Migration Guide

RELEASE 1.3

24. Oktober 2006

Jens Kupferschmidt (Leipzig)

Thomas Scheffler (Jena)

Kathleen Krebs (Hamburg)

Vorwort

Diese Migrationsanleitung soll Anwendungsentwicklern helfen, ihre bestehende MyCoRe-Anwendung an die neuen Mechanismen und Strukturen der Version 1.3 von MyCoRe, anzupassen.

Inhaltsverzeichnis

1 Migration von MyCoRe 1.2 nach 1.3

1.1 Allgemeines

Im Rahmen der Weiterentwicklung des MyCoRe-Kerns gibt es tiefgreifende Veränderungen, die sich auch auf die Anwendungsebene auswirken. Neu hinzugekommen bzw. geändert haben sich folgende Punkte:

- Neues, umfangreiches und dadurch sehr mächtiges Rechte- und Rollensystem (ACLs)
- neue Mechanismen beim Indizieren, Ablegen und Suchen der Metadaten machen MyCoRe-Anwendungen um ein Vielfaches performanter
- Neues Modul zur Bildbetrachtung (ImageViewer)
- Neuer Datentyp MCRMetaISO8601Date
- Internationalisierung mit I18N
- teilweise Umstrukturierung und Neubenennung der Layout-Stylesheets in der Beispielanwendung DocPortal
- Layout des WCMS überarbeitet und neuen WYSIWYG-Editor (FCK) eingebunden, außerdem sind die Webseiten nach einem WCMS-Login direkt aus der Anwendung bearbeitbar
- Verwendung des FCK-Editors im Editorframework, ermöglicht die Kombination von Eingabefeldern mit dem HTML-Editor
- Backend-Implementierung für den Content Manager modularisiert und überarbeitet (noch in Arbeit)

Anwendungen, die den MyCoRe-Kern v1.2 oder älter verwenden müssen angepasst werden, um diese neuen Funktionen nutzen zu können. Auch von den Geschwindigkeitsvorteilen kann der Endnutzer erst profitieren, wenn die Anwendung die neuen Mechanismen auch verwendet.

In den nachfolgenden Abschnitten sind die einzelnen Schritte für eine Anwendungsmigration detailliert beschrieben. Dass die Migration einen solchen Umfang erreicht hat bitten wir, die Entwickler, zu entschuldigen. Jedoch war dieses Vorgehen unumgänglich, da nur so Geschwindigkeit, Zuverlässigkeit sowie einfachere Wartung und Pflege der MyCoRe-Anwendungen für die Zukunft gewährleistet werden kann. Für zukünftige Versionen garantieren die Entwickler jedoch für mehr Konsistenz.

Hinweis: Viele `properties` und Klassen aus MyCoRe v1.2 sind noch in v1.3 enthalten, was eine Abwärtskompatibilität zu alten Anwendungen gewährleistet. Jedoch sind diese als `deprecate` markiert und werden in der nächsten Version nicht mehr enthalten sein.

1.2 Vorbereitung

1.2.1 Download

Download des neuen MyCoRe-Kerns via CVS oder auf den MyCoRe-Webseiten. Prüfen der `build.properties` im `config`-Verzeichnis. Die Anbindung an ein Datenbank-Backend funktioniert nun standardmäßig mittels Hibernate. Dies ist auch die empfohlene Konfiguration, was im Normalfall keine Änderung in den `build.properties` notwendig macht. Sollen noch weitere Module, wie beispielsweise der ImageViewer oder die Unterstützung für den IBM-Content-Manager, eingebunden werden, so müssen diese entsprechend der Anleitung im UserGuide aktiviert werden. Abschließend wird der Kern mit Hilfe des bekannten Aufrufes '`ant jar`' übersetzt. Die sich daraus ergebende Datei hat – im Vergleich zu alten MyCoRe-Versionen - nun nur noch die Backend-unabhängige Bezeichnung `mycore.jar`.

Als nächsten Schritt empfiehlt es sich DocPortal v1.3 ebenfalls via CVS oder die MyCoRe-Webseite herunterzuladen. Damit ist die Basis für die Migration geschaffen.

