Interoperable Werkzeuge zur Laufzeituntersuchung und -steuerung paralleler und verteilter Systeme*

Roland Wismüller
Technische Universität München, Informatik
Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM)
Arcisstr. 21, D-80333 München
email: wismuell@informatik.tu-muenchen.de

4. Januar 2000

1 Einleitung

Zur Programmierung und Nutzung paralleler und verteilter Systeme sind Werkzeuge zur Beobachtung und ggf. Manipulation des Ablaufverhaltens von Programmen unentbehrlich. Aus verschiedenen technischen und wirtschaftlichen Gründen ist es oftmals wünschenswert oder sogar notwendig, mehrere solcher Werkzeuge *gleichzeitig* auf dasselbe Programm anzuwenden. Mit den derzeitig verfügbaren Werkzeugen ist dies unmöglich, da

- sie konfligierende Mechanismen zur Datenerfassung und Manipulation benutzen und
- keine gegenseitige Koordination der Werkzeuge realisierbar ist.

Das erste Problem kann durch die Definition einer universellen Monitorschnittstelle gelöst werden. Eine solche Schnittstelle wurde im Rahmen des Sonderforschungsbereichs 342 am LRR-TUM mit der *On-line Monitoring Interface Specification* (OMIS) entwickelt und mit dem *OMIS Compliant Monitoring system* OCM auch implementiert. Der zweite Aspekt wird im SFB 342 und auch in der Habilitationsarbeit des Autors für konkrete Szenarien interoperabler Werkzeuge untersucht. Eine allgemeine Untersuchung von Konzepten der Werkzeug-Koordination war im Rahmen dieser Forschungsprojekte jedoch nicht möglich, was zur Beantragung des Forschungsstipendiums führte, über dessen Ergebnisse im folgenden berichtet wird.

Das Thema des Forschungsstipendiums war die Nutzung von Monitoringtechniken zur Unterstützung von Interoperabilität, hier der Koordination von Laufzeitwerkzeugen. Konkret sollte untersucht werden, inwieweit das Monitorsystem OCM in Kombination mit der an der Univ. of Wisconsin, Madison entwickelten dynamischen Instrumentierungstechnik *dyninst* die Interaktionen zwischen den Werkzeugen unterstützen kann, die zur Verhaltensintegration erforderlich sind. Nach Sullivan [18] läßt sich diese i.w. darauf zurückführen, daß bei Eintreten eines Ereignisses in einem Werkzeug Aktionen in anderen Werkzeugen ausgeführt werden.

Zur Untersuchung dieser Fragestellung hielt sich der Autor in der Zeit vom 1.10.1999 bis 30.11.1999 an der Univ. of Wisconsin, Madison auf. Die Ziele dieses Aufenthalts waren:

1. Erstellung einer Anforderungsdefinition: welche Klassen von Aktivitäten müssen durch das Monitorsystem überwachbar sein?
2. Konzeption von Erweiterungen von OMIS/OCM, um Werkzeuge und Monitorsystem selbst durch das Monitorsystem überwachen zu können.
3. Prototypische Realisierung des Systems.

*Abschlußbericht für das Forschungsstipendium WI 1346/1-1, gefördert durch die Deutsche Forschungsgemeinschaft.

2 Grundlagen und Anforderungen

Zu Beginn des Forschungsaufenthalts wurde zunächst eine Literaturstudie zu den Themen Werkzeugintegration, Koordinationsmodelle und Koordinationssprachen durchgeführt.

Bereits im Sommer 1999 wurde ein Szenario interoperabler Werkzeuge definiert, das die effiziente Fehlersuche in Programmen mit hoher Ausführungszeit unterstützt [25]. In diesem Szenario arbeiten ein paralleler Debugger, ein System zur Erstellung von Sicherungspunkten, ein Ablauf-Visualisierer sowie ein Werkzeug zur deterministischen Ablaufsteuerung zusammen. Durch die Kombination dieser Werkzeuge kann erreicht werden, daß das untersuchten Programm in relativ kurzer Zeit auf jeden vom Visualisierer angezeigten Globalzustand zurückgesetzt werden kann, so daß dieser Zustand mit Hilfe des Debuggers detailliert zu untersuchen ist. Aus der Analyse der auftretenden gegenseitigen Wechselwirkungen der Werkzeuge wurden nun vier Anforderungen abgeleitet, die von einer Infrastruktur zur Unterstützung interoperabler Werkzeuge erfüllt werden müssen:

1. Um die Datenintegration der Werkzeuge zu ermöglichen, müssen sich diese auf eine gemeinsame, offene Schnittstelle zum on-line Monitoring stützen.

2. Zur Verhaltensintegration ist ein Mechanismus notwendig, der es Werkzeugen erlaubt, Ereignisse in anderen Werkzeugen zu überwachen, Aktionen in anderen Werkzeugen auszuführen, sowie Relationen zwischen Ereignissen und Aktionen zu definieren.

3. Zusätzlich müssen Modifikationen von überwachten Objekten als Ereignisse überwachbar sein, um eine konsistente Systemsicht der Werkzeuge zu gewährleisten.

