien für verschiedene Programmiersprachen und Zwecke. In der vorliegenden Aufgabenbeschreibung hätte eine solche verwendet werden können, um beispielsweise Parameter und Resultate für Simulationen zu übertragen. Eine solche Verwendung bietet jedoch keine unmittelbaren Vorteile gegenüber einer direkteren Herangehensweise. Statt dessen hätte die Verwendung zu zusätzlicher Komplexität geführt, da sämtliche Parameter und Resultate irgendwann aus dem Tupel-Raum in die Datenbank überführt werden müssten. Die Datenbank ist zwar ebenfalls relational, und damit tupel-orientiert, allerdings unterliegen Tupel in der Datenbank den Einschränkungen der Tabelle zu der sie gehören. Dies ist im Linda Tuple Space nicht der Fall. Daher wäre eine Übersetzung nötig geworden.

4 Architektur

Wie bereits in den Anforderungen dargestellt, besteht das System aus einem Werkzeug zur Konfiguration, Überwachung und Auswertung, sowie einem Wrapper (vgl. Abb 1). Der Wrapper, die “Verpackung” für JiST-Simulationen, holt die Simulations-Parameter selbsttätig aus einer Datenbank. Er startet, überwacht und beendet die Simulationen, deren Lebenszyklen damit vollständig durch den Wrapper bestimmt werden. Jeder Wrapper verwaltet zu jedem Zeitpunkt höchstens eine Simulation. Parallelität wird erreicht indem mehrere Instanzen des Wrappers gleichzeitig verwendet werden. Diese können sowohl auf dem selben, als auch auf verschiedenen phyiskalischen Rechnern gestartet werden. Das Konfigurations- und Überwachungswerkzeug stellt Funktionen bereit, die die Definition der Parameter für Simulationen, sowie deren Überwachung und eventuellen Abbruch ermöglichen. Die Überwachung findet durch Beobachtung der schon in der Datenbank vorhandenen Resultate statt. Der Abbruch von Simulationen kann mittels der Funktionen einer am Wrapper vorhandenen RMI-Schnittstelle erreicht werden. Unterstützend steht noch eine Java-Bibliothek zur Abbildung von Java-Objekten in eine Datenbank zur Verfügung, die helfen soll, Resultate zu speichern. Die Verwendung dieser Komponente ist jedoch im Hinblick auf das gesamte System optional, da die Simulation

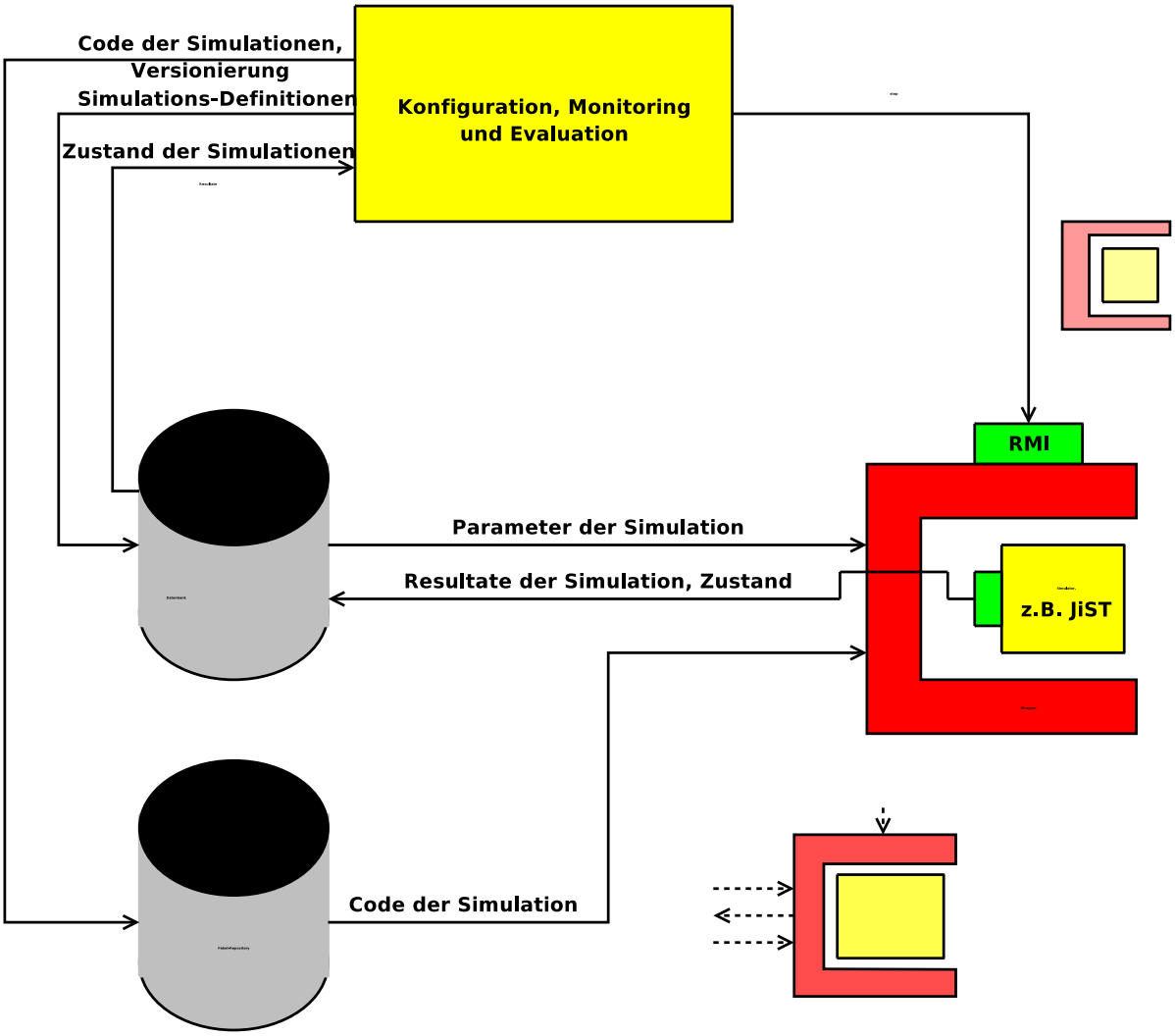

Abbildung 1: Übersicht der Architektur

die Datenbank auch direkt über den Datenbank-Treiber oder mit anderen Methoden verwenden kann.

