Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
Institut Datenbanken und Informationssysteme

Automatischer Client für ADEPT2

Bachelorarbeit
vorgelegt von
Matthias Rink

Oktober 2007

Gutachter: Prof. Dr. Peter Dadam

Zusammenfassung

Um sich wiederholende Arbeitsabläufe in Unternehmen mit Software–Systemen zu unterstützen werden Prozess–Managemenet–Systeme eingesetzt. Sie bieten neben der Zuweisung von Arbeitsschritten an Benutzer auch Überwachung der Ausführung dieser Arbeitsschritte und Verwaltung der verwendeten Daten an. Dabei kommen auch Arbeitsschritte vor die ohne Eingriff eines menschlichen Benutzers automatisch durchgeführt werden können. Diese Arbeitsschritte können von einem Automatischen Client ausgeführt werden. Ein Automatischer Client kann dabei den Aufbau eines normalen Clienten besitzen und bedarf nur weniger Anpassungen. Allerdings muss für einen Automatischen Client, der als Konsolenanwendung gestartet wird, eine Schnittstelle geschaffen werden über die der Status des Clients von einem anderen Rechner aus überprüft werden kann und über die der Client auch gesteuert werden kann.

Der Automatische Client besteht aus einer Arbeitsliste, über die er die Arbeitsschritte zum Ausführen bereit gestellt bekommt, eine Ausführungseinheit. Diese benutzt einen *ThreadPool* um alle Schritte in der Arbeitsliste nebenläufig und unabhängig voneinander auszuführen.

Inhaltsverzeichnis

1 Einleitung

In Organisationen werden häufig feste Arbeitsabläufe abgearbeitet die auch als Geschäftsprozesse bezeichnet werden. Dabei besteht ein Arbeitsablauf aus einer Menge von Arbeitsschritten die nacheinander durchgeführt werden. Ein solcher Arbeitsschritt kann eine beliebige Tätigkeit sein (zum Beispiel „Dokument drucken“, „Vertreter anrufen“ oder auch „15 Minuten auf Ergebnis warten“). Um die Bearbeitung solcher Arbeitsabläufe mit Software–Systemen zu unterstützen werden Prozess–Management–Systeme (PMS)[1] eingesetzt. Diese bieten eine Überwachung und Steuerung der schrittweisen Abarbeitung von Arbeitsabläufen. Dafür bieten sie jedem Benutzern des Systems alle aktuell von ihm zu bearbeitenden Schritte in einer persönlichen Arbeitsliste an. Diese werden vom Benutzer bearbeitet und das PMS sorgt dafür, dass in diesem Arbeitsschritt gewonnene Daten in darauffolgenden Arbeitsschritten zur Verfügung stehen.

1.1 Vorlagen für Geschäftsprozesse

Damit bestimmte Arbeitsschritte eines Geschäftsprozesses von einem PMS unterstützt abgearbeitet werden können, muss zuerst eine Vorlage für den Prozess modelliert und dem System bereitgestellt werden. Dabei besteht ein Ablauf aus einem gerichteten Graphen mit genau einem Start- und einem Endknoten. Zwischen diesen Knoten können beliebig viele unterschiedliche Arbeitsschritte eingefügt werden, die die eigentlichen Tätigkeiten in dem Geschäftsprozess darstellen. In Abbildung 1 ist ein solcher Ablaufgraph einer Vorlage dargestellt. In diesem Ablaufgraph kommen zwei Arten von Arbeitsschritten zum Einsatz, die manuellen und die automatischen Arbeitsschritte. Dabei ist ein manueller Arbeitsschritt einer, der von einem menschlichen Benutzer ausgeführt wird bzw. zumindest dem Eingriff eines menschlichen Benutzers bedarf. Ein automatischer Arbeitsschritt ist dann entsprechend einer, der ohne Einwirkung eines menschlichen Benutzers von einem Rechner bearbeitet werden kann. Die Ausführung des Arbeitsschrittes kann zudem an einen bestimmten Benutzer gebunden werden, indem dieser Benutzer als Bearbeiter dieses Schrittes in der Vorlage eingetragen wird. Diese Bearbeiterzuordnung ist auch in der Abbildung dargestellt.

1.2 Architektur von Prozess–Managemen–Systemen

PMS basieren meist auf einer Client/Server–Architektur [Hol05] wie in Abbildung 2 zu sehen ist. Es laufen also ein oder mehrere Server, die Vorlagen für Arbeitsabläufe anbieten und deren Ausführung überwachen. An diesem Server können sich beliebig viele Benutzer mit Clients anmelden und die angebotenen Vorlagen starten. Dabei wird einem

[1] Beispiele für PMS sind „ADEPT2“ der Universität Ulm [Abt04], „WebSphere MQ Workflow“ von IBM [IBM], die „IProcess–Suite“ von TIBCO [TIB] und die „BPM Suite“ von Ultimus [Ult].

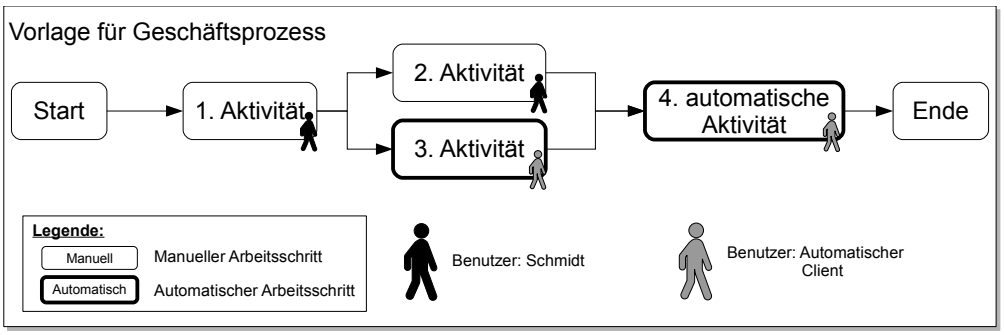

Abbildung 1: Vorlage für Geschäftsprozess

Benutzer eine Arbeitsliste zugewiesen in der der Server dem Benutzer alle gerade verfügbaren Arbeitsschritte zum bearbeiten anbietet. Der Benutzer des Clients kann nun einen oder mehrere Arbeitsschritte auswählen und abarbeiten. Dabei kann ein Schritt sehr unterschiedlich aussehen. Beispielsweise kann ein Dokument mit einer Vorlage in Open Office erstellt werden und vom Benutzer ausgefüllt und gespeichert werden oder es wird eine Email generiert die der Benutzer dann absenden soll. Ist ein Arbeitsschritt vollständig bearbeitet, wird das dem Server gemeldet und er kann in der Ausführung des gestarteten Arbeitsablaufes fortsetzen und dem Benutzer den nächsten Arbeitsschritt zum Bearbeiten anbieten.

1.3 Aufgabenstellung

Die Aufgabe die dieser Abschlussarbeit zugrunde liegt, ist die Planung und Erstellung eines Clients für das Prozess–Management–System ADEPT2, der automatische Arbeitsschritte ausführt. Dabei soll der Aufbau und die Funktionsweise einem normalen Client nachempfunden sein, so dass es keiner unterschiedlichen Behandlung der Clients durch den Server bedarf. Die Zuweisung der Arbeitsschritte an den Automatischen Client soll allein durch die Benutzerzuordnung erfolgen. Der Client soll auf der Kommandozeile laufen und Ausgaben sowohl auf dieser Konsole als auch in eine Datei erlauben. Da durch die Ausführung auf der Konsole keine ansprechende Steuerung des Clients gewährleistet ist, soll zusätzlich eine Schnittstelle entworfen werden über die der Client von einem Kontrollprogramm überwacht und gesteuert werden kann.

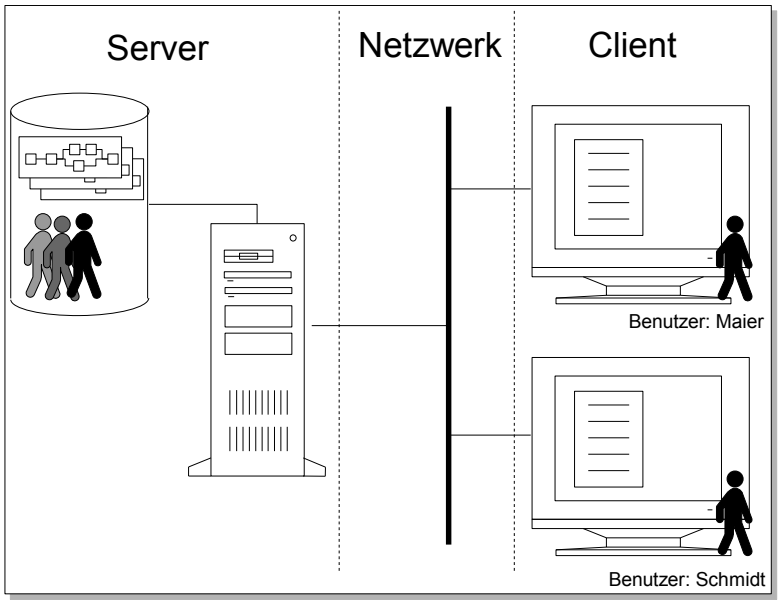

Abbildung 2: Die Client/Server Architektur eines Prozess–Management–Systems

2 Automatischer Client

In diesem Abschnitt wird der Aufbau und die Funktionsweise eines Automatischen Clients für ADEPT2 [Abt04][Abt95][Rei00] vorgestellt. ADEPT2 ist ein Workflow–Management–System, welches derzeit in der Abteilung Datenbanken und Informationssysteme an der Universität Ulm entwickelt wird. Da der Automatische Client wie ein normaler Client des ADEPT2–Systems aufgebaut sein soll, wird zuerst der generelle Aufbau vorgestellt und dabei beschrieben, welche Komponenten zum Einsatz kommen und wofür sie gebraucht werden. Darauf aufbauend sollen die Erweiterungen und Anpassungen vorgestellt werden, die für den Automatischen Client benötigt werden. Als Vorlage für den Client dient der Batch-Client [Kre02] für das ADEPT–System, welches der Vorgänger von ADEPT2 zwei ist.