4. Schreib- und Lesezugriffe auf überwachte Objekte müssen von Werkzeugen abgefangen werden können, um Modifikationen des Zielsystems transparent gestalten zu können.

3 Konzepte zur Umsetzung der Anforderungen

Während des Forschungsaufenthalts wurden Konzepte erarbeitet, wie diese Anforderungen mit Hilfe des Monitorsystems OCM sowie der Technik der dynamischen Instrumentierung umgesetzt werden können.

3.1 Gemeinsame Monitorschnittstelle

Die benötigte gemeinsame Monitorschnittstelle ist bereits durch OMIS und OCM gegeben, die als Grundlage des Projekts dienten. In der Praxis stellte sich jedoch bald heraus, daß die bisherige, im OCM realisierte Methodik der Monitorerweiterungen unzureichend war, die es Werkzeugen ermöglicht, zusätzlich benötigte Monitorfunktionen in den OCM zu integrieren. Dies geschah bisher statisch zur Übersetzungszeit des Monitorsystems, was zum einen die Installation neuer Werkzeuge erschwert, zum anderen auch dazu führt, daß der OCM zu einer monolithischen Komponente wird, was im Widerspruch zu den Zielen dieser Arbeit (modulare Werkzeug-Umgebung) steht.

Durch den Einsatz von Techniken des dynamischen Bindens [10], wie sie auch ein *dyninst* verwendet werden, konnte dieser Mangel beseitigt werden. Die von den Werkzeugen benötigten Monitorerweiterungen werden nun modular in Form von dynamischen Bibliotheken definiert, die vom OCM bei Bedarf zur Laufzeit geladen werden. Das Monitorsystem benötigt dazu keinerlei a priori Wissen über die Werkzeuge oder die Erweiterungen. Sämtliche benötigten Informationen über die von einer Erweiterung definierten Dienste sind in der nachgeladenen Bibliothek enthalten. Sie werden bei der Erstellung der Bibliothek durch einen speziellen Übersetzer aus einer Schnittstellenbeschreibungsdatei erzeugt.

3.2 Verhaltensintegration

Eine wesentliche Eigenschaft interoperabler Werkzeuge ist, daß ihr individuelles Verhalten untereinander koordiniert ist [19]. Als Grundlage dieser Verhaltensintegration wurde das Mediator-Modell von Sullivan [18] bzw. dessen Erweiterung durch Sattler [15] (Interaktoren) verwendet, da einfachere Modelle erhebliche Nachteile aufweisen, wie in [18] dargelegt wurde. In diesen Modellen ist eine Software-Komponente durch drei Schnittstellen definiert, jeweils eine für:

- die nach außen sichtbaren Methoden (Aktionen) der Komponente,

- die von außen sichtbaren Ereignisse innerhalb der Komponente,
- die Methoden (Aktionen), die die Komponente von ihrer Umgebung benötigt.

Das Wesentliche der genannten Modelle ist nun, daß die Verbindungen zwischen diesen Schnittstellen verschiedener Komponenten ausschließlich indirekt über Mediator- bzw. Interaktor-Objekte erfolgt, die das integrierte Verhalten dieser Komponenten beschreiben. Die Mediatoren realisieren dabei die Verbindung von auftretenden Ereignissen mit den auszuführenden Aktionen, Interaktoren binden zusätzlich Methodenaufrufe einer Komponente an Methoden anderer Komponenten.

Im Rahmen der vorliegenden Arbeit ergaben sich mit diesem Ansatz im wesentlichen drei Fragestellungen:

1. Wie kann eine Interoperabilität des Mechanismus zur Verhaltensintegration mit dem Monitorsystem erreicht werden?

   Ein technisches Problem bei der Umsetzung dieses Modells ist, daß die Werkzeuge dabei gleichzeitig mehrere Interaktionmechanismen verwenden müssen, die asynchron und ereignisgesteuert arbeiten. Zum einen ist dies die Kommunikation mit dem OCM, zum anderen die Kommunikation über den Mechanismus zur Verhaltensintegration. In der Praxis führt ein derartiges Schema zu Schwierigkeiten, da z.B. nicht blockierend auf eine Nachricht an beiden Quellen gewartet werden kann, oder – bei unterbrechungsgetriebener Kommunikation – sich die Kommunikationsmechanismen mit unvorhersehbarem Ergebnis gegenseitig unterbrechen.

   Hier tritt also auf niedrigerer Ebene wieder das Problem der Interoperabilität auf. Die einfachste Methode um diese Rekursion aufzulösen ist die Verwendung nur eines einzigen Interaktionmechanismus, d.h. die Verhaltensintegration wird über das Monitorsystem realisiert. Wie der nachfolgende Punkt zeigt, hat dies auch weitere Vorteile.

2. Wie kann erreicht werden, daß die Anwendung des Mediator-Modells mit möglichst geringen Änderungen an vorhandenen Werkzeugen möglich wird?