4.1 Konfiguration und Überwachung von Simulationen

Die wesentliche Funktion des Konfigurations- und Überwachuns-Werkzeugs ist eine editierbare, tabellarische Auflistung von Simulationsparametern, über die Gruppen von Simulationen mit Wertebereichen für Parameter konfiguriert werden können. Das Resultat wird als Menge von Tupeln in eigenen Tabellen in der Datenbank gespeichert. Zusätzlich kann der von den Instanzen des Wrappers dokumentierte Fortschritt der einzelnen Simulationen beobachtet und die Wrapperinstanzen können direkt über ihre RMI-Schnittstellen angesprochen werden. In diesem Zusammenhang stellt das Überwachuns-Werkzeug auch Funktionen bereit, mit denen nicht mehr benötigte Simulationen komplett aus der Datenbank gelöscht werden können.

4.1.1 Parameterdefinition

Wie bei DUCKS werden Simulationen in "Studien" zusammengefasst. Eine Studie definiert die zu verwendenden Code-Pakete und die Befehlszeile für die Ausführung der Simulationen. Außerdem enthält eine Studie "Gruppen" von Parameterdefinitionen. In einer Gruppe wird für jeden Parameter, der den Simulationen übergeben werden soll, ein Wertebereich angegeben. Dies geschieht entweder als eine Auflistung von Literal-Konstanten oder durch Angabe einer oberen und unteren Grenze, sowie eines Intervalls. Obere und untere Grenzen sowie Intervalle können als beliebige mathematische Ausdrücke angegeben werden. Bei der Angabe von mathematischen Ausdrücken können auch andere Parameter referenziert werden.

4.1.2 Parameterauswertung

Durch Auswertung der Ausdrücke wird für jeden Parameter eine Menge von Werten erzeugt. Die Auswertung erfolgt mit Hilfe des "Java Expression Parser" JEP (`http://www.singularsys.com/jep/`). Die Parameter werden dabei nach Abhängigkeiten sortiert. Parameter, die keine Referenzen auf andere Parameter enthalten sind unabhängig und werden als erstes aufgenommen. Parameter die ausschließlich von schon aufgenommenen Parametern abhängen werden hinter diesen angeordnet. Nach diesem Verfahren können alle Parameter sortiert werden, die keine zirkulären Abhängigkeiten aufweisen. Anschließend wird eine dem Kreuzprodukt ähnliche Menge über die Wertebereiche aller

Parameter gebildet (vgl. Abb. 2). Dies geschieht indem ein Parameterbaum aufgebaut wird. Die erste Ebene des Baumes umfasst Knoten für jeden der Werte des ersten, und somit völlig unabhängigen Parameters. Die n-te Ebene umfasst für jeden Knoten aus der (n-1)-ten Ebene weitere Knoten. Diese repräsentieren jeweils einen möglichen Wert des n-ten, also möglicherweise abhängigen, Parameters. Mathematische Ausdrücke werden dabei für jeden Knoten der (n-1)-ten neu ausgewertet. Jedes Blatt des Baumes repräsentiert schließlich eine vollständige Menge von Parametern, die für eine Simulation verwendet werden.