2.1 Aufgabe des Automatischen Clients

Bei der Abarbeitung von Arbeitsabläufen kann man zwischen manuellen und automatischen Arbeitsschritten wie in Abschnitt 1 dargestellt unterscheiden. Die Ausführung von automatischen Schritten kann vom Server übernommen werden, was allerdings bei vielen bzw. sehr rechenintensiven Arbeitsschritten zu einer hohen Belastung des Servers führt und seine eigentliche Aufgabe, das Verteilen von Arbeit, beeinträchtigt.

Deshalb ist es sinnvoll diese automatischen Schritte wie die manuellen auf einem separaten Client auszuführen. Da dieser Client nur automatische Schritte ausführen muss aber sonst die selbe Arbeitsweise wie ein normaler Client besitzt, bietet sich die Möglichkeit an, die Zuteilung von automatischen Arbeitsschritten an diesen Client analog zu der Zuteilung an einen Benutzer zu gestalten. Folglich ist es nur nötig einen Benutzer im System anzulegen der automatische Schritte ausführen soll, diese Schritte beim Erstellen von Vorlagen dem Benutzer zuzuweisen und einen Automatischen Client zum Ausführen dieser Arbeitsschritte mit dem Benutzer am System anzumelden.

2.2 Aufbau eines Workflow–Client

Ein Client des ADEPT2–Systems besteht aus drei Komponenten, der Arbeitsliste, der Ausführungseinheit und der Laufzeitumgebung (siehe Abbildung 3). Die Anmeldung am Server und die Verwaltung von auszuführenden Arbeitsschritten übernimmt die Arbeitsliste. Sie meldet den Client mit einem Benutzername und Passwort am Server an und ist daraufhin bereit vom Server Arbeitsschritte zugewiesen zu bekommen. Aus Sicht des Benutzers eines Clients stellt sich die Arbeitsliste als Momentaufnahme von aktuell ausführbaren Arbeitsschritten dar. Über die Liste können Schritte gestartet, unterbrochen oder abgelehnt werden. Desweiteren wird beim Anmelden dem Server mitgeteilt, in welcher Position der Benutzer des Clients in der Organisation beschäftigt ist. Damit wird dem Server mitgeteilt welche Arbeitsschritte der Benutzer des Clients ausführen kann bzw. darf.

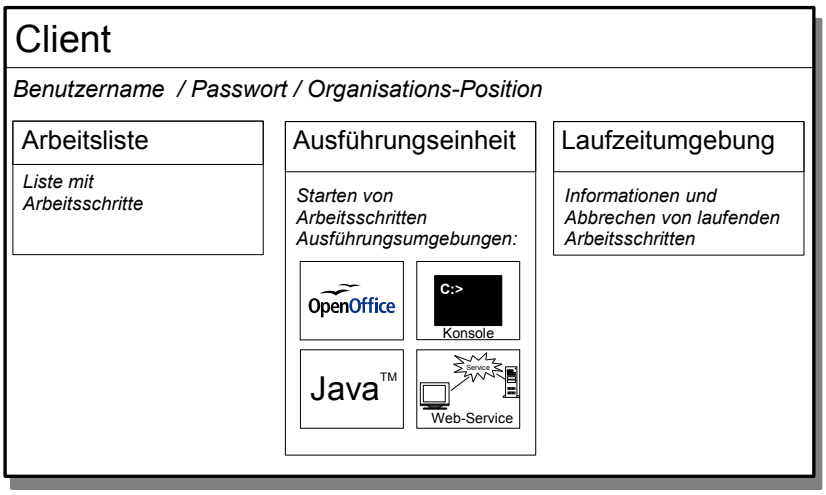

Abbildung 3: Aufbau eines Workflow–Clients

Die Ausführungseinheit kommt zum Zuge wenn der Benutzer des Clients einen Arbeitsschritt aus der Arbeitsliste ausführt. Dabei wird für einen Arbeitsschritt durch die Ausführungseinheit eine passende Ausführungsumgebung ausgewählt und der Schritt darin gestartet. Die verschiedenen Ausführungsumgebungen werden benötigt um verschiedenartige Arbeitsschritte auszuführen. So ist es zum Beispiel möglich Java-Klassen direkt oder ein externes Programm mit einer ausführbaren Datei zu starten. Auf einen gestarteten Arbeitsschritt kann durch die Laufzeitumgebung zugegriffen werden, die eng mit der Ausführungseinheit verbunden ist. Über die Laufzeitumgebung kann beispielsweise ein laufender Arbeitsschritt abgebrochen oder Informationen über dessen Zustand eingeholt werden.

2.3 Aufbau des Automatischen Client

Der generelle Aufbau des oben beschriebenen Workflow–Clients ist auch im Automatischen Client so übernommen und in Abbildung 4 dargestellt. Die Arbeitsliste und die auszuführenden Arbeitsschritte werden im *WorklistHandler* verwaltet. Alle Arbeitsschritte, die in der Arbeitsliste auftauchen werden vom *WorklistExecutor* ausgeführt, indem für jeden Schritt ein *WorklistExecutable*-Objekt erzeugt wird und in einem eigenen *Thread*[2] [GPB+06] der Arbeitsschritt gestartet wird. Das wird über die Ausführungseinheit und die Laufzeitumgebung realisiert. Die angegebenen Handlungen sind in Abbildung 6 dargestellt.

[2]Ein *Thread* ermöglicht die Ausführung von mehreren Aufgaben parallel, indem jede Aufgabe in einem eigenen *Thread* ausgeführt wird.

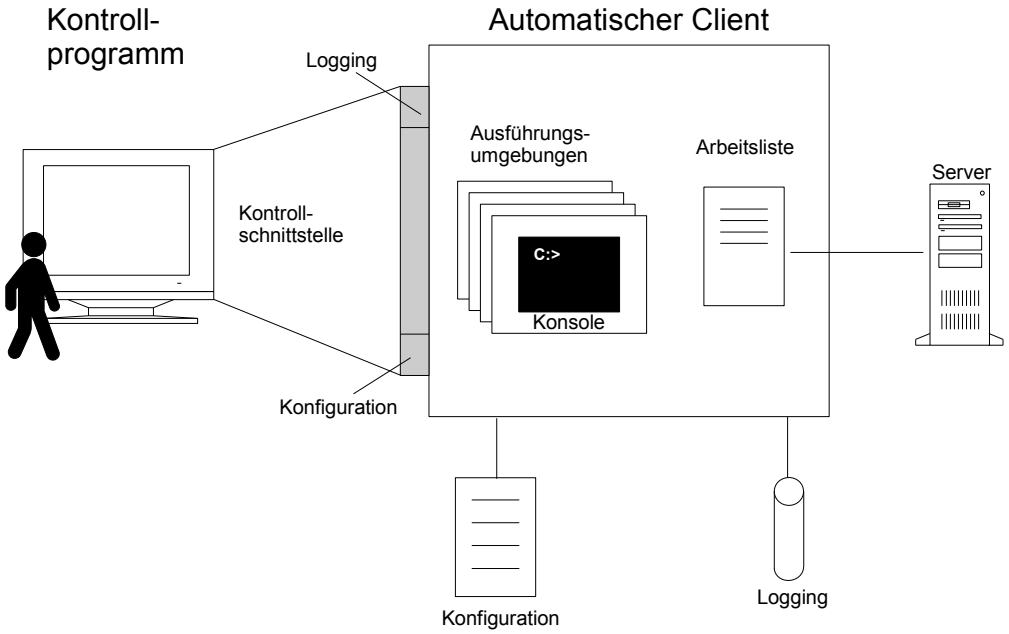

Abbildung 4: Aufbau und Schnittstelle des Automatischen Client

2.3.1 Die Arbeitslistenverwaltung

Um den Client am Server anzumelden, wird von der ADEPT2–Registry der *WorklistManager* abgefragt, der als Schnittstelle zwischen dem Client und dem Server dient. Über diesen wird mit der Methode *LoginAgent(String username, String password, int orgPosition)* der Client am Server angemeldet, wonach der Client seine persönliche Arbeitsliste abrufen kann. Diese Arbeitsliste ist fortan für die Zuweisung von neuen Arbeitsschritten an den Client im Server registriert und der Client erhält Nachrichten, wenn neue Arbeitsschritte zur Verfügung stehen oder wenn andere Änderungen an den Arbeitsschritten stattgefunden haben. Um diese Benachrichtigungen zu erhalten, implementiert der WorklistHandler das Interface *WorklistListener*, dessen Methode *updateWorklist()* vom Server aufgerufen wird, wenn Änderungen anstehen. Beim Aufruf von *updateWorklist()* lädt der *WorklistHandler* die Updates in seine Arbeitsliste und übergibt die zum Abarbeiten bereiten Arbeitsschritte an den *WorklistExecutor* weiter.