1.2.2 Herangehensweise

Je nachdem wie nah die eigene Anwendung an der Beispielanwendung DocPortal ist, empfehlen sich zwei Herangehensweisen:

1. Anpassung von DocPortal durch Übernahme eigener Daten, Layout, Such- und Eingabemasken. Dies empfiehlt sich vor allem für Anwendungen, die nahe am DocPortal sind. Das bedeutet, dass es keine größeren Änderungen an Layout und Datenmodell vorgenommen wurde und auch die Such- und Eingabemasken kaum bis gar nicht modifiziert worden.
2. Übernahme neuer Dateien aus DocPortal in die eigene Anwendung und ggf. Anpassung derer. Außerdem sind einige Dateien nicht mehr notwendig und an anderen Stellen müssen Anpassungen gemacht werden. Da dies sehr aufwendig und fehleranfällig ist, empfiehlt sich diese Herangehensweise nur, wenn die Anwendung bereits weit von DocPortal entfernt ist, also eigene Datenmodelle erstellt, entsprechend neue Such- und Eingabemasken definiert wurden und die Anwendung zusätzlich ein stark verändertes Layout hat.

Im nachfolgenden werden beide Vorgehensmodelle erläutert. Vorher noch ein kurzer Überblick der betroffen folgende Komponenten.

1.2.3 Komponenten

Datenmodell

Der alte Datentyp MCRMetaDate wurde durch ein weitaus umfassenderen und flexibleren Typ MCRMetaISO8601Date abgelöst. Außerdem gibt es im `<service>`-Bereich eines MyCoRe-Objektes nun die Möglichkeit, im Rahme eines `<servacl>`-Elementes, ACLs zu definieren.

Suche

Für die neue Suchfunktion ist es notwendig, die durchsuchbaren Felder

separat zu definieren. Auch hat sich die Form des XMLs geändert, das die Treffer zurückliefert. Suchmasken werden ab v1.3 mit Hilfe des Editor-Frameworks erstellt und nicht mehr wie bisher durch eine weitere eigene XML-Spezifikation.

Internationalisierung

MyCoRe v1.3 nutzt für die Mehrsprachigkeit den Java-Standard I18N. Dadurch fallen sämtliche de-, en- etc. Ausprägungen der Stylesheets weg. Bezeichner, die bis jetzt anhand von Variablen in verschiedenen Sprachen definiert wurden, werden so zu Properties, die in einer Datei für jede Sprache definiert und gepflegt werden können. Dies gilt auch für die Definition der Eingabe- und Suchmasken mittels des Editor-Frameworks.

Grund-Layout der Anwendung

Durch eine jetzt klare Aufteilung der für das Layout notwendigen Dateien und derer für das WCMS-Modul zuständigen, haben sich die Positionen und Namen einiger Dateien für statische Webseiten, Gestaltungsvorlagen und Navigationsdefinitionen verändert.

Build-Prozess

Die `build.xml`-Datei wurde grundlegend überarbeitet und sollte für eigene Anwendungen übernommen und neu angepasst werden. Auch die Integration der Module findet mittels ant statt.

Module

Mit v1.3 von MyCoRe gibt es echte Module, die je nach Bedarf verwendet werden können. Auf die konkrete Integration einzelner Module in die eigene Anwendung wird hier nicht weiter eingegangen. Dies ist ausführlich im *UserGuide* beschrieben.

2 Migration einer DocPortal-nahen Anwendung

2.1 Konfiguration

Im `config`-Verzeichnis von DocPortal befinden sich nach dem Download vier `properties`-Dateien:

- `mycore.properties`
- `mycore.properties.application`
- `mycore.properties.deprecate`
- `mycore.properties.private.template`

Die `deprecate`-Datei enthält alle veralteten Konfigurationen. Anwendungsspezifische Konfigurationen können in `mycore.properties.application` angegeben werden. Per Standard ist diese Datei leer. Einzig relevant sind die beiden verbleibenden Dateien. Das `private.template` muss (wie üblich) kopiert und danach angepasst werden. Die Verwendung der Variablen `MCR.basedir` erleichtert hierbei das Anpassen der Pfade.

2.2 Migration des Datenbestandes

Klassifikationen können einfach übernommen werden, hierfür sind keine Anpassungen notwendig. Aufgrund der Abhängigkeiten müssen die Klassifikationen als erstes in die neue Version der Anwendung eingespielt werden.