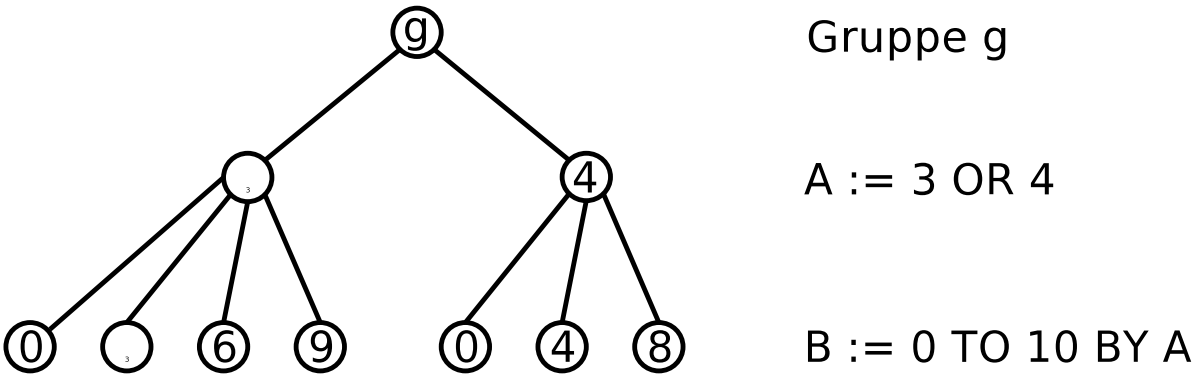

Abbildung 2: Auswertung der Gruppenparameter

4.1.3 Paketverwaltung

Das Paketsystem erlaubt eine Zuordnung von Code-Paketen zu Parametern und Resultaten von Simulationen. So ist gewährleistet, dass die Nutzerin den Überblick darüber behält, welcher Quellcode zu welchen Ergebnissen geführt hat. Zu diesem Zweck können mittels einer Datenbank-Tabelle Pakete definiert werden. Diese bestehen aus einem Namen, einer Version, der Zielarchitektur und der URL, über die der tatsächliche Code verfügbar ist. Es wird dabei erwartet, dass der Inhalt, der über die URL zur Verfügung steht, gleich bleibt. Es bietet sich zum Beispiel an, einen FTP-Server, in dem regelmäßige Schnappschüsse einer Code-Basis abgelegt werden, als Paket-Speicher zu verwenden. Die Zielarchitektur wird als einfache Zeichenkette angegeben, die sich in der Konfiguration des Wrappers wiederfindet. Die Architektur ist nicht nötig, wenn architekturunabhängige Sprachen, wie Java, für die Simulation verwendet werden. Für diesen Fall ist die spezielle Architektur “all” vorgesehen. Nachdem das System aber auch Simulatoren in anderen Sprachen unterstützen soll, ist die Möglichkeit der Definition von Zielarchitekturen hier gegeben. Eine weitere Tabelle ordnet Pakete konkreten Studien zu. Hier wird die

Studien-ID-Nummer, der Name und die Version des Paketes referenziert und ein Verzeichnis definiert, in das das Paket entpackt werden soll.

4.1.4 Die Konfigurations-Bibliothek

Die Funktionen der Parameterdefinition und Paketverwaltung sind sowohl über die grafische Oberfläche des Konfigurations- und Überwachuns-Werkzeugs als auch über eine Java-Bibliothek verfügbar. Im Rahmen dieser Bibliothek kann die Funktionalität auch ohne grafische Oberfläche und für die automatisierte Definition von Simulationen verwendet werden. Auch kann beispielsweise der Algorithmus zur Erzeugung der Simulationen aus Gruppen-Parametern abgewandelt werden. Dies erlaubt dem Nutzer, Simulationen nach Belieben zu Gruppen und Studien zu bündeln, ohne dabei auf das beschriebene Verfahen Rücksicht zu nehmen.

4.2 Der Wrapper

Für jeden Simulationsprozess, der simultan ablaufen kann, wird ein spezieller Server gestartet, der die Kontrolle über eine Reihe sequentiell ausgeführter Simulationen übernimmt. Dieser "Wrapper" hat die vollständige Kontrolle über seinen jeweiligen Simulations-Prozess und dokumentiert dessen Fortschritt in der Datenbank. Er kann benutzt werden, um fehlerhafte Simulationen abzubrechen. Zusätzlich kann der Wrapper selbst beendet werden, wenn er keine weiteren Simulationen mehr annehmen soll. Diese Funktionalität wird über ein RMI-Schnittstelle exportiert, so dass das Überwachungs-Werkzeug darauf zugreifen kann. Vor Beginn der Simulation liest der Wrapper Konfigurationsdateien und Code-Pakete aus der Datenbank und kopiert bzw. entpackt diese in das lokale Dateisystem. Der Wrapper wird nur Code laden, dessen Architektur entweder seiner eigenen oder der speziellen Architektur "all" entspricht. Er prüft jedoch nicht, ob die in der Konfiguration angegebene Architektur den Angaben entspricht, die beispielsweise der virtuellen Java-Maschine entnommen werden könnten. Dies kann für die Definition eigener, virtueller Architekturen genutzt werden. So kann es beispielsweise nötig sein, auf Rechnern, die Java 1.5 unterstützen anderen Code auszuführen als auf solchen die nur Java 1.4 unterstützen. Es wird zudem erwartet, dass die Pakete als im "zip"-Format gepackte Dateien vorliegen. Sobald die Umgebung eingerichtet ist, werden die Parameter für die Simulation aus der Datenbank gelesen und als Java-Properties formatiert. Anschließend wird der Simulationsprozess mit der in der Datenbank angegebenen Befehlszeile gestartet und diesem die Parameter über die Standard-Eingabe

übergeben. Eventuell auftretende Dateien mit Simulationsresultaten werden nach dem Ende der Simulation von dem Wrapper in der Datenbank gespeichert. Anschließend werden alle Konfigurations- und Resultat-Dateien sowie alle Code-Pakete wieder gelöscht. Mittels Transaktionen in der Datenbank wird sichergestellt, dass jeder Wrapper eine eigene Simulation bekommt und keine Simulation mehreren Wrappern zugewiesen wird.

Abgebrochene Simulationen werden in der Datenbank als solche gekennzeichnet. Die bis zum Abbruch aufgelaufenen Resultate werden nur auf expliziten Wunsch des Nutzers wieder gelöscht. Das Löschen der durch die OR-Abbildungs-Bibliothek angelegten Daten kann durch kaskadierende Fremdschlüsselbescränkungen leicht bewerkstelligt werden. Für das Löschen anderer Daten muss die Nutzerin selbst sorgen. Da die Daten nicht automatisch gelöscht werden, stehen Informationen bereit, aus denen Gründe eines unabsichtlichen Abbruchs erschlossen werden können. Zudem erhält der Nutzer die Möglichkeit eine Simulation abzubrechen, wenn keine neuen Erkenntnisse mehr erwartet werden und die bisherigen Daten aber durchaus wertvoll sind.