2.3.2 Ausführung von Arbeitsschritten

Der *WorklistExecutor* verwaltet alle laufenden Arbeitsschritte. Für die Ausführung verwendet er Threads die mittels eines Threadpools verwaltet werden. Ein Threadpool ist ein Container für eine Menge an Threads die zur Abarbeitung von Aktivitäten verwen-

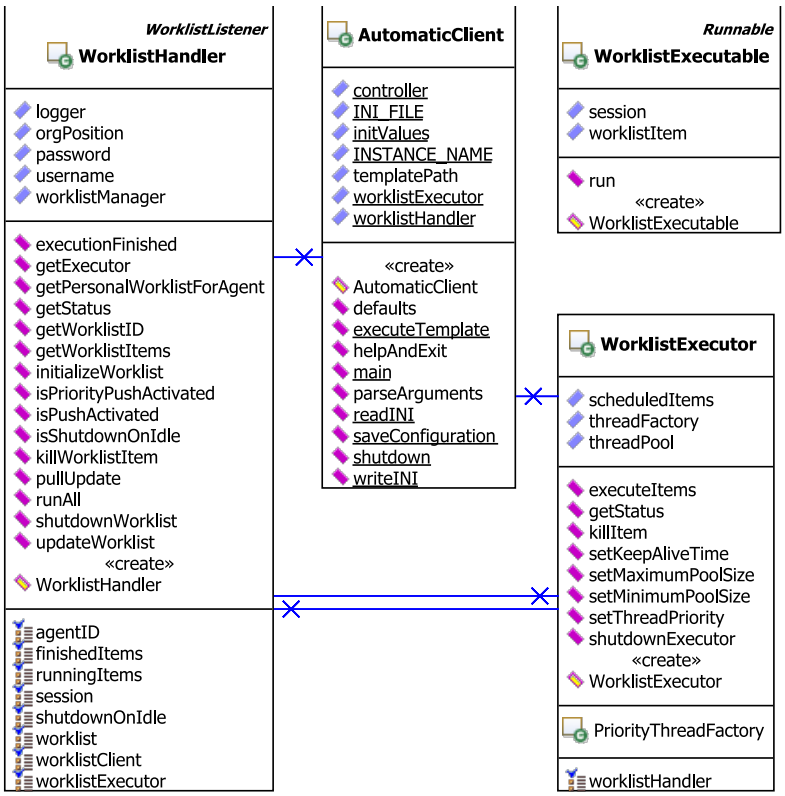

Abbildung 5: Klassendiagramm des Automatsichen Clients

det werden. Wird eine neue Aktivität gestartet, wird ein Thread aus dem Container verwendet bzw falls keiner zur Verfügung steht ein neuer erzeugt und zum Threadpool hinzugegefügt. Um nicht für jede neu auszuführende Aktivität einen neuen Thread erzeugen zu müssen werden Threads, die ihre Aktivität erfolgreich abgearbeitet haben, eine gewisse Zeit als inaktiver Thread vorgehalten. Wird innerhalb dieser Zeit eine neue Aktivität ausgeführt, kann der inaktive Thread verwendet werden. Geschieht dies nicht innerhalb dieser Zeit, wird der Thread beendet und aus dem Threadpool entfernt.

2.4 Anpassungen des Thread–Pools

Der verwendete *ExecutorService* von Java [SM04b] bietet eine komfortable Schnittstelle zum Ausführen von vielen Aktivitäten in einem jeweils eigenen Thread mithilfe eines Threadpools. Durch das Wiederverwenden von Threads wird die aufwändige Erzeugung von Threads minimiert. Zudem werden aber bei langer Inaktivität von Threads deren Ressourcen wieder freigegeben. Durch die Angabe von minimaler und maximaler Anzahl an Threads und der Zeitspanne wie lange ein inaktiver Thread vorgehalten wird,

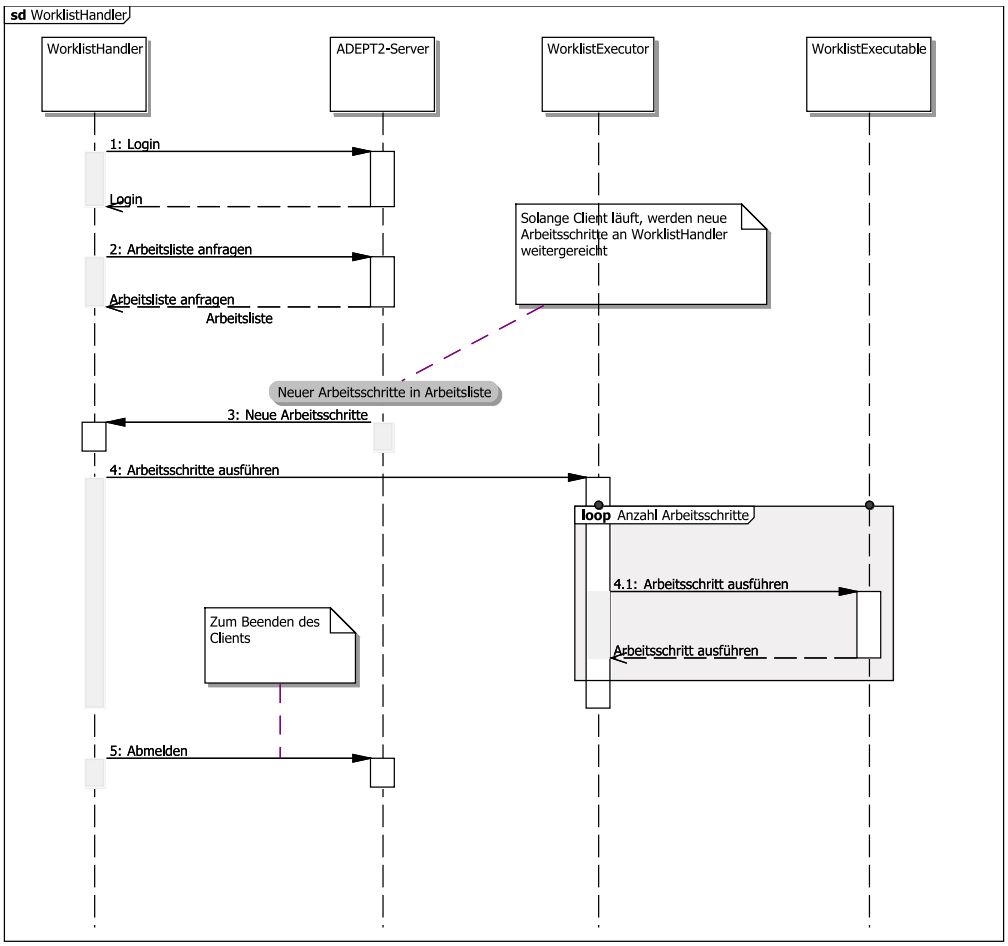

Abbildung 6: Sequenzdiagramm des WorklistHandlers und des WorklistExecutors

kann der Threadpool optimiert werden. Diese Funktionalität wird von dem verwendeten ThreadPoolExecutor schon in vollem Umfang unterstützt.

Eine Anpassung ist aber nötig um die Threads mit einer definierten Priorität[3] laufen zu lassen. Dabei ist darauf zu achten, dass die Priorität nicht höher ist als die des Automatischen Clients selbst, da dann die Verwaltung der Arbeitsliste bei vielen zu bearbeitenden Arbeitsschritten nicht mehr befriedigend erfolgen kann. Deshalb ist die Priorität für die Ausführung von Arbeitsschritten optimaler Weise kleiner zu wählen als

[3]Die Priorität eines Threads beeinflusst die Häufigkeit, wie oft der Thread zur Ausführung kommt. Somit ist eine hohe Priorität ein Indiz dafür, dass ein Thread sehr häufig anderen Aktivitätsträgern vorgezogen wird und somit schneller abgearbeitet wird als ein Thread der mit niedriger Priorität läuft. [Fri02]

die der Verwaltung der Arbeitsliste.

2.4.1 Beenden des Thread–Pools

Beim Beenden des Automatischen Client, bei dem zuerst die Arbeitslistenverwaltung und dann die Ausführungseinheit beendet wird, ist zu beachten, dass die gerade laufenden Arbeitsschritte nicht ohne weiteres unterbrochen werden dürfen. Denn dadurch wäre nicht gewährleistet, dass die Ausführung des Arbeitsablaufes in einem konsistenten Zustand ist und weiter bearbeitet werden kann, wenn mitten in der Abarbeitung eines Arbeitsschrittes dieser unterbrochen wird. Von daher ist es wichtig, dass beim Beenden der Ausführungseinheit solange gewartet wird bis alle gerade laufenden Arbeitsschritte beendet sind. Das wird durch die Methode *awaitTermination()* des *ExecutorService*s gewährleistet, die unter Angabe einer maximalen Zeitspanne wartet, bis kein Thread mehr mit Bearbeiten eins Arbeitsschrittes beschäftigt ist. Erst danach kann der Automatische Client endgültig beendet werden.