   Die existierenden Realisierungen des Modells (siehe [18, 17, 14]) erfordern, daß die Werkzeuge vorgegebene Schnittstellenprotokolle verwenden und über diese die exportierten bzw. importierten Aktionen und Ereignisse definieren. Neben dem Aufwand zur Anpassung von bereits existierenden Werkzeugen ist dies auch aus anderen Gründen problematisch: Zum einen wird das Werkzeug von den Schnittstellenprotokollen und damit von der Infrastruktur, z.B. CORBA [14, 12, 16], abhängig, die diese implementiert. Insbesondere können zwei Werkzeuge, die auf verschiedenen Infrastrukturen basieren, nicht ohne Änderungen interoperieren. Zum anderen erfordert dieses Vorgehen, daß der Entwickler des Werkzeugs bereits sämtliche möglicherweise von anderen benötigten Aktionen und Ereignisse voraussieht, was extrem schwierig ist und im Regelfall zu unnötig großen und komplexen Schnittstellen führt.

   Andererseits erlauben dynamische Monitortechniken wie *dyninst* [8, 9, 20] das Überwachen von bestimmten Ereignissen (bei *dyninst* derzeit den Aufruf von und die Rückkehr aus Funktionen), ohne daß dazu im überwachten Prozeß Vorkehrungen getroffen werden müßten. Zusätzlich erlaubt *dyninst* auch das dynamische Einbinden von neuem Code, d.h. unter anderem auch von neuen Schnittstellen, in überwachte Prozesse.

   Wenn also die relevanten Ereignisse einer Komponente durch den Aufruf bzw. die Rückkehr von Funktionen gegeben sind und Schnittstellen für Aktionen durch reines Hinzufügen von Code geschaffen werden können, so kann unter Verwendung dieser Monitortechnik ein beliebiges Werkzeug so in die Infrastruktur eingepaßt werden, daß eine Verhaltensintegration mit anderen Werkzeugen möglich wird, ohne daß dazu der Quellcode dieses Werkzeugs verändert werden müßte. In gut strukturierten Werkzeugen kann davon ausgegangen werden, daß die genannten Voraussetzungen erfüllt sind. Selbst wenn diese erst durch Modifikation des Codes geschaffen werden müssen, hat die Methode den entscheidenden Vorteil, daß diese Änderungen unabhängig von der zugrundeliegenden Infrastruktur sind.

3. Wie können Dynamik und Erweiterbarkeit unterstützt werden?

   Sullivan stellt in [18] die Bedeutung der einfachen Erweiterbarkeit und Anpaßbarkeit von Systemen heraus. In dem hier zugrundegelegten Anwendungsszenario können durch den Anwender prinzipiell jederzeit neue Laufzeitwerkzeuge gestartet werden, die mit den bereits ablaufenden interoperieren müssen. Dies bedeutet, daß die zur Verhaltensintegration notwendigen Ereignis-/Aktions-Beziehungen dynamisch etabliert werden müssen. Wie bereits angesprochen, können hinzugekommene Werkzeuge evtl. erweiterte Schnittstellen vorhandener Werkzeuge benötigen, die dann ebenfalls dynamisch bereitgestellt werden müssen.

Durch die Realisierung des Mediator-Modells über das Monitorsystem können auch diese Anforderungen erfüllt werden.

Konzeptuell läßt sich eine Verhaltensintegration auf Basis des Mediator-Modells also wie folgt durch das Monitorsystem unterstützen [24]:

- Die Werkzeuge werden mit in das überwachte System aufgenommen. Für das Monitorsystem sind Werkzeuge dabei eine Unterklasse von Prozessen, d.h. alle Prozeßdienste sind auch auf Werkzeuge anwendbar.
- Dies betrifft insbesondere die Möglichkeit zur Überwachung von Ereignissen, etwa des Aufrufs einer spezifizierten Funktion.
- Werkzeuge stellen Aktionen als OCM-Dienste bereit. Der zur Schnittstellenanpassung erforderliche Code wird dabei automatisch durch dynamisches Binden zur Laufzeit zum Werkzeug hinzugefügt.
- Die Verknüpfung von Ereignissen mit Aktionen läßt sich dann in natürlicher Weise auf die von OMIS definierte Ereignis-/Aktionssprache [11] abbilden.

3.3 Überwachung von Objekt-Modifikationen

Um eine konsistente Systemsicht aller Werkzeuge zu ermöglichen, müßte jede vom OCM aufgrund einer Anfrage eines Werkzeug vorgenommene Zustands-Modifikation eines überwachten Objekts für die anderen Werkzeuge über ein Ereignis sichtbar sein. Die explizite Implementierung jedes einzelnen Ereignisses im OCM verbietet sich jedoch aufgrund der durch die hohe Anzahl verschiedener Ereignisse verursachten Komplexität und der dadurch entstehenden Unübersichtlichkeit des Monitor-Codes.