4.3 Objekt-Relationale Abbildung

Die Abbildung von Objekten auf Datenbank-Tabellen zur Speicherung von Resultaten ist in einer Hilfsbibliothek für die Simulation implementiert. Als Sprache wurde Java gewählt, weil die Mehrzahl der verfügbaren Simulatoren in Java geschrieben sind. Die eigentliche Umsetzung von Java-Klassen in Datenbank-Tabellen zur Laufzeit nimmt dabei die Hibernate-Bibliothek vor. Hibernate ist eine allgemeine Bibliothek für objekt-relationale Abbildung und ist im Internet unter `http://www.hibernate.org` zu finden. Hibernate kann Java-Objekte in Datenbanktabellen speichern, wobei einer Beschreibung der entsprechenden Klassen im Format eines “Mappings” gefolgt wird. Die Erzeugung dieses Mappings sollte in diesem Fall möglichst automatisch geschehen, damit dem Nutzer am Ende eine einfache “save”-Methode zur Verfügung gestellt werden kann und er sich nicht um die Einzelheiten kümmern muss.

Diese Funktionalität wird mit Hilfe einer abgewandelten Version der Hibernate-Extensions (`http://sourceforge.net/projects/hibernate/` realisiert. Die Hibernate-Extensions sind ein Projekt, das eine Erweiterungs-Bibliothek für Hibernate bereitstellt. Diese Bibliothek kann Hibernate-Mappings aus Java-Bytecode, Java-Quellcode aus Hibernate-Mappings oder Hibernate-Mappings aus Datenbanktabellen semi-automatisch erzeugen. Der verwendete Ansatz beschränkt sich darauf, einen sinnvollen Ausgangspunkt für manuelle Anpassungen zu liefern und der Nutzerin repetitive Tätigkeiten abzunehmen. Die hier benötigte Funktionalität ist weiter gefasst. So ist es

nötig eine möglichst vollständige und in jedem Fall korrekte Abbildung vorhandener Java-Klassen zu erzeugen. Es kommt dabei weniger darauf an, komplizierte Datenstrukturen besonders geschickt abzubilden und alle Finessen der Hibernate-Bibliothek zu verwenden. Zu diesem Zweck wurde "class2hbm", das Modul der Hibernate-Extensions, das ein Mapping aus Klassen erzeugt, angepasst und auf die neueste Version von Hibernate portiert.

4.4 Datenbank-Schemata

Für die Definition der Parameter ist ein Datenbank-Schema notwendig. Die Definition der Parameter soll einerseits in einer für den Nutzer semantisch sinnvollen Struktur erfolgen, andererseits soll diese Struktur jedoch für jede Art von Simulation anpassbar sein. Dies geschieht mit den Tabellen "studies", "groups" und "simulations", die die grundlegenden Parameter der jeweiligen Elemente halten. Außerdem werden die Tabellen "group_numeric_params" für Gruppenparameter in Form von Ausdrücken, "group_string_params" für Gruppenparameter in Form von Literalen und "params" für Simulationsparameter benötigt. Für Konfigurations- und Resultat-Dateien sowie Code-Pakete werden die Tabellen "group_in_files", "group_out_files" und "study_packages" verwendet. Die Tabellen sind mit Hilfe von Fremdschlüsseln unter einander und mit den Tabellen der Resultate-Datenbank verknüpft, so dass einerseits die Zusammenhänge zwischen den einzelnen Elementen rekonstruiert werden können und andererseits keine versehentliche Löschung noch benötigter Einträge erfolgen kann. Diese Verknüpfung ist jedoch nur möglich, wenn die betroffenen Tabellen auf dem selben Datenbank-Server liegen.

5 Benutzung

5.1 Installation

Die Installation des DistSim-Paketes beschränkt sich im Wesentlichen darauf, geeignete Datenbankschemata anzulegen und ein oder mehrere Instanzen des Wrappers zu starten. Der Client wird einfach mittels "ant run" in seinem Verzeichnis gestartet und die OR-Abbildungs-Bibliothek kann mittels "ant dist" gebaut und anschließend als normale Java Jar-Bibliothek verwendet werden. Zum Anlegen der Datenbankschemata steht ein SQL-Skript namens "tabellen" im Verzeichnis des Wrappers zur Verfügung. Es setzt eine MySQL-Datenbank, mindestens in Version 4.3, voraus und legt alle benötigten Tabellen

an. Als Konvention werden dabei zwei Schemata erzeugt: “simulation” für die Definitionen und Parameter, “simulation_results” für die Ergebnisse. Um den Client erfolgreich zu betreiben ist ein Datenbank-Nutzer nötig, der auf allen Tabellen in der Datenbank “simulation” Lese- und Schreib-Rechte hat, sowie Daten einfügen kann. Ein weiterer Nutzer mit Lese-Rechten für die Datenbank “simulation_results” ist ebenfalls nötig, um die Resultate im Überwachungs-Werkzeug anzuzeigen. Für die volle Funktionalität sollten auch Schreib-Rechte vorliegen, allerdings kann hierauf verzichtet werden, wenn es nicht nötig oder gewollt ist, dass Ergebnisse gelöscht werden. Für den Wrapper ist ein Benutzer nötig, der Lese-Rechte auf allen Tabellen der Datenbank “simulation” hat und zusätzlich Schreib-Rechte auf den Tabellen “simulations” und “hosts”. Ein weiterer Benutzer muss Schreib-Rechte auf der Tabelle “simulation_results.progress” haben, damit der Fortschritt der Simulation beobachtet werden kann. Soll eine Simulation mittels der OR-Abbildungs-Bibliothek Resultate zurückschreiben, so muss letzterer Benutzer auch Schreib- und Einfüge-Rechte für die entsprechenden Tabellen bekommen und Tabellen in “simulation_results” anlegen können. Natürlich kann für alle diese Aufgaben der selbe Benutzer verwendet werden, allerdings besteht dann kein Schutz vor versehentlichen oder durch Programmfehler hervorgerufenen Löschoperationen. Die Namen der Datenbanken sind ebenfalls konfigurierbar und die gegebenen Namen sind nur als Vorschläge zu betrachten. Eine Instanz des Wrappers wird nun durch einige Java-Properties konfiguriert, die über die Standard-Eingabe übergeben werden. Zunächst müssen nach folgendem Muster die Datenbankeigenschaften angegeben werden:

```
definitions.host amsel.informatik.hu-berlin.de
defintions.database simulation
definitions.username foo
definitions.password bar

results.host amsel.informatik.hu-berlin.de
results.database simulation_results
results.username bar
results.password foo
```