2.4.2 Scheduling von Arbeitsschritten

Die Ausführungsreihenfolge der Arbeitsschritte eines Arbeitsablaufes ist im Wesentlichen (bis auf Zweige mit paralleler Ausführung von Arbeitsschritten) von deren Anordnung in der Vorlage abhängig. Werden aber auf einem Server viele Instanzen von Arbeitsabläufen bearbeitet kann ein Automatischer Client beliebig viele Arbeitsschritte gleichzeitig zum Bearbeiten zugewiesen bekommen. Für die Ausführungsreihenfolge dieser Arbeitsschritte lassen sich verschiedene Scheduling–Strategien[4] [Pin01] entwickeln. Dabei ist die einfachste offensichtlich die Ausführung in der Reihenfolge, in der sie vom Server in die Arbeitsliste eingefügt werden. Ein Ansatzpunkt einer sinnvollen Scheduling–Strategie bieten die Arbeitsschritte selbst. Sie haben als eine Eigenschaft eine Frist bis zu der sie abgearbeitet sein müssen. Diese Frist kann verwendet werden um Arbeitsschritte so abzuarbeiten, dass die mit der frühesten Frist als erstes ausgeführt werden. Eine analoge Möglichkeit bietet sich durch eine weitere Eigenschaft, nämlich dem Zeitfenster für das Starten der Ausführung eines Arbeitsschrittes. Dieses ist als maximale Zeitspanne ab der Beendigung des vorausgehenden Arbeitsschrittes bis zum Starten dieses Arbeitsschrittes zu verstehen. Diese beiden Eigenschaften bieten also einen Ansatzpunkt eine gute Abfolge von Arbeitsschritten festzulegen.

Implementiert werden können die beiden vorgestellten Scheduling–Strategien in Java durch die Verwendung einer *PriorityQueue* [SM04a]. Diese verwaltet intern die Elemente in einem Heap[5] [Sch01] und kann über die Angabe eines Vergleichsoperators angepasst

[4] Unter einer Scheduling–Strategie versteht man das Festlegen einer Ausführungsreihenfolge von Aktivitäten

[5] Ein Heap ist eine Datenstruktur die das geordnete Entnehmen von Werten erlaubt und sehr effizient arbeitet.

werden. In dem Fall der frühesten Deadline werden also in der Priority–Queue die Deadlines verglichen und so ein Entnehmen der Arbeitsschritte aus der Queue in der Reihenfolge erreicht, dass der Arbeitsschritt mit der kleinsten Deadline (also dem frühesten Datum) als erstes entnommen und daraufhin ausgeführt wird. Analog kann auch die Implementierung der Zeitfenster erfolgen, wobei nur der Vergleichsoperator der Queue angepasst werden muss, dass er die Zeitfenster miteinander vergleicht.

2.4.3 Die Hauptklasse des Automatischen Clients

Als Einstiegspunkt zum Starten des Automatischen Clients dient die Klasse *AutomaticClient*. Dem Konstruktor der Klasse können Konfigurationsdaten übergeben werden, die dann die bisherigen Werte überschreiben (nähreres siehe Anhang A). Nachdem die Konfiguration vorgenommen und etwaige Änderungen gespeichert wurden, wird zuerst der WorklistExecutor gestartet. Dadurch ist der Client im Grunde bereit zum Ausführen von Arbeitsschritten und der WorklistHandler wird gestartet. Somit ist das System auch soweit Arbeitsschritte vom ADEPT2–Server zugewiesen zu bekommen.

Zudem stellt die Klasse die Funktionen zum sicheren Schließen des Clients zur Verfügung. Dabei werden zuerst die Arbeitslistenverwaltung und danach die Ausführungsumgebung beendet. Diese Reihenfolge ist wichtig, da bei Beendigung der Ausführungsumgebung als erstes die Möglichkeit besteht, dass neue Arbeitsschritte in die Arbeitsliste eingefügt werden, diese aber nicht mehr bearbeitet werden können.

2.5 Schnittstelle für Überwachung und Steuerung

Da der Automatische Client als Kommandozeilenprogramm auf einem beliebigen Rechner laufen kann kann eine Steuerung und Überwachung nicht in ansprechender Art und Weise erfolgen. Deshalb wurde eine Schnittstelle für den Client entwickelt, über die sich ein Benutzer auf den Client aufschalten kann, um den aktuellen Status des Clients einzusehen und bei Bedarf auch eine Steuerung des Clients zu übernehmen. Dieses Aufschalten sollte so ermöglicht werden, dass der Benutzer auch von einem anderen Rechner aus auf den Client zugreifen kann. Für diesen entfernten Zugriff bieten sich zwei Möglichkeiten an, nämlich einmal über ein eigenes Protokoll direkt über eine Netzwerkverbindung oder über *Remote Method Invocation* (RMI) von Java [SM]. RMI ermöglicht den einfachen Aufruf von Methoden an Objekten, die in einer anderen Java Virtual Machine oder auf einem anderen Rechner laufen. Dabei muss sich der Programmierer nicht um die Kommunikation kümmern und kann die Methoden verwenden, als ob sie direkt aufgerufen werden. Die eigentliche Kommunikation wird durch das RMI–Framework übernommen.

Die Entwicklung und Implementierung eines Protokolls für die direkte Kommunikation zwischen dem Kontrollprogramm und dem Automatischen Client über *Sockets* [CD01] bietet größt mögliche Flexibilität. Jedoch ist der Aufwand für die Entwicklung, Implementierung und vor allem das Testen sehr hoch einzuschätzen. Von daher wird für den

Automatischen Client RMI verwendet. Dafür wurden zwei Schnittstellen entworfen, die die Kommunikation zwischen dem Client und dem Kontrollprogramm ermöglichen.

Desweiteren wurde ein Interface entwickelt, dass die Ausgabe von Statusmeldungen in beliebigen Ausgabeströmen ermöglicht. Das dient einerseits der Ausgabe von Meldungen direkt auf der Konsole des Client als auch optional in eine Logging–Datei. Aber auch das Kontrollprogramm kann dieses Interface implementieren und sich beim Automatischen Client registrieren und somit alle Ausgaben die der Automatische Client ausgibt auch auf einem entfernten Rechner ausgeben.

2.5.1 Ausgaben mit dem Logger

Java bietet mit der Java–Logging–API [SM02] eine einfache und komfortable Möglichkeit, um Ausgaben auf unterschiedlichen Medien auszugeben. Da der Automatische Client als Konsolenprogramm läuft, bietet es sich an die Ausgabe auf der Konsole über diese API vorzunehmen und gleichzeitig bietet es sich an, Ausgaben sowohl in eine Datei (für spätere Einsicht) als auch über das Netzwerk anzubieten. Um zu steuern, über welche Medien die Ausgabe erfolgt werden Logging–Handler spezifiziert und der Logger–Instanz bereitgestellt. Für die Konsole wird der mitgelieferte *ConsoleHandler* und für die Ausgabe in eine Datei der *FileHandler* verwendet.

Um einem Kontrollprogramm den Zugriff auf die Ausgaben zu ermöglich, können *SocketHandler* verwendet werden. Diese verbinden sich mit einem Socket und übertragen alle Ausgaben über das TCP–Protokoll [Tan96]. Somit muss ein Kontrollprogramm einen *SocketServer* mit einem Computername und der entsprechenden Port–Nummer erzeugen und an diesem Port darauf warten, dass sich der SocketHandler verbindet. Danach kann das Kontrollprogramm alle Ausgaben wie sie sind entweder ausgeben oder sie nach belieben verwerden.

2.5.2 Der ControllerServer des Automatischen Clients

Um mehreren Kontrollprogrammen gleichzeitig Zugriff auf den Automatischen Client gewähren zu können, ist eine Aufgabe des *ControllerServer* das Registrieren und Entfernen von Controllern. In Abbildung 7 ist die Klassenstruktur des ControllerServer. Die registrierten Controller werden über deren Schnittstelle darauf hingewiesen wenn Änderungen am Automatischen Client passiert sind (wenn beispielsweise neue Arbeitsschritte in die Arbeitsliste eingefügt wurden). Zudem ist der ControllerServer auch die Anlaufstelle für die Controller um Informationen über den Automatischen Client zu erfragen bzw. ihn zu steuern. Dabei können die Informationen variieren von einfachen Statusmeldungen bis zu Konfigurationswerten. Die Steuerung des Client schließt das Ändern der Konfiguration ein aber auch das Starten von Vorlagen von Arbeitsabläufen oder das Beenden des Automatischen Client. All diese Funktionen werden lokalen als auch entfernten Controllern gleichermaßen angeboten. Dabei kann man sich als Einsatzszenario denken, dass ein

