Plus CD!

Im Gespräch ▶ Wayne Beaton, Jeff McAffer, Mik Kersten, Roy Ganor, David Williams u.v.m.

5.09

eclipse MAGAZIN

Deutschland € 9,80
Österreich € 10,80, Schweiz CHF 19,20

S&S

eclipse MAGAZIN

www.eclipse-magazin.de

EXKLUSIV AUF CD:

FSHARE-SERVER-PLUG-IN
File-Sharing mit der Net4j Signalling Platform

EMFText
Komplette Beispiel-DSL aus unserem Tutorial „Sprechen Ihre Modelle Klartext?"

Eclipse Classic 3.5
Galileo-Basispaket: Eclipse-Plattform, Java Development Tools, Plug-in Development Environment u.v.m.

Xtext 0.7
Das neue Framework zur Entwicklung externer textueller DSLs

Eclipse Galileo!

Großer Jahresrückblick: Was die Community im Galileo-Jahr bewegte » 10

Was gibt´s Neues in Galileo JDT? » 27

Quantensprung für Equinox » 30

PDT 2.1: Des Kaisers neue Kleider » 36

Runde sechs für CDT » 32

Modeling-Trends in Eclipse Galileo » 40

▶ **DSLs mit Xtext** » 67
Domänenspezifische Sprachen selbst gemacht

▶ **My Mylyn** » 46
Eigene Mylyn Connectors entwickeln

▶ **Vorschau auf Eclipse 4.0** » 93
User Interface Styling mit CSS

„Wir werden e4 in Helios wiedersehen" » 43
Wayne Beaton über Gegenwart und Zukunft von Eclipse

D 68864

Datenträger enthält Info- und Lehrprogramme gemäß § 14 JuSchG

Einen Mylyn Connector selbstgemacht

My Mylyn

Quellcode auf CD!

>> DOMINIK HIRT

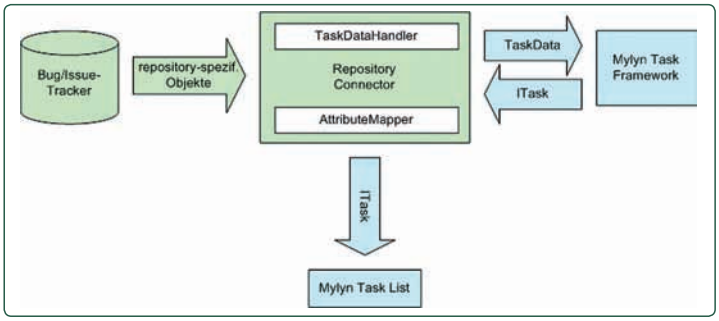
Der überwältigende Erfolg von Mylyn kommt nicht von ungefähr. Die Task-fokussierte Sicht auf den Sourcecode und die perfekte Unterstützung im Eclipse UI sind sicherlich die Hauptgründe dafür. Darüber hinaus liegt es aber auch an der breiten Verfügbarkeit von Connectors für diverse Task Repositories, primär aus dem Open-Source-Bereich. Schade ist es dann, wenn das im eigenen Unternehmen genutzte Task Repository nicht dabei ist. Doch der gewiefte Entwickler weiß sich zu helfen.

Seitdem Mylyn, das damals noch Mylar hieß, 2006 das Licht des Eclipse-Universums erblickte, stieg seine Popularität stetig an. Einer aktuellen Umfrage zufolge ist es mittlerweile auf Rang 6 der populärsten Eclipse-Projekte angelangt [1]. Entsprechend dieser Wertschätzung wird es, mit einer einzigen Ausnahme, jedem der möglichen Eclipse Packages standardmäßig beigelegt [2]. Dieser Erfolg beruht zum einen auf dem innovativen Konzept einer Task-fokussierten Oberfläche für Eclipse, die überflüssige Informationen filtert und den Entwickler auf seine aktuelle Aufgabe lenkt. Diese Aufgaben können lokal definiert werden oder aus einem Remote Task Repository stammen. Das sind in der Regel unternehmensweit genutzte Bug/Issue Tracker, die mittels Connectors an Mylyn angebunden werden. Und hier ist ein weiterer Grund für den Erfolg von Mylyn zu finden: die große Anzahl an verfügbaren Connectors aus dem kommerziellen und freien Softwarebereich. Bugzilla und JIRA werden „von Hause aus", sprich mit dem offiziellen Mylyn-Plug-in, bereits unterstützt. Viele weitere stehen unter [3] zur Verfügung, darunter findet sich auch ein generischer Web-Connector (Kasten: „Web Template Connector").