Außerdem müssen noch einige Eigenschaften des verwendeten Rechners und des Wrappers selbst bereitgestellt werden. Zu beachten ist, dass jede Instanz des Wrappers eine eindeutige ID-Nummer benötigt. Diese Host-Properties sollten nach folgendem Muster angegeben werden:

```
host.id 42
host.description irgendein Text
host.architecture x86_linux
host.name star.informatik.hu-berlin.de
```

Sind diese Daten angegeben und in geeigneten Dateien gespeichert, so kann der Wrapper beispielsweise mittels

```
cat db.properties host42.properties | ant run
```

gestartet werden. Damit die Properties korrekt übergeben werden, ist Ant mindestens in der Version 1.6.5 erforderlich. Frühere Versionen leiten die Standardeingabe nicht an den gestarteten Prozess weiter.

5.2 Definition einer Studie

Zur Definition einer Studie mit den damit verbundenen Simulationen kann das grafische Konfigurationswerkzeug verwendet werden. Zunächst muss hier die Datenbankverbindung angegeben werden, wozu die oben beschriebenen Datenbank-Nutzer verwendet werden sollten. Anschließend können neue Studien definiert, vorhandene in abgewandelter Form wiederverwendet, Code-Pakete definiert und letztere zu Studien hinzugefügt werden. Die Verzeichnisse, die hier angegeben werden, werden nach der Simulation vollständig gelöscht. Sie müssen für jede Instanz des Wrappers eindeutig sein. Dies wird unterstützt indem das oben beschriebene Ant-Skript vor dem Start ein eindeutiges Verzeichnis anlegt, in dem der Wrapper und die Simulation laufen. Solange relative Pfade angegeben werden, ist sichergestellt, dass diese für jede Instanz eindeutig sind. Zu beachten ist außerdem, dass nur die Studien modifizierbar sind, die auch in der momentanen Sitzung definiert wurden. Dies verhindert die nachträgliche Veränderung von bereits bestehenden und eventuell abgeschlossenen Studien. Zwar ist es grundsätzlich vorgesehen, dass Studien nachträglich erweitert werden können, doch destruktive Operationen, wie beispielsweise das Überschreiben eines Code-Paketes mit einem anderen, werden dadurch ausgeschlossen. Ist jedoch eine Studie bisher “leer”, so kann sie komplett wieder gelöscht werden. Ist dies nicht der Fall, so führt der Versuch zu einer Verletzung einer Fremdschlüsselbeschränkung in der Datenbank und somit zu einer Fehlermeldung.

Sind die grundsätzlichen Parameter einer Studie definiert, so kann die Parameterdefinition für die Gruppen und Simulationen vorgenommen werden. Hierbei gilt wieder, dass vorhandene Studien mit neuen Gruppen erweitert werden können, bestehende Gruppen

aber nicht verändert werden dürfen. Löschen von Gruppen ist wieder erlaubt, so lange keine aktiven oder abgeschlossenen Simulationen mit diesen Gruppen assoziiert sind, ansonsten wird die Datenbank einen Fehler melden. Zu beachten ist auch, dass aus den bestehenden Gruppen jederzeit neue Simulationen generiert werden können, auch wenn die selben Simulationen bereits existieren. So lässt sich eine redundante Ausführung der Simulationen erzielen.

Parameter werden grundsätzlich sortiert nach Typ angezeigt, wobei die Ausdrücke in der Tabelle über den Literalen stehen. Diese Sortierung hat keinen Einfluss auf die Abhängigkeitsanalyse bei der Generierung von Simulationen. Zusätzlich können Ein- und Ausgabe-Dateien definiert werden. So können beispielsweise log4j-Properties als extra Datei in die Datenbank geladen und über den Wrapper der Simulation übergeben werden.

Zusätzlich zur Parameterdefinition werden auch einfache Funktionen zur Überwachung von laufenden Simulationen, sowie zur Auswertung von Ergebnissen angeboten. Insbesondere die Auswertung kann jedoch nicht abschließend behandelt werden, da hierfür spezifische Methoden nötig wären, die je nach Art der Simulation variieren. Zu beachten ist, dass die Liste der überwachten Simulationen bei der Auswahl der Studie geladen wird, und somit keine Neuerzeugung von Simulationen notwendig ist, wenn lediglich der Fortschritt überwacht werden soll.

5.3 Konzepte der Konfigurations-API

Die Konfigurations-Bibliothek ist als Hierarchie von Datenbank-gestützten Objekten aufgebaut. Ein DB-gestütztes Objekt hat selbst Methoden und Daten und kann zusätzlich abhängige Objekte verwalten (vgl. Abb. 3). So ist beispielsweise eine Simulation abhängig von einer Gruppe, während die Befehlszeile ein direkter Parameter einer Studie ist. Jedes DB-gestützte Objekt kann in die Datenbank eingefügt, aus der Datenbank gelöscht oder in der Datenbank erneuert werden. Diese Operationen haben einen entsprechenden Einfluss auf alle von diesem Objekt abhängigen Unterobjekte. Abhängige Objekte können mittels der Methoden “add” und “remove” zu “Elternobjekten”, von denen sie abhängig sind, hinzugefügt und entfernt werden. Die Methode “commit” des Elternobjekts wird so vorgenommene Änderungen in die Datenbank übertragen. Bestimmte Klassen sind zudem in der Lage, mit “loadFromDb” eine Repräsentation des Zustands der Datenbank zu laden. Diese Operation lädt grundsätzlich die direkt abhängigen Objekte, jedoch nicht rekursiv. So wird das Laden einer Studie das Laden aller Gruppen, die zu dieser Studie gehören, auslösen. Die zu den Gruppen gehörenden Simulationen wer-