Mit Hilfe der Dateien `mcr_migration13-object.xsl` und `mcr_migration13-derivate.xsl` im `stylsheets`-Verzeichnis von DocPortal ist es möglich den bestehenden Datenbestand aus der alten MyCoRe-Anwendung zu exportieren, an die Änderungen anzupassen und in die neue Anwendung zu importieren. Vor der Verwendung der XSL-Dateien wird empfohlen, diese zu prüfen. Insbesondere der ACL-Teil, der durch diese Dateien an jedes Dokument, jede Person etc. angehangen wird, kann sich anwendungsabhängig unterscheiden.

Im Einzelnen sind folgende Schritte notwendig:

1. Kopieren der XSL-Dateien in das Stylesheet-Verzeichnis der zu migrierenden Anwendung.
2. Aktualisieren der Commandline, wenn veraltet.
3. Starten der MyCoRe-Kommandozeile: `bin/mycore.sh` bzw. `bin\mycore.cmd`
4. Aufruf des Export-Befehls für jeden MyCoRe-Objekttypen bzw. die Derivate:
   ```
   export object <mcrobjecttype> to directory <export_dir>
   with migration13
   export derivate <mcrderivatetype> to directory
   <export_dir> with migration13
   ```
   Beispiel:
   ```
   export object document to directory export/document
   with migration13
   export derivate derivate to directory export/derivate
   with migration13
   ```
5. Importieren der Daten mit Hilfe der Kommandozeile der neuen Anwendung. Starten dieser durch den Aufruf von: `bin/mycore.sh` bzw. `bin\mycore.cmd` im Home-Verzeichnis der neuen Version der Anwendung.
6. Laden der Daten mittels des Load-Befehls:
   ```
   load all objects from directory <export_dir>
   load all derivates from directory <export_dir>
   ```

Im `bin`-Verzeichnis steht dafür ein Linux-Skript zur Verfügung (`MigrateTo13.sh`), das vor der Verwendung gegebenenfalls angepasst werden muss.

2.3 Suchmasken

Die Suchmasken basieren mit der neuen MyCoRe-Version auf dem Editor-Framework. Die alten XML-Dateien werden nicht mehr verwendet und können gelöscht werden. Die im DocPortal mitgelieferten Suchmasken dienen als Beispiel und Vorlage für eigene Suchmasken. Bevor diese definiert werden können ist es jedoch notwendig, die `searchfields.xml` im `config`-Verzeichnis von DocPortal zu prüfen und ggf. anzupassen. DocPortal-nahe Anwendungen sollten mit den dort definierten Feldern jedoch auskommen.

Die Beispielsuchmasken befinden sich im `config`-Verzeichnis von DocPortal und beginnen mit `editor-search`. Eine ausführliche Dokumentation dazu ist Teil des *Programmer Guides* (siehe Kapitel 3.1).

2.4 Ergebnislisten

Je MyCoRe-Objekttyp gibt es jetzt nur noch eine XSL-Datei. Am Beispiel DocPortal: Im `stylesheet`-Verzeichnis befinden sich die Dateien `author.xsl`, `document.xsl` und `institution.xsl`. Darin sind alle Darstellungen, egal ob innerhalb der Ergebnisliste oder als Detailansicht, festgelegt.

Ergebnislisten sind umrahmt von dem XML-Element `mcr:hit`. Entsprechend muss dieses Template nach Bedarf angepasst werden. Nachstehend ein Code-Auszug aus document.xsl, um den passenden Code-Schnipsel leichter zu finden:

```
<!-- Template for result list hit -->
<xsl:template match="mcr:hit[contains(@id,'_document_')]">
  <xsl:param name="mcrobj" />
  <xsl:param name="mcrobjlink" />
  <xsl:variable name="DESCRIPTION_LENGTH" select="100" />
  [...]
</xsl:template>
```

2.5 Detailansicht eines Treffers

Die Detailansicht wird wie bereits in Abschnitt 2.4 beschrieben, ebenfalls in der XSL-Datei des jeweiligen MyCoRe-Objekttyps definiert. Die jeweilige Spezifikation kann aus der zu migrierenden Anwendung übernommen werden. Einzig der XPath hat sich etwas geändert, da es hier kein umschließendes `mcrresult`-Element gibt. Durch Suchen nach `mcr_result/mycoreobject` und Ersetzen dessen durch Nichts, sind die hier notwendigen Anpassungen abgeschlossen.