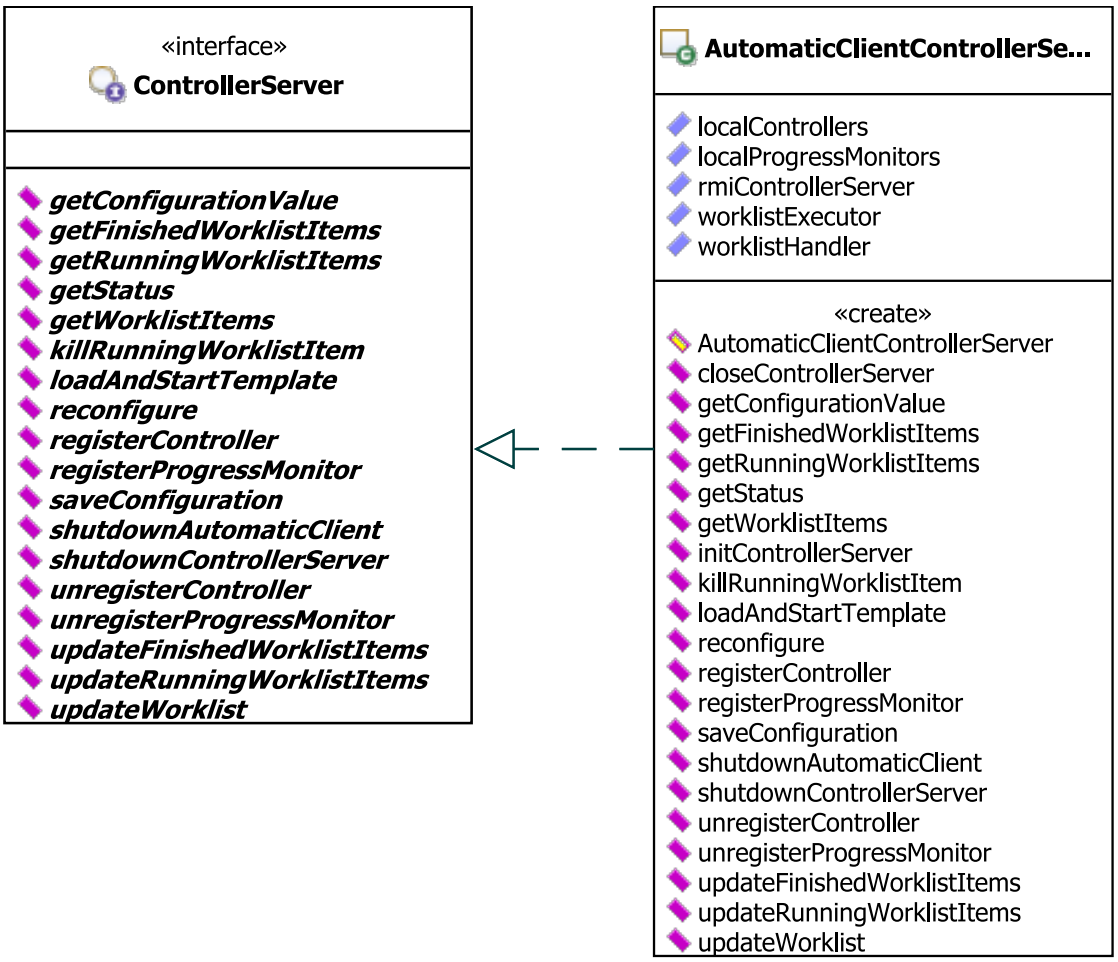

Abbildung 7: Klassendiagramm des ControllerServer

Betriebssystemdienst implementiert wird, der direkt die Ausführung des Automatischen Client anbietet[6]. Entfernte Kontrollprogramme sind natürlich prädestiniert für die Kontrolle eines Automatischen Clients, der auf der Konsole gestartet wurde. Dabei ist es nicht wichtig, ob beide Progamme auf einem Rechner laufen oder auf unterschiedlichen und über ein Netzwerk miteinander verbunden sind.

[6]Die Implementierung mit einem Betriebssystemdienst bietet eine einfache Möglichkeit, um den Automatischen Client beim Systemstart auszuführen, ihn aber auch bequem zu stoppen und neu zu starten. Zudem kann eine einfache grafische Oberfläche zum Steuern des Clients angeboten werden, die die lokale Schnittstelle des Automatischen Client verwendet, also nicht auf RMI angewiesen ist.

2.5.3 Der ControllerClient für das Kontrollprogramm

Um von einem Kontrollprogramm Zugriff auf den Automatischen Client zu erhalten implementiert es das Interface *ControllerClient*. Es definiert nur Methoden um Benachrichtigungen über Änderungen am Client zu erhalten. Die erste Methode *updateWorklistItems()* übergibt eine Liste mit allen Arbeitsschritten die sich aktuell in der Arbeitsliste des Automatischen Clients befinden. Die beiden anderen Methoden sind *updateRunningItems* und *updateFinishedItems()*, die eine Liste mit aktuell ausführenden bzw. mit schon vollständig ausgeführten Arbeitsschritten übergeben. Alle weiteren Informationen die das Kontrollprogramm bentötigt, erhält es über den ControllerServer.

2.6 Zusammenfassung

In diesem Abschnitt wurde der Aufbau eines Clients für ADEPT2 dargestellt und darauf aufbauend die Gemeinsamkeiten und Unterschiede des Automatischen Client erklärt. Desweiteren wurde detailliert auf die Funktionsweise des Clients und die Kommunikation mit dem ADEPT2–Server eingegangen. Zum Abschluss wurde eine Schnittstelle zur Überwachung und Steuerung des Clients entwickelt die sowohl von demselben als auch von einem entfernten Rechner möglich ist.

3 Kontrollprogramm für Automatischen Client

Um die Überwachung und Steuerung des Automatischen Clients zu demonstrieren wurden zwei Kontrollprogramme entwickelt. Zuerst sollte ein kleines Kommandozeilenprogramm einfache Funktionen veranschaulichen, wie Ausführen einer Vorlage, Ausgeben des Status des Automatischen Client und die Ausgabe von Statusmeldungen des Clients über einen *ProgressMonitor*. Um die reichhaltigen Möglichkeiten der Schnittstelle zu veranschaulichen wurde als zweites ein Eclipse–Plug–In[7] entwickelt, welches anhand mehrerer Ansichten die Überwachung von wichtigen Eigenschaften, wie aktueller Arbeitssliste, laufende Arbeitsschritte und abgearbeitete Arbeitsschritte, ermöglicht. Zudem kann der Client bequem über die grafische Oberfläche konfiguriert werden.

3.1 Kommandozeilenprogramm

Das Kontrollprogramm kann mit Kommandozeilenargumenten beim Start konfiguriert werden. Dabei ist vor allem der Pfad der Vorlagendatei anzugeben, sofern diese auf dem Client ausgeführt werden soll. Zudem kann noch die IP–Adresse bzw. der Computername angegeben werden, auf dem der Automatische Client ausgeführt wird, falls dieser nicht auf dem selben Computer läuft wie das Kontrollprogramm. Ein möglicher Programmaufruf zum Starten des Kontrollprogramms um einen Client auf dem Rechner „server“ über den Port „3427“ zu steuern und über diesen eine Vorlagendatei zu starten könnte dann, *java AutomaticClientController -templ“D:/Daten/MeinTemplate.template“ -host“-//server:3427“* lauten.

Nach dem Start erzeugt das Kontrollprogramm einen SocketServer, der am Port 7777 lauscht und Verbindungen annimmt. Daraufhin meldet er sich beim Automatischen Client als Logger–Monitor an. Das führt dazu, dass der Automatische Client seinem Logger einen SocketHandler mit der IP–Adresse des Kontrollprogramms und dem Port 7777 hinzufügt, über den ab diesem Zeitpunkt alle Meldungen mit ausgegeben werden. Dadurch wird eine Verbindung vom Automatischen Client zum Kontrollprogramm geöffnet. Das Kontrollprogramm gibt daraufhin alle Zeichen, die über diese Verbindung versendet werden, auf der Konsole aus. Folglich werden alle Ausgaben des Automischen Client 1:1 auf der Konsole des Kontrollprogramms ausgegeben. Desweiteren wird regelmäßig alle 30 Sekunden der Status des Client abgefragt und auf der Konsole ausgegeben. Die Steuerung (Beenden des Clients, Abbrechen von ausführenden Arbeitsschritten) wird in diesem Kontrollprogramm nicht verwendet.

[7] Ein Eclipse–Plug–In bietet ein Framework um aufbauend auf Eclipse eigene Anwendungen zu entwickeln. Dabei bieten sich insbesondere die Möglichkeiten der komponentenorientierten Entwicklung und die Unterstützung bei grafischen Oberflächen für ein Kontrollprogramm des Automatischen Client an.

3.2 Grafischer Controller als Eclipse–Plug–In

Um die Möglichkeiten der Kontrollschnittstelle des Automatischen Clients besser zu verdeutlichen, wird hier noch ein zweiter Prototyp eines Kontrollprogramms vorgestellt, der eine grafische Oberfläche bietet. Er ist, wie die anderen grafischen Anwendungen im ADEPT2–Projekt, als Eclipse–Plug–In [Ecl04][CR06] realisiert. Dabei kommen verschiedene Ansichten (*Views*) zum Einsatz, die unterschiedliche Informationen über den Automatischen Client darstellen. Die Ansichten sind standardmäßig in einer Perspektive, angeordnet die in Abbildung 13 im Anhang auf Seite 26 zu sehen ist. In den folgenden Abschnitten werden die Struktur des Kontrollprogramms und die einzelnen Ansichten vorgestellt. Im Anhang C sind Bilder zu den einzelnen Ansichten zu sehen.

3.2.1 Plug-In Struktur

Der Aufbau des Kontrollprogramms ist durch die Eclipse–Architektur mit vorgegeben. Es besteht aus mehreren Ansichten, die jeweils einzeln implementiert sind und sich um die Darstellung der Informationen des Automatischen Clients kümmern. Die Informationen selbst werden von der Klasse *AutomaticClientControllerClient* bereitgestellt, die das Interface *ControllerClient* implementiert und die eine Instanz einer Implementierung von *RMIControllerClient* vorhält, die sich um die eigentliche Kommunikation mit dem Automatischen Client kümmert. Somit ist der *AutomaticClientControllerClient* nur ein Wrapper damit die Ansichten und andere Objekte sich nicht um Kommunikationsfehler kümmern müssen, sondern das zentral geschieht. Neben den Ansichten für die Überwachung des Clients ist noch ein Konfigurationsdialog implementiert, der die Konfiguration der Parameter des Clients ermöglicht und eine Einstellungsseite wurde erstellt um die Konfiguration des Kontrollprogramms zu ermöglichen.