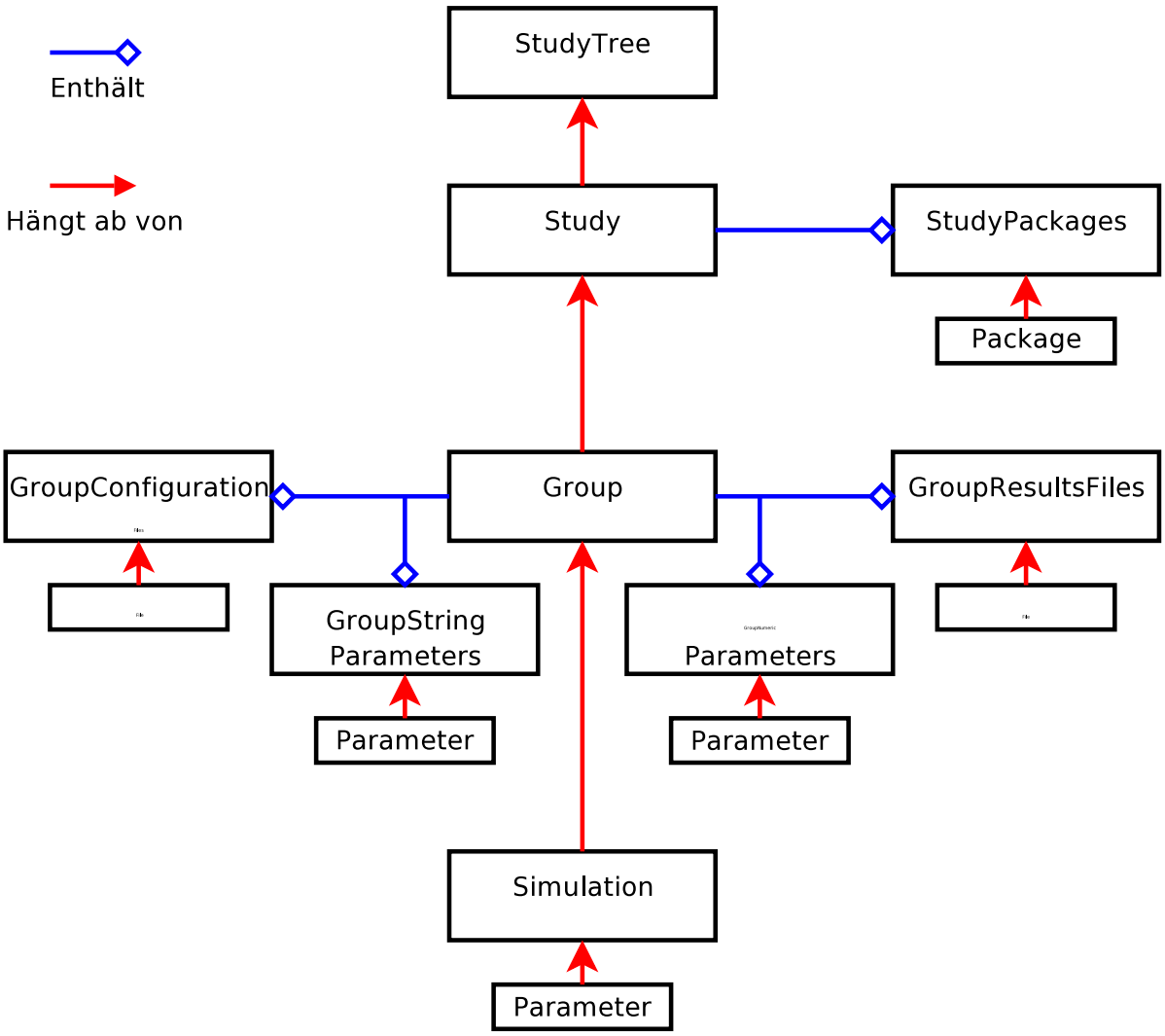

Abbildung 3: Klassen und Abhängigkeiten der Konfigurations-Bibliothek

den jedoch zunächst nicht geladen. Beim Laden eines Studienbaums, also einer Entität, die alle Studien mit einem bestimmten Namen repräsentiert, werden die dazu gehörigen Studien geladen, jedoch nicht deren Gruppen. Die selbe Funktion haben die Konstruktoren und "getDependent"-Methoden, die ein ResultSet als Parameter übernehmen. Die schreibenden Methoden, wie "commit" oder "performDbInsert", wirken jedoch rekursiv in beliebiger Tiefe. Der Grund für dieses Verhalten liegt darin, dass es zu aufwändig wäre, beim Laden eines Studienbaum-Objektes gleich alle Studien mit allen zugehörigen Simulationen und Parametern zu laden. Beim Schreiben wird jedoch eine überschaubare Auswahl an Objekten erwartet, die im Speicher vorliegen.