Welche Objekttypen es gibt bzw. welche XSL-Dateien für die Darstellung der Ergebnisse oder Details notwendig sind, gibt man in der Datei `objecttypes.xsl` bekannt. Auch diese befindet sich im `stylesheet`-Verzeichnis von DocPortal und kann ggf. angepasst werden.

2.6 Eingabemasken

Die Eingabemasken werden wie bereits in älteren MyCoRe-Versionen mit Hilfe des Editor-Frameworks erstellt. Dies funktioniert auch weiterhin auf die bekannte Art und Weise. Jedoch sind einige Neuerungen hinzugekommen, die der Anwendungsentwickler nutzen sollte. So sollten alle Klassifikationsaufrufe, die früher mit Hilfe des `MCRQueryServlets` realisiert wurden, durch den Aufruf des URI-Resolvers abgelöst werden. Das nachstehende Beispiel aus `import-commons.xml` (zu finden im `config`-Verzeichnis von DocPortal) enthält beispielhaft einen solchen URI-Aufruf für die Klassifikation `DocPortal_class_00000002`.

```
<list id="COriginPers" width="300" type="dropdown">
   <item value="">
     <label xml:lang="de">(bitte wählen)</label>
     <label xml:lang="en">(select please)</label>
   </item>
   <include uri="classification:editor:-1:children:
               DocPortal_class_00000002"
          cacheable="false"/>
   <condition id="CoOriginPers" required="true" >
     <label xml:lang="de">Bitte wählen Sie einen Wert aus!</label>
     <label xml:lang="en">Please select one entry!</label>
   </condition>
 </list>
```

QueryServlet-Aufrufe fuer Personen-Listen muessen ebenfalls angepasst werden!

<subselect id="sub.select1" type="webpage"
href="indexpage?searchclass=creators_sub" sortnr="6" >
und wenn ich das richtig sehe, sind die ...-to-items.xsl-Dateien weg
also je nachdem was du als searchclass angibst, musst du eine entsprechend benamste xsl-Datei haben
also fuer href="indexpage?searchclass=creators_sub" brauchst du eine indexpage-creators_sub.xsl

3 Migration einer DocPortal-fernen Anwendung

[ToDo]

3.1 Konfiguration

- **Build-Prozess** – hier ist jedem Entwickler dringend anzuraten, die entsprechenden *build.xml*-Files gemäß den Vorlagen anzupassen. Mit der Datei *common-modules.xml* wurden mehrfach benötigte Definitionen ausgelagert. Dies verbessert die Handhabung des Build-Prozesses.

- **Suche** – um die Indizierung der zu suchenden Felder zu veranlassen, ist es erforderlich, diese in der Datei *config/searchfields.xml* zu definieren. In diesem Zusammenhang sollte auch die Datei *config/mycore.properties.private* den neuen Anforderungen (Beispiel siehe *config/mycore.properties.private.template*) angepasst werden.

3.2 Migration des Datenbestandes

Die Übernahme der Daten ist am Einfachsten. Durch die Kommandoerweiterung `export` im CommandLineTool von MyCoRe 1.2 ist es möglich, Daten direkt zu exportieren und dabei via Stylesheet zu transformieren. Die aktuelle Version von MyCoRe 1.2 enthält entsprechende XSLT-Dateien und einen Kommandoaufruf zur Daten-Migration.

Die so transformierten Daten werden in das Verzeichnis *workflow* kopiert und mittels Batch-Job in das neue System geladen.

3.3 Suchmasken

xx

3.4 Ergebnislisten

xx

3.5 Detailansicht eines Treffers

xx

3.6 Eingabemasken

- **Datenpräsentation** – der erheblichste Teil der Umstellung betrifft die Gestaltung der neuen Suchmasken und der Ergebnispräsentation. Im Gegensatz zur Version 1.2 werden jetzt alle Suchmasken auch als Editor-Definitionen abgebildet. Somit ist das alte SearchMask-Konzept obsolete. Die Gestaltung und der Aufruf der neuen Suchmasken entspricht der Programmierung von Editormasken zu 100 %. Da sich als Ergebnis der neuen Suche die Struktur der zurückgelieferten Result-Container vollständig geändert hat, ist es notwendig alle Datenpräsentationen darauf anzupassen.