Bei der Beantragung des Projekts wurde daher vorgeschlagen, die Überwachung dieser Ereignisse wie in Abschnitt 3.2 ebenfalls durch dynamische Instrumentierung vorzunehmen. Leider zeigte sich, daß dieser Ansatz aus zwei Gründen nicht durchführbar ist:

1. Ein technischer Grund ist die Tatsache, daß *dyninst* nicht zur Selbstmodifikation bzw. -überwachung verwendet werden kann. Dies liegt z.T. darin begründet, daß die zugrundeliegenden Betriebssystemdienste dies verbieten. Eine dynamische Instrumentierung interessierender Objektmodifikationen innerhalb des OCM ist damit ohne erhebliche Änderungen in *dyninst* nicht möglich.
2. Maßgeblicher ist jedoch ein konzeptueller Grund. Die skizzierte Vorgehensweise würde Ereignisse der Form „Prozeß x des OCM hat Zustandskomponente $y.z$ von Objekt y modifiziert“ liefern. Benötigt werden aber Ereignisse der Form „Zustandskomponente $y.z$ von Objekt y wurde modifiziert“. D.h. das Ereignis muß zum Objekt y gehören, nicht zu einem Prozeß des OCM. Demnach muß auch die Überwachung bei diesem Objekt bzw. seinem Stellvertreterobjekt innerhalb des OCM ansetzen, statt bei den Prozessen des OCM, wie dies bei Verwendung von *dyninst* der Fall wäre.

Diese Überlegungen führten zur Untersuchung eines detaillierteren Objektmodells für OMIS und OCM. Ein Objekt (z.B. ein Prozeß der überwachten Anwendung) besitzt dabei beliebig viele Zustandskomponenten, sowie zwei generische, im Idealfall implizit aufgerufene Methoden, um diese Komponenten zu lesen und zu schreiben. Diese beiden Methoden führen einen Mustervergleich des Komponentennamens mit zur Laufzeit registrierten Mustern durch und lösen ggf. ein Ereignis aus.

Wäre für den OCM ein statisches Objektmodell ausreichend, könnte dieses einfach auf Java- oder C++-Objekte abgebildet werden. Die Erweiterbarkeit des OCM hat jedoch zur Konsequenz, daß auch Objekte erweiterbar sein müssen. So kann eine zur Laufzeit nachgeladene OCM-Erweiterung neue Teile des Prozeßzustandes, z.B. Deskriptoren offener Dateien, überwachen, was zur dynamischen Erweiterung der Stellvertreterobjekte aller Prozesse führen muß.

Im Rahmen des Forschungsaufenthalts wurde ein Implementierungskonzept für ein derartiges, dynamisches Objektmodell entwickelt. Berücksichtigt wurde dabei insbesondere die Vermeidung von Namenskonflikten, falls verschiedene OCM-Erweiterungen Komponenten zur selben Objektklasse hinzufügen, sowie ein möglichst effizienter Objektzugriff. Wegen des hohen zeitlichen Aufwands der erforderlichen syntaktischen Änderung aller Objektzugriffe innerhalb des OCM-Codes wurde dieses Modell jedoch noch nicht in den OCM integriert.

Ein noch nicht abschließend geklärtes Problem ist das des Aliasing: zwei OCM-Erweiterungen könnten z.B. einem Prozeß-Stellvertreterobjekt y verschieden benannte Komponenten hinzufügen (z.B. u bzw. v), die jedoch dieselbe Zustandskomponente des realen Prozesses repräsentieren. Ein Werkzeug, das Änderungen von $y.u$ überwacht würde dabei eine Zustandsänderung übersehen, die durch Zugriff auf $y.v$ verursacht wird. Aliasing muß also entweder vermieden oder explizit durch die Erweiterungen bekanntgemacht und beim Mustervergleich berücksichtigt werden.

3.4 Abfangen von Zugriffen

Bei der gleichzeitigen Anwendung mehrerer Laufzeitwerkzeuge treten Situationen auf, in denen ein Werkzeug Modifikationen am Zielsystem vornimmt, die von anderen Werkzeugen jedoch nicht gesehen werden sollen. In diesen Situationen ist es notwendig, die entsprechenden Zugriffe anderer Werkzeuge abzufangen und ggf. deren Ergebnis zu modifizieren.

Abstrakt bedeutet dies, daß in dem in Abschnitt 3.2 definierten Mediator-Modell Ereignisse nicht nur asynchron, sondern auch synchron behandelt werden müssen, d.h. die Komponente, das das Ereignis auslöst wird temporär blockiert, bis alle Aktionen ausgeführt sind. Die Aktionen können dabei insbesondere den Zustand der blockierten Komponente verändern. Ein derartiges Konzept ist z.B. bei dem modularen Texteditor *emacs* [6] in Form sog. *hooks* realisiert, aber bisher in der Literatur zum Thema Kompositionsmodelle (siehe [1, 3, 7, 5, 15, 2]) nicht systematisiert worden.