3.2.2 Status des Automatischen Client

Der Status des Automatischen Client kann als Zeichenkette von diesem über die Methode *ControllerServer.getStatus()* abgefragt werden. Um den Status anzuzeigen, wird ein Textfeld in einer Ansicht dargestellt und periodisch der Status des Automatischen Clients abgefragt und in dem Textfeld dargestellt. Das Intervall in dem der Status abgefragt wird, kann auf der Eigenschaften–Seite des Plug–Ins eingestellt werden.

3.2.3 Listen mit Arbeitsschritten

Der Automatische Client bietet 3 Kategorien von Listen mit Arbeitsschritten an. Das sind zum einen die Arbeitsschritte die aktuell in der Arbeitsliste zum Abarbeiten vorgehalten werden, die Arbeitsschritte die gerade ausgeführt werden und die schon abgearbeiteten Arbeitsschritte. Für alle drei Kategorien bietet die Kontrollschnittstelle des Clienten eine

Abbildung 8: Klassendiagramm der Ansichten des Kontrollprogramms

Benachrichtigungsfunktion für Kontrollprogramme. Das heißt, das Kontrollprogramm erhält bei jeder Änderung der Liste von Arbeitsschriten die komplette Liste und kann diese nach Belieben verwenden. In diesem Fall des Kontrollprogramms werden drei Ansichten angeboten, eine für jede der genannten Kategorie, die die Arbeitsschritte in einer Liste anzeigen. Dabei werden aber unterschiedliche Eigenschaften der Arbeitsschritte neben dem Namen und der ID angezeigt. Bei den Schritten der Arbeitsliste ist es nur der aktuelle Status[8], bei den aktuell laufenden Arbeitsschritten ist es die Zeit, zu der die Ausführung gestartet wurde und bei den abgearbeiteten Arbeitsschritten ist es zusätzlich zur Start- auch die Endzeit, also wann die Abarbeitung erfolgreich abgeschlossen wurde.

Die beiden Ansichten der Arbeitsliste und der Liste der abgearbeiteten Arbeitsschritte sind beide nur passiv, das heißt sie zeigen nur den aktuellen Stand an. Dem gegenüber bietet die Liste der gerade laufenden Arbeitsschritte noch eine Zusatzfunktion. Über sie kann man durch üeber ein Kontext–Menü den Automatischen Client anweisen einen Arbeitsschritt anzuhalten[9] oder abzubrechen. Ist dies nicht möglich, weil der Arbeitsschritt das Abbrechen nicht unterstützt, wird stattdessen eine Fehlermeldung angezeigt.

[8] Mögliche Werte sind BEREIT, LAUFEND, ANGEHALTEN.

[9] Das Anhalten von Arbeitsschritten wird beim ADEPT2–System dazu verwendet, einen schon gestarteten Arbeitsschritt einem anderen Benutzer zu übertragen, damit dieser ihn weiter bearbeitet.

Abbildung 9: Klassendiagramm der Verbindung des Kontrollprogramms zum Automatischen Client

3.2.4 Konfiguration des Clients

Die Konfiguration des Automatischen Clients ist neben der direkten Konfiguration wie in Anhang A beschrieben auch über die Kontrollschnittstelle möglich. Um dies zu demonstrieren, wurde ein Konfigurationsdialog dem Kontrollprogramm beigegeben, der im Menü unter *Client->Configuration...* aufgerufen werden kann. Der Dialog bietet die Konfigurationsmöglichkeiten an, deren Änderung zur Laufzeit sinnvoll sind. Darunter fallen alle Eigenschaften außer Benutzername, Passwort und RMI–spezifischen Konfigurationen. Die Änderungen die in dem Konfigurationsdialog getätigt werden werden direkt an den Automatischen Client weitergegeben und dieser setzt die entsprechenden Werte in den betroffenen Komponenten. Somit sind die Änderungen auch sofort durch den Status des Clients überprüfbar. Im Konfigurationsdialog kann auch noch angegeben werden ob die Änderungen temporär nur für diese Sitzzung des Clients erfolgen sollen oder ob die Konfiguration auch dauerhaft in der INI–Datei gespeichert werden soll.

3.2.5 Konfiguration des Kontrollprogramms

Auch das Kontrollprogramm bietet konfigurierbare Eigenschaften. Es werden zwei Eigenschaften zum Konfigurieren bereit gestellt, die der RMI–Verbindungsdaten zum Automatischen Client und das Aktualisierungsintervall der Statusmeldung des Automatischen Client. Die RMI–Verbindungsdaten bestehen aus der IP–Adresse bzw. des Computerna-

mens des Rechners, auf dem der Client gestartet wurde und dem Port, an dem der Client auf Verbindung von Kontrollprogrammen annimmt.

3.3 Zusammenfassung

Ausgehend von der Kontrollschnittstelle des Automatischen Clients wie im vorherigen Abschnitt beschrieben, wurden zwei Arten von Kontrollprogrammen dargestellt und damit die Möglichkeiten der Schnittstelle aufgezeigt. Dafür wurde zuerst ein minimales Kommandozeilenprogramm erklärt, das neben dem Starten einer Vorlage eines Arbeitsablaufes nur die Meldungen des Automatischen Clients auf der Kommandozeile ausgibt. Danach wurde ein Kontrollprogramm vorgestellt, welches als Eclipse–Plug–In über eine grafische Oberfläche die wichtigsten Informationen des Automatischen Clients darstellt und auch die Konfiguration des Clients erlaubt.

4 Zusammenfassung und Ausblick

In diesem letzten Teil der Ausarbeitung wird eine Zusammenfassung über die gesamte Arbeit gegeben. Zum Abschluss werden die Möglichkeiten einer Weiterentwicklung des Automatischen Clients und damit verbunden auch der Schnittstelle für Kontrollprogramme gegeben.

4.1 Zusammenfassung

Um sich wiederholende Arbeitsabläufe in Unternehmen mit Software–Systemen zu unterstützen werden Prozess–Managemenet–Systeme eingesetzt, die die Zuweisung von Arbeitsschritten an Benutzer, die Überwachung der Ausführung dieser Arbeitsschritte und Verwaltung der verwendeten Daten anbieten. Die Arbeitsschritte lassen sich in zwei Gruppen einteilen, die automatischen und die manuellen Arbeitsschritte. Automatische Arbeitsschritte sind die, die auch ohne Eingriff von einem menschlichen Benutzer bearbeitet werden können, während bei Manuellen ein Benutzer mit eingebunden werden muss.

Für die Ausführung der automatischen Arbeitsschritte bietet es sich an, dass diese auf einem extra Client stattfindet. Dieser Automatische Client kann den Aufbau eines normalen Client haben und die Zuweisung der Arbeitsschritte an den Client wird anhand der Benutzerzuordnung der einzelnen Arbeitsschritte durchgeführt. Der Client erhält nach der Anmeldung am System eine Liste mit auszuführenden Arbeitsschritten und führt diese in jeweils einem eigenen Thread aus.

Der Automatische Client läuft als Konsolenprogramm und bietet somit nur bedingt Möglichkeiten zur Ausgabe von Informationen. Zudem bietet es sich nicht an eine Steuerung über Eingaben an der Konsole anzubieten. Deshalb wurde eine Schnittstelle entwickelt über die von einem anderen Programm auf dem selben oder einem entfernten Rechner der aktuelle Status abgefragt werden kann. Zudem ist es möglich den Automatischen Client über diese Schnittstelle zu konfigurieren und zu beenden.

Um die Fähigkeiten der Kontrollschnittstelle zu demonstrieren wurden zwei Kontrollprogramme vorgestellt, eines für die Konsole das nur die Ausgaben des Client auf der eigenen Konsole ausgibt und ein umfangreicheres mit einer grafischen Oberfläche, welches sowohl Informationen über den Client anzeigt als auch die Konfiguration und Steuerung des Clients unterstützt.

4.2 Ausblick

Die Kommunikation zwischen dem Automatischen Client und den Kontrollprogrammen basiert derzeit auf RMI. Das hat den Vorteil, dass auf einem Rechner auf dem das Kontrollprogramm ausgeführt wird nur dieses installiert sein muss. Allerdings bietet das ADEPT2–System an diese Kommunkation über einen ADEPT2–Service einfacher zu gestalten. Damit ist es möglich den Automatsichen Client als Service zu implementieren

und am ADEPT2–System zu registrieren. Dann kann ein Kontrollprogramm über eine einfache Schnittstelle an der ADEPT2–Registry diesen Dienst anfragen und erhält einen Stumpf (*stub*) einer Client–Klasse. Die Methoden dieser Klasse können dann verwendet werden um auf die Funktionen des Automatischen Clients zugreifen zu können. Dabei muss man sich nicht um die Kommunikation kümmern die zwischen den beiden Programmen abläuft, da diese vom ADEPT2–System übernommen wird, der entfernte Aufruf der Methoden erfolgt also vollkommen transparent.

Die Verwendung der ADEPT2–Services bedeutet dann allerdings, dass zumindest die Kernkomponenten des ADEPT2–Systems auch auf dem Rechner installiert sein müssen, auf dem das Kontrollprogramm läuft. Sind diese Vorraussetzungen erfüllt, kann noch untersucht werden, inwiefern schon vorhandene Views von ADEPT2 für den Einsatz in Kontrollprogrammen geeignet sind. Dabei ist insbesondere des *WorklistVisualization–Plugin* zu untersuchen. Dieses Plug–In bietet eine Ansicht an, in der die aktuelle Arbeitsliste angezeigt wird und kommt auch im „normalen“ Client von ADEPT2 zum Einsatz. Als Besonderheit für den Automatischen Client muss überprüft werden, ob die Zuweisung des Worklist–Managers des Automatischen Client funktioniert und ob es Probleme bereitet, dass ein Benutzer über diese Ansicht Arbeitsschritte in der Liste ausführen kann. Insbesondere die Möglichkeit der Ausführung von Arbeitsschritten sollte aber kein Problem darstellen, da der Automatische Client alle Arbeitsschritte die er zugewiesen bekommt sofort ausführt. Insofern sollte jeder Arbeitsschritt in der dargestellten Arbeitsliste schon gestartet sein.