- **Internationalisierung** – Für alle Textstellen in den Editor-/Suchmasken-Konfigurationen ist es jetzt möglich, eine Internationalisierung via I18N durchzuführen. Dabei sind die <label>-Abschnitte der Definition durch ein i18n-Attribut im übergeordneten Tag zu ersetzen und die Texte im jeweiligen *messages_...properties*-File einzutragen. Die Nutzung der alten <label>-Tags ist aber auch weiterhin möglich[1]. Für den Bereich der Datenpräsentation ist die Umstellung komplexer. Hier müssen alle **_lang.xsl* Dateien in *.xsl umbenannt werden und im Header eine zusätzliche Definition des i18n-Namensraumes erhalten. Die sprachabhängigen Stylesheets **_de.xsl* sind aufzulösen und ebenfalls in das *messages_...properties*-File zu integrieren. Somit sollte die Sprachabhängigkeit der Editor- und Präsentationsdefinitionen nur noch über I18N erfolgen. Dies vereinfacht das Hinzufügen weiterer Sprachen erheblich.

[1] siehe Programmer Guide

4 Erfahrunsberichte

4.1 Migrationstipps von Thomas Scheffler

4.1.1 Ausgangspunkt

- Pflege von eigener MyCoRe- und DocPortal-Version im eigenen CVS
- Anpassung von DocPortal an eigene Bedürfnisse
  - Deployment and Tomcat via Ant, eigene Benutzer/Gruppen
  - Eigenes Datenmodell
  - Eigenes Layout
  - Halbfertige Anwendung auf Basis der jeweiligen 1.2er Komponenten mit Backports aus Version 1.3

Soweit die Ausgangslage. Nachdem ich den CODE bei mir im CVS gemergt hatte, wahren alle Codestände wieder auf 1.3er Niveau und das prinzipielle Starten der Webanwendung klappte, da die Anwendung jetzt wieder eher DocPortal glich als einer Archivanwendung. Es folgen nun die Schritte, wie ich vorgegangen bin, um das System wieder zu einer Archivanwendung umzugestalten.

4.1.2 Migration: Schritt für Schritt

„Dieser Nutzer ist für die Arbeit mit dem Editor nicht berechtigt"

- `config/user/permission.xml` muss ergänzt werden um `create-<datentyp>`
- Aufwand: gering
- Beispiel:

```
<mcrpermission name="create-akte" description="Erzeugen einer
Akte">
  <condition format="xml">
    <boolean operator="or">
      <condition value="admingroup" operator="=" field="group" />
    </boolean>
  </condition>
</mcrpermission>
```

Editoren gehen nicht mehr

- `imports-common.xml` enthält jetzt Abhängigkeiten zum DocPortal-Datenmodell und versucht DocPortal Klassifikationen zu laden. Entsprechende `include`-Anweisungen in eigenen Editoren umstellen auf eigene Datei, z.B. `import-archiv.xml` und dort nötige Sachen einfügen.
- Aufwand: gering bis mittel

Editoren - `Subselect` hat sich geändert

- `Subselects` gehen nicht mehr über das `MCRQueryServlet` (deprecated) sondern über das Index-Servlet. Beispiel-Aufruf sie DocPortal `document`-Editor. Es ist ein Anpassen der `properties` an lokales Datenmodell erforderlich
- Aufwand: gering bis mittel

`searchfields.xml` anpassen

- `searchfields.xml` muss zur Indizierung erweitert werden, so das Daten im eigenen Datenmodell indiziert werden. Als Beispiel kann wieder DocPortal genommen werden. Ich habe folgendes als `allMeta`-Feld genommen, um auch verlinkte Objekte zu Finde (z.B. Dokumente wo Autor "Müller" enthält)

```
<field name="allMeta" type="text" source="objectMetadata"
  xpath="/mycoreobject/metadata/*/*/text() |
  /mycoreobject/metadata/*/*/@xlink:title" value="."/>
```

- Aufwand: mittel (Abhängig vom Datenmodell)

Editoren: `MCRQueryServlet` Aufrufe ersetzen

- Meistens betrifft das Klassifikationen, die man durch URIResolver-Aufrufe ersetzen kann. Beispiele finden sich bei DocPortal (Editor für Dokumente und Suchmasken)
- Aufwand: mittel (Abhängig vom Datenmodell)