Gruppen und Studien besitzen außerdem noch direkt "enthaltene", aber nicht oben beschriebenen Sinne abhängige Unterobjekte, die benutzt werden, um die Abhängigkeiten zu strukturieren (vgl. Abb. 3). So sind die für eine Studie vorgesehenen Pakete zwar abhängige Objekte, jedoch über eine Zwischenklasse namens "StudyPackages" angebunden. Jedes Objekt der Klasse "Study" verweist dann auf genau ein Objekt der Klasse "StudyPackages" und umgekehrt. Ebenso sind die enthaltenen Objekte der Gruppen aufgebaut. Wird an Objekten der Klassen "Study" oder "Group" eine der oben beschriebenen Methoden aufgerufen, so wird dieser Aufruf, nach der oben beschriebenen Ausführung, auch an die enthaltenen Objekte weitergegeben.

Mit Hilfe der vorgestellten Methoden lassen sich nun leicht Studien erstellen und editieren. Dabei sollte jedoch beachtet werden, dass die Schutzmechanismen, die im grafischen Client eine ungewollte Veränderung bestehender Studien verhindern, hier nicht greifen. Es ist prinzipiell möglich Studien und Simulationen nach Belieben zu verändern. Ebenfalls erwähnenswert ist, dass jede Simulation, die in die Datenbank zurückgeschrieben wird, sofort zur Ausführung bereit steht. Daher sollte sichergestellt werden, dass alle Parameter der Studie bereits definiert und in die Datenbank geschrieben sind, wenn Simulationen zurückgeschrieben werden.

5.4 Benutzung der OR-Abbildungs-Bibliothek

Zur Interaktion mit der OR-Abbildungs-Bibliothek steht die Klasse DBSaver im Paket or_mapper.util zu Verfügung. Die OR-Abbildungs-Bibliothek wird, wenn mittels save(Object o, String entity) ein Objekt übergeben wird, eine Datenbank-Transaktion eröffnen und versuchen, das Objekt zu speichern. Falls dies nicht gelingt, wird die Transaktion zurückgerollt und der Grund untersucht. Liegt der Grund darin, dass eine Klasse noch nicht in das Hibernate-Mapping aufgenommen wurde, so wird diese neu aufgenommen und von vorne angefangen. Dabei wird die Objekt-Struktur rekursiv abgearbeitet.

Für jedes zu speichernde Objekt werden alle referenzierten Objekte und Objekte der Oberklassen mitgespeichert. Kreisförmige Abhängigkeiten zwischen Objekten werden dabei von Hibernate erkannt. Zusätzlich zu save(...), stehen auch noch die Methoden quickSave(...) und rebuildSession(...) zur Verfügung. Mit quickSave(...) können Objekte gespeichert werden, ohne eine neue Transaktion zu eröffnen. Wenn hier jedoch eine unbekannte Klasse angetroffen wird, so führt dies zu einem inkonsistenten Zustand der Datenbank und einer Ausnahmebedingung. Allerdings wird mit quickSave(...) eine erheblich kürzere Laufzeit benötigt, als mit save(...). Mit rebuildSession können einzelne Klassen in das Hibernate-Mapping aufgenommen werden, ohne dass sofort eine Instanz gespeichert wird. Bei all diesen Methoden wird die Java-Reflection Funktionalität intensiv genutzt. Um ein Objekt zu speichern, müssen seine Unterobjekte also nicht unbedingt "public" sichtbar sein.

In der Datenbank werden verschiedene Tabellen angelegt. Die wichtigste ist java_lang_Object. Hier wird eine generierte ID-Nummer, der Name der Laufzeit-Klasse des Objekts, die ID-Nummer der Simulation und die beim Aufruf übergebene "Entity" eingetragen. Für jedes gespeicherte Objekt wird in dieser Tabelle ein Eintrag angelegt. Zusätzlich wird für jede Klasse eine weitere Tabelle mit einem entsprechenden Namen angelegt. Diese Tabellen enthalten Spalten für die Objekt-ID, die sich auf die Tabelle java_lang_Object bezieht, und alle Attribute der Klasse. Ein zu speicherndes Objekt führt also zu Einträgen in der Tabelle seiner Laufzeit-Klasse und den Tabellen für alle Oberklassen bis java.lang.Object. Für Arrays und Collections werden zusätzliche Hilfstabellen angelegt.

Leider bringt dieses Verfahren einige Beschränkungen mit sich. So ist es unmöglich auf diese Art mehrdimensionale Arrays zu speichern. Der Versuch, dies zu tun, wird in einer Ausnahme enden. Allgemein können polymorphe Referenzen auf Arrays nicht abgebildet werden, weil es nicht möglich ist, eine Tabelle für ein Array als Unterklasse von java.lang.Object anzulegen. Einfache Arrays als explizite Attribute einer Klasse können jedoch problemlos gespeichert werden. Als weitere Beschränkung hat sich gezeigt, dass Hibernate das Speichern von java.lang.Class Objekten im polymorphen Kontext nicht erlaubt. Hier wird bedingungslos eine Ausnahme erzeugt. Das heißt unter Anderem, dass Class-Objekte, die in Arrays oder Collections vorkommen, nicht gespeichert werden können. Explizite, nicht polymorph referenzierte Attribute vom Typ Class können allerdings gespeichert werden, da Hibernate hier ein spezielles Verfahren anbietet. Noch eine Einschränkung wird durch das in Hibernate enthaltene "hbm2ddl"-Werkzeug vorgegeben, das intern verwendet wird um aus Hibernate-Mappings Datenbank-Tabellen

zu generieren. hbm2ddl ist nicht in der Lage, vorhandene Tabellen zu erweitern. Wenn also in einer vorigen Simulation eine Klasse mit einer bestimmten Zahl an Attributen gespeichert wurde und später in einer neuen Simulation die gleiche Klasse weitere Einträge bekommen hat und wieder in der selben Datenbank gespeichert werden soll, so wird dies nicht gelingen.