Das Ereignis-/Aktionsschema des OCM erlaubt ein solches Vorgehen, da standardmäßig das auslösende Objekt eines Ereignisses so lange suspendiert wird, bis alle mit diesem Ereignis verknüpften Aktionen ausgeführt sind [11]. Aufgrund von Leistungsüberlegungen sollte das Abfangen von Zugriffen jedoch auch mit einem leichtgewichtigeren Mechanismus, etwa über Rückruffunktionen erfolgen können. Ein derartiger Mechanismus arbeitet zwar nicht verteilt, in der Regel sind die auszuführenden Aktionen jedoch ohnehin lokal. Mit dem in Abschnitt 3.3 skizzierten Objektmodell ist dies problemlos zu realisieren, da die Lese- und Schreibmethoden bei einem erfolgreichen Mustervergleich direkt die entsprechenden Rückruffunktionen aktivieren können.

4 Abstrakte Struktur des Monitorsystems

Schließt man die skizzierten Konzepte in ein Monitorsystem wie den OCM ein, so entsteht eine Struktur, die einem Komponentenmodell wie CORBA ähnlich ist, jedoch auch entscheidende Unterschiede aufweist. Da kein prinzipieller Unterschied mehr zwischen Werkzeugen und überwachten Objekten besteht, kann das Gesamtsystem aus Werkzeugen, Monitorsystem und Zielsystem abstrakt als eine Menge verteilter, interagierender Objekte (i.w. Anwendungsprozesse und Werkzeuge) betrachtet werden, wie in Abb. 1 dargestellt ist. Jedes Objekt bietet dabei Dienste an, die von anderen Objekten benutzt werden können. Die dazu nötige Kommunikation wird durch eine verteilte Infrastruktur realisiert, die Teil des Monitorsystems ist.

Im Gegensatz zu üblichen verteilten objektorientierten Systemen werden jedoch hier die Dienste nicht notwendigerweise von dem Objekt selbst realisiert. Für das Objekt „Prozeß“ existiert beispielsweise ein Dienst „anhalten“, der nicht vom jeweiligen Prozeß, sondern vom Monitorsystem realisiert wird. Einige Dienste müssen auch zentral bzw. in verteilter Weise realisiert werden. Das Vermitteln (*routing*) von Dienst-Anforderungen ist beim OCM daher erheblich vielseitiger.

Ein weiterer wichtiger Unterschied ist die explizite Unterstützung von Ereignissen, die weit über einen Ereignisdienst hinausgeht, wie er z.B. von CORBA realisiert wird. Ereignisse gehören wie andere Dienste mit zur Schnittstelle, die ein Objekt anbietet. Die Verknüpfung zwischen Ereignissen und auszuführenden Aktionen ist nicht fest, sondern beliebig und zur Laufzeit frei programmierbar. Dies entspricht modernen Komponentenmodellen, wie sie in [18, 15] entwickelt wurden.

Das Monitorsystem kann also als die Realisierung eines solchen Komponentenmodells betrachtet werden, das zusätzlich einige spezielle Dienste auf den beteiligten Objekten anbietet. Das System ist dabei vollständig dynamisch und erlaubt nicht nur das Hinzufügen neuer Objekte und Objekttypen zur Laufzeit, sondern auch die Erweiterung der Objektschnittstellen, d.h. das Hinzufügen neuer Dienste zu existierenden Objekten.

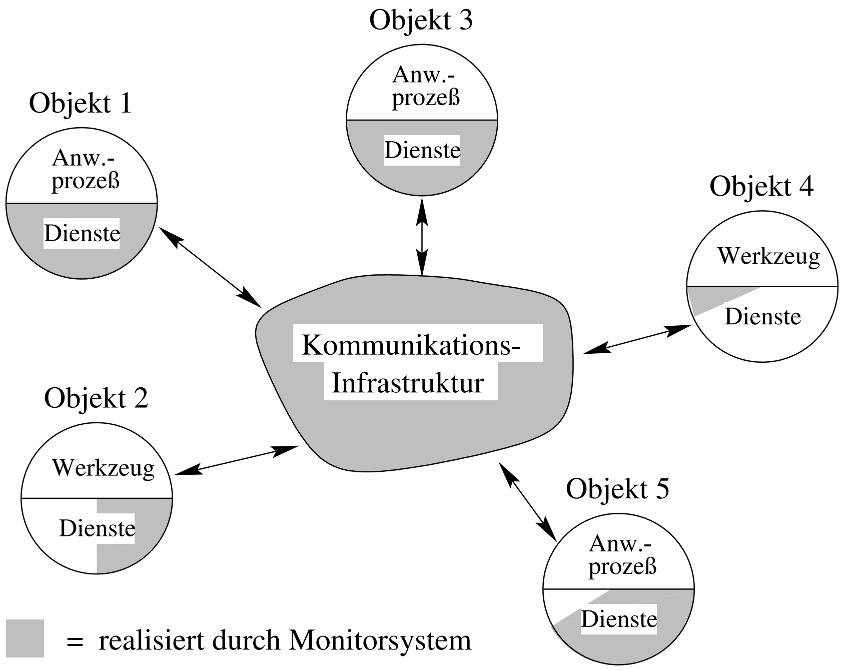

Abbildung 1: Abstrakte Systemstruktur

5 Realisierung

Entsprechend der obenstehenden Überlegungen wurde zunächst die Implementierung des OCM klarer strukturiert: Die Basis des Monitorsystems bildet die Kommunikations-Infrastruktur, die die Ereignis/Aktionsverwaltung sowie die Vermittlung einschließt, aber prinzipiell keinerlei Monitoring-Dienste beinhaltet. Diese sind nun ausschließlich in Form von OCM-Erweiterungen realisiert.