Suchmasken neu machen

- Suchmasken basieren jetzt auf dem MyCoRe Editor-Framework und müssen daher neugestaltet werden, um XML-Queries für die neue Suche zu generieren. Beispiele finden sich wieder bei DocPortal, jedoch muss man bei all zu viel Copy & Paste darauf achten, dass die condition-Definitionen korrekt durchnummeriert werden, sonst gibt es Fehlermeldungen von JDOM das Attributenamen wie field[2] nicht zulässig sind.
- Aufwand: mittel (Abhängig vom Datenmodell, aber gut copy-and-pastable)

Layouts anpassen

- Pro Datentyp gibt es jetzt nur noch eine XSLT-Datei, was die Übersicht im stylesheet-Ordner erhöht.
  - Am Beispiel von DocPortal sollte man zu erst anfangen das Template für `mcr:hit` zu designen, damit Treffer richtig angezeigt werden.
  - Für die Metadatensicht kann das alte Template übernommen werden aus der 1.2er Anwendung (Anpassung erforderlich für verteilte Suche (Anzeige von Derivaten etc.): siehe DocPortal). Dann lässt man über das Template ein Suchen&Ersetzen laufen und ersetzt `mcr_result/mycoreobject` durch `NICHTS`, sprich entfernt alle Vorkommen. Das sollte eigentlich reichen.
- Aufwand: mittel (Abhängig vom Datenmodell)

Weitere Anmerkungen

- Bei der Übernahme von Daten ist mir aufgefallen, das nach dem Export alle Klassifikations-IDs in den Objekten grundsätzlich klein geschrieben sind und man daher das `migration`-Stylesheet von DocPortal anpassen muss, um z.B. `docportal_class_*` wieder zu `DocPortal_class_*` zu überführen. Genau erforscht habe ich das aus Zeitgründen noch nicht.
- Eigene Anpassung an der DocPortal `build.xml` habe ich jetzt in eine `archiv-build.xml` überführt, die build.xml importiert. Damit werden alle gleich benannten Targets überschrieben und zukünftige Anpassungen der build.xml treffen lokal nicht so stark zu. Meine archiv-build.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<project name="archiv" default="usage" basedir=".">
  <import file="${basedir}/build.xml" />
  <property name="war.file" value="${ant.project.name}.war" />
  <property file="build.properties" />

  <path id="tomcat.classpath">
    <path refid="docportal.classpath" />
    <fileset dir="${tomcat.home}/common/lib">
    </fileset>
    <fileset dir="${tomcat.home}/server/lib" />
  </path>

  <target name="create.default-rules" depends="init"
          description="Load default ACL rules">
    <property name="commands.file"
              value="${basedir}/commands.txt" />
    <echo file="${commands.file}" append="false">
      update permission read for id default with rulefile
        grant-all.xml
      update permission writedb for id default with rulefile
        grant-editors.xml
      update permission deletedb for id default with rulefile
        grant-admins.xml
    </echo>
    <antcall target="invoke.cli">
      <param name="cli.directory" value="${basedir}/config/acl"
      />
      <param name="cli.command" value="process ${commands.file}"
      />
    </antcall>
    <delete file="${commands.file}" />
  </target>

  <target name="war" depends="webapps">
    <jar jarfile="${war.file}" basedir="${docportal.webapps}" />
  </target>

  <target name="deploy" depends="war">
    <taskdef name="deploy"
             classname="org.apache.catalina.ant.DeployTask"
             classpathref="tomcat.classpath" />
    <list url="${tomcat.url}/manager"
          username="${manager.username}"
          password="${manager.password}" />
    <deploy url="${tomcat.url}/manager"
            username="${manager.username}"
            password="${manager.password}"
            path="/${ant.project.name}" war="${war.file}" />
  </target>

  <target name="undeploy">
    <taskdef name="undeploy"
             classname="org.apache.catalina.ant.UndeployTask"
             classpathref="tomcat.classpath" />
```

```
    <undeploy url="${tomcat.url}/manager"
              username="${manager.username}"
              password="${manager.password}"
              path="/${ant.project.name}" />
  </target>

  <target name="reload">
    <taskdef name="reload"
             classname="org.apache.catalina.ant.ReloadTask"
             classpathref="tomcat.classpath" />
    <reload url="${tomcat.url}/manager"
            username="${manager.username}"
            password="${manager.password}"
            path="/${ant.project.name}" />
  </target>

  <target name="redeploy" depends="war,undeploy,deploy" />

</project>
```