6 Erweiterungsmöglichkeiten und Grenzen der Architektur

DistSim ist momentan in der Lage Simulationen "als Ganzes" auf verschiedene Rechner zu verteilen. Andere Ansätze befassen sich damit, einzelne sehr komplexe Simulationen verteilt über mehrere Rechner auszuführen und so deren Laufzeit zu verringern. Ein Beispiel für solch einen Ansatz wäre PDNS (`http://www.cc.gatech.edu/computing/compass/pdns/`). Es wäre mit DistSim nun möglich, die verteilten Komponenten einer Simulation zur Laufzeit als eigene Simulationen zu definieren und so parallel auszuführen. Dazu könnte eine Start-Komponente, die als initiale "Eltern"-Simulation eingetragen wird, die Konfigurations-Bibliothek verwenden.

Ähnlich könnte auch ein automatisches Unit-Testing mittels Simulationen aussehen. Ein Prozess könnte regelmäßig Updates einer Code-Basis untersuchen und mit diesen eine festgelegte Menge von Simulationen definieren. Die Analyse der Resultate würde dann das Ergebnis des Tests liefern.

In eine andere Richtung ginge die Idee eines komplexeren Werkzeuges zur Auswertung von Resultaten. Momentan ist die Auswertungsfunktionalität des Überwachungs-Werkzeugs mehr als Vorschau auf bereits angefallene Resultate realisiert. In der tabellarischen Auflistung lassen sich nur eingeschränkt Zusammenhänge erkennen und es fehlen Auswahl- und Sortierfunktionen. Ein Werkzeug wie das mit DUCKS vorgestellte SEDU könnte hier durch grafische Aufarbeitung der Resultate mehr Überblick ermöglichen. Zudem wäre es unter Umständen sinnvoll, ein Verfahren zu definieren, mit dem Protokolle von Abläufe in Netzwerken, wie der Empfang und das Versenden von Informationen an bestimmten Knoten, standardisiert in die Datenbank eingetragen und ausgelesen werden. Dies würde eine nachträgliche Visualisierung der Abläufe mit verschiedenen vorhandenen Werzeugen, wie beispielsweise dem "Network Animator" (`http://www.isi.edu/nsnam/nam/`), ermöglichen.

Das größte Problem der gewählten Architektur liegt wohl darin, dass für deren vollständige Umsetzung eine Anpassung des Simulators unumgänglich ist. So gibt es zwar

die Möglichkeit, Resultatdateien nach der Simulation in die Datenbank zu überführen, jedoch werden auf diese Art keine strukturierten Daten angelegt. Der Simulator muss also, um die Designprinzipien umzusetzen verändert werden. Er muss die OR-Abbildungs-Bibliothek, oder zumindest ein ähnliches Schema verwenden, um Resultate in die Datenbank einzutragen.

Literatur

Barr 2004

Barr, Rimon: *An efficient, unifying approach to simulation using virtual machines*, Cornell University, Dissertation, May 2004

Barr u. a. 2004

Barr, Rimon ; Haas, Zygmunt J. ; Renesse, Robbert van: JiST: Embedding Simulation Time into a Virtual Machine. In: *Proceedings of EuroSim Congress on Modelling and Simulation* (2004)

Barr u. a. 2005a

Barr, Rimon ; Haas, Zygmunt J. ; Renesse, Robbert van: *Scalable Wireless Ad hoc Network Simulation. Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks.* Kap. Ch. 19, S. 297–311, CRC Press, 2005

Barr u. a. 2005b

Barr, Rimon ; Haas, Zygmunt J. ; Renesse1, Robbert van: JiST: An efficient approach to simulation using virtual machines. In: *Software Practice & Experience* 35(6) (2005), S. 539–576

Breslau u. a. 2000

Breslau, Lee ; Estrin, Deborah ; Fall, Kevin ; Floyd, Sally ; Heidemann, John ; Helmy, Ahmed ; Huang, Polly ; McCanne, Steven ; Varadhan, Kannan ; Xu, Ya ; Yu, Haobo: Advances in Network Simulation. In: *IEEE Computer* 33(5) (2000), S. 59–67

Fall 1999

Fall, Kevin: *Network Emulation in the Vint/NS Simulator.* 1999

Gelernter 1985

Gelernter, David: Generative communication in Linda. In: *ACM Transactions on Programming Languages and Systems* 7(1) (1985)

Haas und Barr 2005

Haas, Zygmunt J. ; Barr, Rimon: Density-Independent, Scalable Search in Ad hoc Networks. In: *Proceedings of IEEE International Symposium on Personal Indoor and Mobile Radio Communications* (2005)

Heidemann u. a. 2000

Heidemann, John ; Bulusu, Nirupama ; Elson, Jeremy ; Intanagonwiwat, Chalermek ; Lan, Kun chan ; Xu, Ya ; Ye, Wei ; Estrin, Deborah ; Govindan, Ramesh: *Effects of Detail in Wireless Network Simulation*. 2000

Huang u. a. 1998

Huang, P. ; Estrin, D. ; Heidemann, J.: *Enabling Large-scale Simulations: Selective Abstraction Approach to The Study of Multicast Protocols*. 1998

Kargl 2006

Kargl, Frank: *VANET Simulations with JiST/ SWANS*. 2006. – Folien zur Präsentation, gehalten beim Secure Vehicle Communication Workshop in Lausanne

Riley u. a. 2000

Riley, George F. ; Ammar, Mostafa H. ; Fujimoto, Richard: Stateless Routing in Network Simulations. In: *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (2000)

Erklärung

Hiermit erkläre ich, dass ich diese Studienarbeit selbstständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel, benutzt habe.

20. März 2007, Ulf Hermann