Um die in Abschnitt 3.2 skizzierte Methode zu realisieren wurde zunächst das OMIS Objektmodell um die Klasse „Werkzeug“ erweitert, die von der Klasse „Prozeß“ abgeleitet ist. Die beiden Klassen unterscheiden sich i.w. nur in der Art und Weise, wie der OCM die Kontrolle über den entsprechenden Prozeß gewinnt. Nach einer intensiveren Einarbeitung in die an der Univ. of Wisconsin, Madison entwickelten *dyninst*-Programmierbibliothek wurde anschließend eine OCM-Erweiterung realisiert, die auf der Basis von *dyninst* die folgenden Funktionen für beliebige überwachte Prozesse (d.h. insbesondere auch für Werkzeuge) realisiert:

- Erkennung eines Ereignisses bei Start bzw. Rückkehr einer mit Namen gegebenen Funktion,
- dynamisches Dazubinden von beliebigen Code-Bibliotheken zu dem Prozeß,
- Aufruf von Funktionen im Kontext des überwachten Prozesses durch *inferior RPC*[1].

Dabei ist keinerlei Vorbereitung bzw. Modifikation des Programmcodes dieser Prozesse erforderlich.

Die ersten beiden Funktionen können direkt zur Realisierung der Verhaltensintegration verwendet werden. Ein Aufruf von Aktionen eines Werkzeugs durch *inferior RPC* stellte sich jedoch als unpraktikabel heraus, da auf diese Weise keine Kontrolle über den exakten Aufrufzeitpunkt im Ablauf des Werkzeugs möglich ist. Erfolgt der Aufruf (zufällig) innerhalb einer nicht wiedereintrittsfähigen Funktion, kann ein Fehlverhalten des Werkzeugs resultieren. Der Aufruf von Aktionen in Werkzeugen wird daher durch ein Trojanisches Pferd realisiert: Das OMIS-Protokoll verlangt von Werkzeugen den Aufruf einer Nachrichtenbehandlungsfunktion der OMIS-Bibliothek, wenn eine Nachricht vom Monitorsystem eintrifft [11]. Diese Funktion wurde so erweitert, daß sie zusätzlich abprüft, ob Aktionen auszuführen sind und sie ggf. ausführt. Der dazu notwendige Code kann mittels *dyninst* wieder zur Laufzeit in das Werkzeug eingebunden werden.

[1] Mit diesem Term wird in *dyninst* der Aufruf einer Funktion innerhalb eines anderen Prozesses bezeichnet. Im Gegensatz zum üblichen RPC (*remote procedure call*) muß dieser jedoch auf dem lokalen Rechenknoten ablaufen.

Bei der Realisierung der Erweiterung zeige sich, daß durch die gleichzeitige Prozeß-Überwachung durch *dyninst* und die im OCM bereits vorhandenen Mechanismen nun Interoperabilitätsprobleme innerhalb des OCM auftraten, was die Forderung nach einer einheitlichen Monitortechnik unterstreicht. Leider unterstützt die derzeitige Version von *dyninst* noch nicht alle Funktionen, die notwendig wären, um den OCM vollständig auf diese Schnittstelle zu stützen. An entsprechenden Erweiterungen wird an der Univ. of Maryland, College Park gearbeitet. Durch die Zusammenarbeit mit den Entwicklern in Madison konnten schließlich aber alle auftretenden Probleme gelöst oder umgangen werden. Ein Erfahrungsbericht mit Vorschlägen zur Verbesserung der Interoperabilität von *dyninst* mit existierenden Monitorsystemen wurde an die Entwickler weitergegeben [22].

Um eine zu starke Abhängigkeit des OMIS/OCM-Projekts vom *dyninst*-Projekt zu vermeiden, wurde weiterhin untersucht, inwieweit die benutzten Funktionen von *dyninst* im OCM mit akzeptablem Aufwand nachgebildet werden können. Es zeigte sich, daß dies nur dann möglich ist, wenn man eine statische Vorbereitung der zu koordinierenden Werkzeuge in Kauf nimmt. Im wesentlichen müssen diese dazu neu gebunden werden, da zum einen zunächst ein Stück Code statisch in das Werkzeug eingebunden werden muß, das dann das dynamische Nachladen von weiterem Code ermöglicht. Zum anderen muß zur Erkennung von Funktionsaufrufen bzw. -rückkehr eine Instrumentierung des Werkzeugcodes vorgenommen werden. In der konkreten Implementierung erfolgt diese durch die Methode des *binary wrapping* [4], die zumindest die Notwendigkeit eine Neuübersetzung des Werkzeugs vermeidet.