Literatur

[Abt95] Abteilung DBIS - Universität Ulm. ADEPT - Next Generation Workflow Technology (since 1995), 1995. URL: `http://www.informatik.uni-ulm.de/dbis/index01.htm?01/forschung/forschung-m.htm` [Online; Zugriff 24. September 2007].

[Abt04] Abteilung DBIS - Universität Ulm. Next Generation Enterprise Process Management: Component-oriented Development of Adaptive Process-oriented Enterprise Software, 2004. URL: `http://www.aristaflow.de/` [Online; Zugriff 24. September 2007].

[CD01] Kenneth L. Calvert and Michael J. Donahoo. *TCP/IP Sockets in Java: Practical Guide for Programmers (The Practical Guides)*. Morgan Kaufmann, 2001.

[CR06] Eric Clayberg and Dan Rubel. *Eclipse: Building Commercial-Quality Plug-ins (2nd Edition) (The Eclipse Series)*. Addison-Wesley Professional, 2006.

[Ecl04] Eclipse.org. The Eclipse homepage, 2004. URL: `http://www.eclipse.org/` [Online; Zugriff 15. September 2007].

[Fri02] Jeff Friesen. Achieve Strong Performance with Threads, Part 1. *JavaWorld*, 2002. URL: `http://www.javaworld.com/javaworld/jw-07-2002/jw-0703-java101.html?` [Online; Zugriff 20. September 2007].

[GPB+06] Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea. *Java Concurrency in Practice*. Addison-Wesley Professional, 2006.

[Hol05] David Hollingsworth. Workflow Management Coalition – The Workflow Reference Model, 2005. URL: `http://www.wfmc.org/standards/docs/tc003v11.pdf` [Online; Zugriff 25. September 2007].

[IBM] IBM. WebSphere MQ Workflow. URL: `http://www-306.ibm.com/software/integration/wmqwf/` [Online; Zugriff 24. September 2007].

[Kre02] Ulrich Kreher. Performanzaspekte bei der Programmierung mit Java. Diplomarbeit, Universität Ulm, Fakultät für Informatik, 2002.

[Pin01] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems (2nd Edition)*. Prentice Hall, 2001.

[Rei00] Manfred Reichert. Dynamische Ablaufänderungen in Workflow-Management-Systemen. Dissertation, Universität Ulm, Fakultät für Informatik, 2000.

[Sch01] Uwe Schöning. *Algorithmik*. Spektrum Akademischer Verlag GmbH, Heidelberg-Berlin, 2001.

[SM] Inc Sun Microsystems. Java Remote Method Invocation - Distributed Computing for Java. URL: `http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp` [Online; Zugriff 19. September 2007].

[SM02] Inc. Sun Microsystems. JavaTM logging overview, 2002. URL: `http://java.sun.com/javase/6/docs/technotes/guides/logging/overview.html` [Online; Zugriff 20. September 2007].

[SM04a] Inc Sun Microsystems. Class PriorityQueue<E>, 2004. URL: `http://java.sun.com/j2se/1.5.0/docs/api/java/util/PriorityQueue.html` [Online; Zugriff 16. September 2007].

[SM04b] Inc Sun Microsystems. Interface ExecutorService, 2004. URL: `http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/ExecutorService.html` [Online; Zugriff 15. September 2007].

[Tan96] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, Upper Saddle River, 1996.

[TIB] TIBCO. TIBCO iProcess Suite. URL: `http://www.tibco.com/software/business_process_management/iprocess_suite/default.jsp` [Online; Zugriff 24. September 2007].

[Ult] Ultimus. Ultimus BPM Suite. URL: `http://www.ultimus.com/` [Online; Zugriff 24. September 2007].

A Konfiguration des Automatischen Client

Es werden drei Wege angeboten um die Standardkonfiguration des Automatischen Client anzupassen. Der Client kann durch Übergabe von Kommandozeilenargument beim Start an die persönlichen Bedürfnisse angepasst werden. Alle geänderten Konfigurationswerte werden in einer INI–Datei gespeichert. Diese kann vom Benutzer auch bearbeitet werden. Oder es wird ein Kontrollprogramm verwendet wie in Abschnitt 2.5 genauer erläutert wurde.

Für den Automatischen Client werden folgende Konfigurationsparameter angeboten, wobei der angegebene Bezeichner für die Konfiguration in der Ini–Datei verwendet wird und hinter der Erklärung die Verwendung des Parameters auf der Kommandozeile in Klammern beschrieben ist:

UserID Der Benutzername mit dem sich der Automatische Client am ADEPT2–Server anmeldet. (-u“*Name*“)

Password Das Passwort des automatischen Benutzers. (-pw“*Passwort*“)

orgPosition Die Position die der Benutzer in der Organisation einnimmt. (-orgp$X$)

minPoolSize Die minimale Anzahl bereitgestellter Threads für die Ausführung der Arbeitsschritte. (-min$X$)

maxPoolSize Die maximale mögliche Anzahl an Threads für die Ausführung der Arbeitsschritte. (-max$X$)

keepAliveTime Die Zeitspanne die ein inaktiver Thread im Thread–Pool verweilt bevor er terminiert wird. (-keepAlive$X$)

workerPriority Die Priorität mit der Threads laufen, die Arbeitsschritte ausführen. (-prio$X$)

verboseLevel Ein Filter für die Ausgabe von Nachrichten auf der Konsole. (-vX)

rmiControllerServer Gibt an, ob ein Controller Server für entfernte Controller angeboten werden soll (siehe nächster Abschnitt). (-rmi$Y|N$)

rmiHost Es kann eine spezifische IP angegeben werden, an die der RMI–Controller–Server gebunden wird, er also nur Verbindungen von Kontrollprogrammen annimmt, die sich über eine entsprechende IP–Adresse anmelden. (-rmiHost*xxx.xxx.xxx.xxx*)

rmiPort Der Port an dem der RMI–ControllerServer Verbindungen von Kontrollprogrammen annimmt. (-rmiPort$X$)

B Klassendiagramme

In diesem Abschnitt werden die wichtigsten Klassen und deren Zusammenhänge des Automatischen Clients als Klassendiagramme dargestellt.

Die Klassen in Abbildung 10 zeigen die Schnittstelle die für die Kommunikation zwischen dem Automatischen Client und Kontrollprogrammen über RMI verwendet werden.

Abbildung 10: RMIControllerServer

Die folgenden beiden Abbildungen 11 und 12 stellen die Schnittstelle der Kontrollprogramme dar, die diese Implementieren müssen um entweder lokal (ControllerClient in Abbildung 11) oder entfernt (RMIControllerClient in Abbildung 12) auf den Automatischen Client zugreifen zu können. Dabei ist jeweils nur ein Interface zu implementiern. Die Klasse *AutomaticClientMonitor* stellt dabei das einfache Kontrollprogramm dar, das in Abschnitt 3.1 auf Seite 14 beschrieben ist.

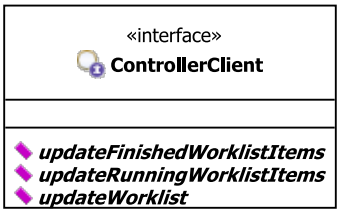

Abbildung 11: ControllerClient

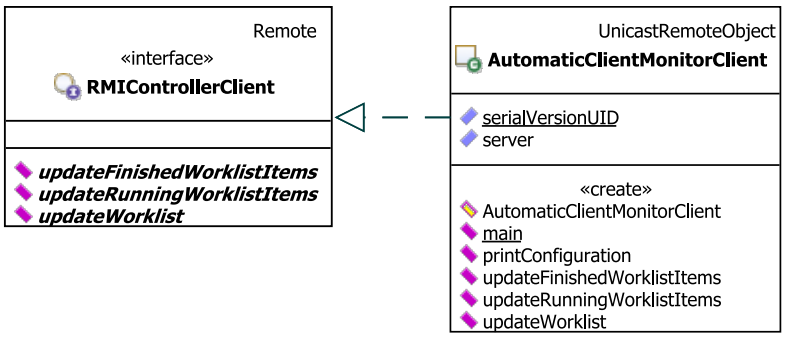

Abbildung 12: RMIControllerClient

C Screenshots des Kontrollprogramms

In diesem Abschnitt werden einige Bilder des Kontrollprogramms mit grafischer Oberfläche gezeigt. Dabei sind die einzelenen Ansichten der Arbeitsliste, die Statusmeldung des Automatischen Clients als auch der Konfigurationsdialog für den Automatischen Client gezeigt.

Die Ansichten sind beim ersten Start des Kontrollprogramms wie in Abbildung 13 angeordnet. Dabei sind die Ansichten der Arbeitsliste links und der Status des Automatischen Clients rechts angeordnet.