Mit der so entstandenen Infrastruktur wurde schließlich eine Umgebung interoperabler Werkzeuge realisiert, die aus dem Debugger DETOP [13] und dem Visualisierungswerkzeug VISTOP [21] besteht. Ein zusätzliches „Werkzeug“ realisiert das Mediator-Objekt, das die Aufgabe hat, die Werkzeugumgebung koordiniert zu starten, sowie über das Monitorsystem die benötigten Ereignis-/Aktionsbeziehungen zu programmieren (siehe Abb. 2). Die entstandene Umgebung wurde in einer abschließenden Vorführung für Prof. Barton Miller und die Mitglieder seiner Forschungsgruppe demonstriert.

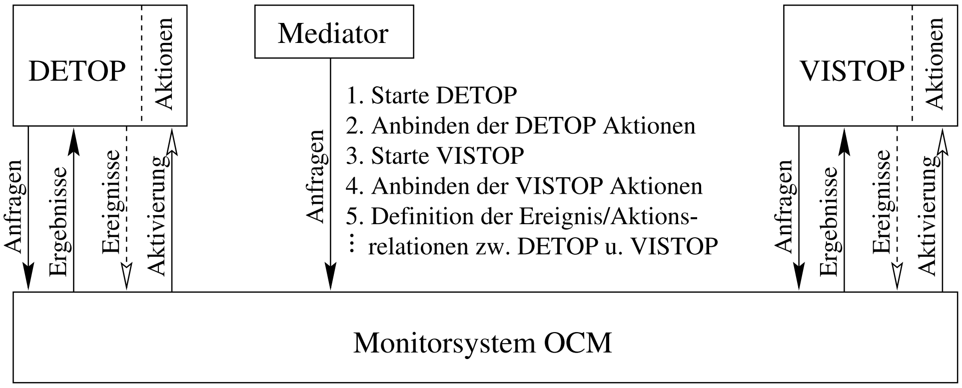

Abbildung 2: Struktur der Demonstrations-Werkzeugumgebung

6 Zusammenfassung

Im Rahmen des geförderten Forschungsaufenthalts wurden Techniken zur Unterstützung der Interoperabilität von Laufzeitwerkzeugen durch das zugrundeliegende Monitorsystem entwickelt und implementiert. Sie basieren auf dem Mediator-Modell nach Sullivan, das durch eine synchrone Ereignisbehandlung erweitert wurde. Dies führt zu einer neuen Sichtweise des Monitorsystems, das in erster Linie ein Komponentenmodell realisiert und sozusagen lediglich als Erweiterung noch Monitoring-Dienste auf den beteiligten Objekten bietet. Die entwickelten Konzepte wurden in das Monitorsystem OCM integriert und basierend auf diesem System eine Demonstrations-Werkzeugumgebung erstellt und der gastgebenden Forschungsgruppe präsentiert.

Die berichteten Arbeiten werden außerhalb des Förderprojekts fortgeführt. Das Ziel ist, die komplette, in Abschnitt 2 skizzierte Werkzeugumgebung, sowie die dazu noch notwendigen Erweiterungen des Monitorsystems zu realisieren. Dazu müssen weitere Interoperabilitätsaspekte betrachtet werden, die speziell das Monitoring betreffen. Als Beispiele seien hier die Bestimmung der Reihenfolge von Instrumentierungen sowie der Reihenfolge von Ereignisbehandlungen desselben Ereignisses genannt. Die Ergebnisse der gesamten Arbeit werden bis Ende 2000 in der Habilitationsschrift des Autors [23] veröffentlicht.

Literatur

[1] F. Arbab, P. Ciancarini, und C. Hankin. Coordination Languages for Parallel Programming. *Parallel Computing*, 24(7):989–1004, 1998.

[2] J. A. Bergstra und P. Klint. The TOOLBUS Coordination Architecture. In P. Ciancarini und C. Hankin, Herausgeber, *Coordination : languages and models : First International Conference COORDINATION '96*, Nummer 1061 in Lecture Notes in Computer Science, Seiten 75–88, Cesena, Italy, April 1996. Springer Verlag.

[3] P. Ciancarini und C. Hankin, Herausgeber. *Coordination Languages and Models – First International Conference, COORDINATION '96*, Band 1061 aus *Lecture Notes in Computer Science*, Cesena, Italien, April 1996. Springer Verlag.

[4] J. Cargille und B. P. Miller. Binary Wrapping: A Technique for Instrumenting Object Code. *ACM SIGPLAN Notices*, 27(6):17–18, Juni 1992.

[5] P. Ciancarini und A.L. Wolf, Herausgeber. *Coordination Languages and Models – Third International Conference, COORDINATION '99*, Band 1594 aus *Lecture Notes in Computer Science*, Amsterdam, Niederlande, April 1999. Springer Verlag.

[6] GNU Emacs. WWW-Seite der Free Software Foundation, Boston, MA, USA. `http://www.gnu.org/software/emacs/emacs.html`.

[7] D. Garlan und D. Le Metayer, Herausgeber. *Coordination Languages and Models – Second International Conference, COORDINATION '97*, Band 1282 aus *Lecture Notes in Computer Science*, Berlin, September 1997. Springer Verlag.

[8] J. K. Hollingsworth, B. P. Miller, und J. Cargille. Dynamic Program Instrumentaiton for Scalable Performance Tools. In *Proc. Supercomputing'94*, Seiten 841–850, Washington D.C., November 1994. IEEE.
`ftp://grilled.cs.wisc.edu/technical_papers/dyninst.ps.Z`.

[9] J. Hollingsworth. An Application Program Interface for Runtime Code Patching. März 1996.
`http://www.cs.umd.edu/~hollings/papers/dyninstAPI.ps`.

[10] J. Levine. *Linkers and Loaders*. Morgan Kaufmann Publishers, 1999. ISBN 1-55860-496-0.

[11] T. Ludwig, R. Wismüller, V. Sunderam, und A. Bode. *OMIS — On-line Monitoring Interface Specification (Version 2.0)*, Band 9 aus *LRR-TUM Research Report Series*. Shaker Verlag, Aachen, Germany, 1997. ISBN 3-8265-3035-7.
`http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz`.

[12] Object Management Group Home Page. WWW-Seite der Object Management Group Inc., Framingham, MA, USA.
`http://www.omg.org/`.

[13] M. Oberhuber und R. Wismüller. DETOP - An Interactive Debugger for PowerPC Based Multicomputers. In P. Fritzson und L. Finmo, Herausgeber, *Parallel Programming and Applications*, Seiten 170–183. IOS Press, Amsterdam, Mai 1995.
`http://wwwbode.in.tum.de/~wismuell/pub/zeus95.ps.gz`.

[14] G. Paul und K.-U. Sattler. MediatorService - Integration von verteilten Objekten durch Beschreibung der Interaktionen. In *Proc. Int. Workshop Trends in Distributed Systems*, Aachen, Oktober 1996.
`http://wwwiti.cs.uni-magdeburg.de/~sattler/papers/trevs.ps`.

[15] K.-U. Sattler. *Tool-Komposition in integrierten Entwurfsumgebungen*. Doktorarbeit, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 1998. Erschienen als Fortschritt-Bericht Nr. 543 im VDI-Verlag, Düsseldorf.

[16] J. Siegel. *CORBA Fundamentals and Programming*. John Wiley & Sons, 1996. ISBN 0471-12148-7.

[17] K. J. Sullivan, I.J. Kalet, und D. Notkin. Evaluating the Mediator Method: Prism as a Case Study. *IEEE Transactions on Software Engineering*, 22(8):563–579, August 1996. auch Techn. Bericht CS-95-29 der Univ. Virginia.
`ftp://ftp.cs.virginia.edu/pub/techreports/CS-95-29.ps.Z`.

[18] K. J. Sullivan. Mediators: Easing the Design and Evolution of Integrated Systems. Techn. Bericht 94-08-01, Dept. of Computer Sciences and Engineering, Univ. of Washington, 1994.
`ftp://ftp.cs.washington.edu/tr/1994/08/UW-CSE-94-08-01.PS.Z`.

[19] J. Trinitis. *Interoperability of Distributed Checkpointing and Debugging Tools*. Doktorarbeit, Fakultät für Informatik, Technische Universität München, Dezember 1999.

[20] An Application Program Interface (API) for Runtime Code Generation. WWW-Seite der University of Maryland, College Park, USA.
`http://www.cs.umd.edu/projects/dyninstAPI/`.

[21] R. Wismüller. State Based Visualization of PVM Applications. In A. Bode, J. Dongarra, T. Ludwig, und V. Sunderam, Herausgeber, *Parallel Virtual Machine – EuroPVM'96*, Band 1156 aus *Lecture Notes in Computer Science*, Seiten 91–99, Munich, Germany, Oktober 1996. Springer Verlag.
`http://wwwbode.in.tum.de/~wismuell/pub/europvm96.ps.gz`.

[22] R. Wismüller. Experience Report on Using the dyninstAPI. Interner Bericht für die Forschergruppe um Prof. Barton Miller, Univ. of Wisconsin, Madison, USA, 1999.
`http://www.in.tum.de/~wismuell/pub/madison.ps.gz`.

[23] R. Wismüller. *Interoperable Laufzeit-Werkzeuge für parallele und verteilte Systeme*. Habilitationsschrift, Technische Universität München, 2000. In Vorbereitung.

[24] R. Wismüller und T. Ludwig. Interoperable Run-Time Tools for Distributed Systems – A Case Study. *The Journal of Supercomputing*. Im Okt. 1999 zur Veröffentlichung eingereicht.
`http://www.in.tum.de/~wismuell/pub/tjsc99.ps.gz`.

[25] R. Wismüller und T. Ludwig. Interoperable Run-Time Tools for Distributed Systems - A Case Study. In H. R. Arabnia, Herausgeber, *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Application, PDPTA'99, Volume IV*, Seiten 1763–1769, Las Vegas, NV, USA, Juni 1999. CSREA Press.