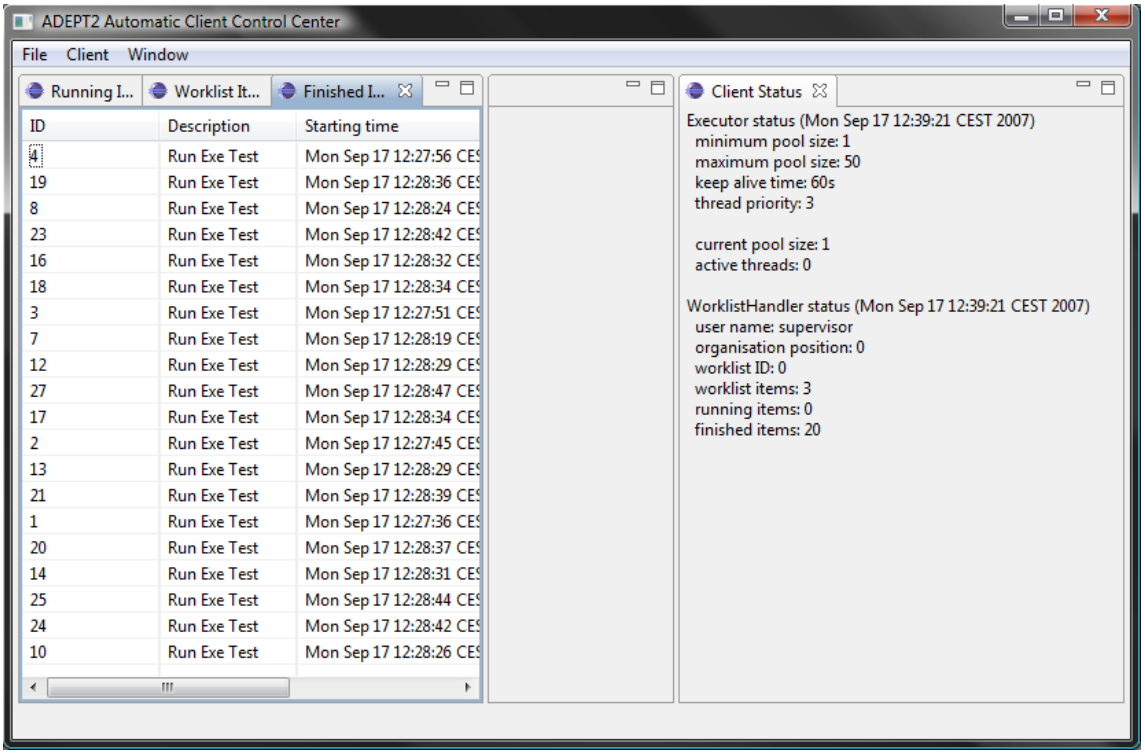

Abbildung 13: Die Ansicht des Kontrollprogramms

Die Statusmeldung des Automatischen Client umfasst neben der Konfiguration des WorklistHandlers und des WorklistExecutors auch die aktuellen Zahlen bezüglich der Ausführung von Arbeitsschritten, also wieviele sich gerade in der Arbeitsliste befinden, wieviele ausgeführt werden und wieviele schon komplett abgearbeitet worden sind. Die Meldung wird in einfacher textueller Form wie in Abbildung 14 angezeigt.

Die Ansicht in Abbildung 15 zeigt die Liste der Arbeitsschritte die momentan in der Arbeitsliste auf die Ausführung warten bzw. schon gestartet sind (dann mit Status „STARTED“). Da der Client alle verfügbaren Arbeitsschritte sofort ausführt, sollten in der Liste

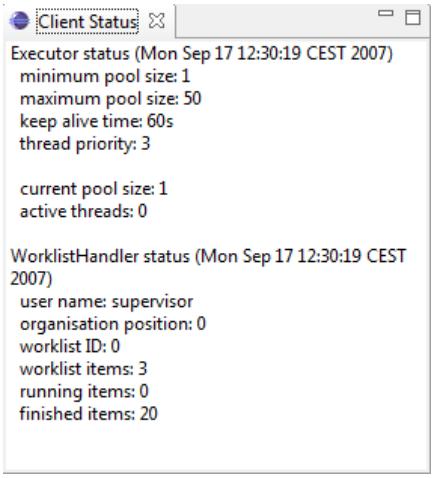

Abbildung 14: Die Statusmeldung des Automatischen Client

nur schon gestartete Arbeitsschritte auftauchen.

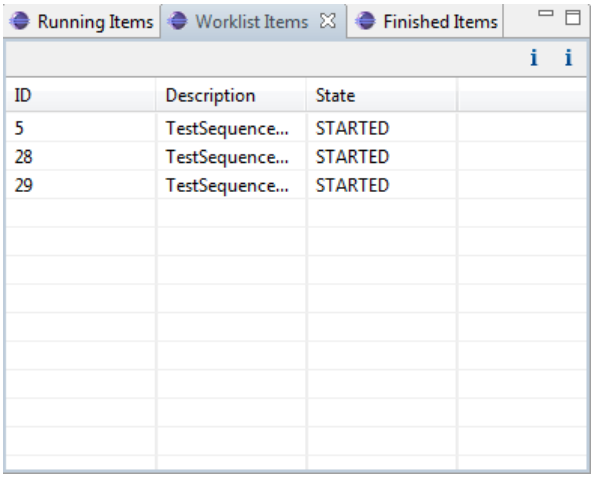

Abbildung 15: Die Arbeitsliste mit aktuellen Arbeitsschritten

In der Ansicht in Abbildung 16 werden alle Arbeitsschritte angezeigt, die gerade ausgeführt werden. Dabei besteht die Möglichkeit über einen Doppleklick auf einen Schritt diesen in der Abarbeitung abzubrechen.

Die Ansicht in Abbildung 15 zeigt die Liste der fertig ausgeführten Arbeitsschritte. Die Liste enthält alle Arbeitsschritte die seit dem Start des Automatischen Client erfolgreich bearbeitet wurden.

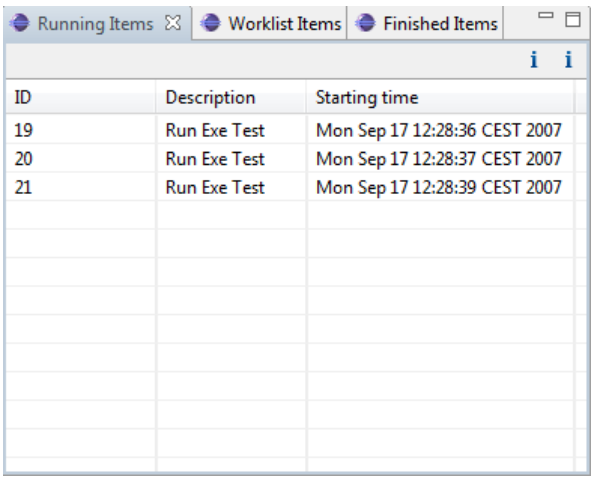

Abbildung 16: Die Liste mit gerade ausgeführten Arbeitsschritten

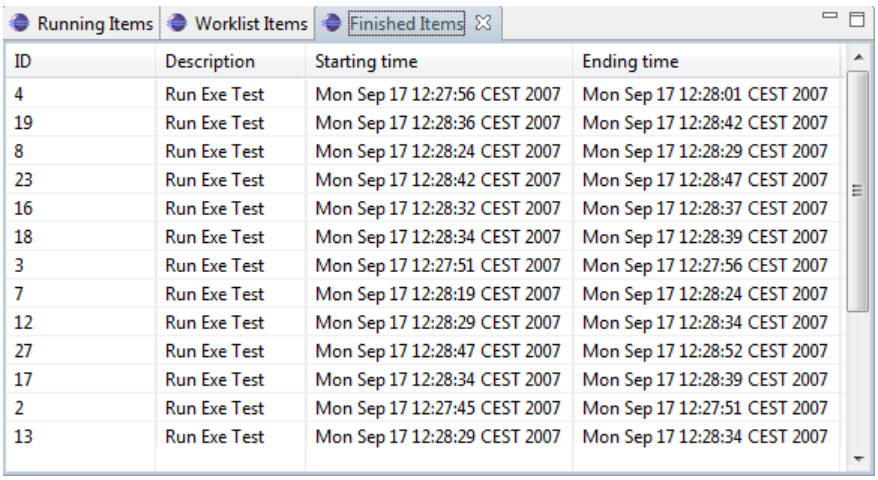

Abbildung 17: Die Liste mit abgearbeiteten Arbeitsschritten

In dem Dialog in Abbildung 18 kann der Automatische Client neu konfiguriert werden. Dafür werden die Daten in die entsprechenden Felder eingetrage und mit setzen der Option für permanentes Speichern der Konfiguration kann angegeben werden, dass auch bei einem Neustart des Automatischen Clients die Änderungen erhalten bleiben.

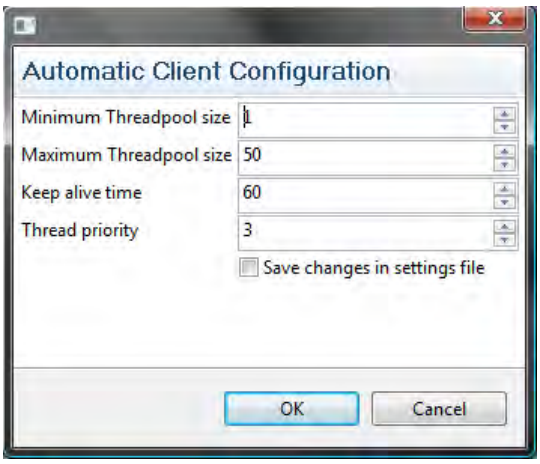

Abbildung 18: Der Konfigurationsdialog für den Automatischen Client

Erklärung

Matthias Rink, Matrikel-Nr.: 526057

Ich erkläre, daß ich die Bachelorarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 04.10.2007