Aber es gibt immer noch Lücken, und so kam der Autor dieses Artikels, ein begeisterter Mylyn-Nutzer, nicht umhin, sich mit dem Thema Connector-Implementierung selbst zu befassen. Ziel sollte es sein, für das unternehmensweit genutzte kommerzielle Bug-Tracking-System eine Anbindung an die Mylyn Tasklist zu ermöglichen. Darüber hinaus sollte erreicht

werden, die so eingebundenen Bugs/Issues gleich innerhalb Eclipse zu bearbeiten. Wie sich herausstellte, bietet Mylyn dafür zwei Möglichkeiten: online per Browser und offline mit einem Rich Editor, alles innerhalb Eclipse. Damit könnte das ständige Wechseln der Applikationen eingespart werden und wir wären wieder ein bisschen glücklicher.

Abb. 1: Mylyn-Task-list-Block-Schema



The Big Picture

Dieser Artikel gibt eine Einführung zur Erstellung eines Mylyn Connectors. Wir werden als Erstes einen Task Repository Connector in Mylyn registrieren und einen Wizard erzeugen, der gezielt Tasks aus einem entfernten Bug/Issue Tracker laden wird. Dabei ist es notwendig, die Repository-spezifischen Attribute und Eigenschaften so umzuwandeln, dass Mylyn diese richtig interpretieren kann. Ein Großteil der Implementierung befasst sich daher mit der Modellierung und Transformierung von Eigenschaften solcher Task-Objekte. Die dazu genutzten Klassen werden im Kasten „Tasks, Tasks, Tasks" vorgestellt. Wir werden sehen, welche Extension Points erweitert werden müssen, wofür wir einen *TaskDataHandler* und einen *AttributeMapper* benötigen werden. Abbildung 1 zeigt das „Big Picture".

Der Repository Connector

Sollten Sie nicht gerade das Classic Package von Eclipse nutzen, dann haben Sie Mylyn bereits installiert. Ansonsten kann es per Updatemanager von [4] bezogen werden (mind. Version 3.0). Da ein Mylyn Connector später in Form eines normalen Eclipse-Plug-ins deployt wird, starten wir mit dem Anlegen eines neuen Plug-in-Projekts und erweitern einige Extension Points. Wir öffnen den Manifest-Editor der PDE, gehen zum DEPENDENCIES-Reiter und fügen zwei Plug-ins ein, die die benötigten Extensions bereitstellen: *org.eclipse.mylyn.task.ui* und *org.eclipse.mylyn.task.core*. Nun kann man auf dem EXTENSION-Reiter den ersten Extension Point erweitern: *org.eclipse.mylyn.tasks.ui.repositories*. Abb. 2 zeigt das Ergebnis.

Mit dem ersten Extension-Element, *ConnectorCore*, definieren wir einen *RepositoryConnector*. Ein Klick auf das verlinkte *class**-Label neben dem Textfeld öffnet den New Java Class Wizard, der alles Weitere für uns erledigen wird. Wir geben lediglich Package- und Klassennamen an. Es entsteht eine Klasse, die von *org.eclipse.mylyn.tasks.core.AbstractRepositoryConnector* abgeleitet ist. Der *RepositoryConnector* ist die Zentrale unseres Mylyn Connectors. Hier definieren wir einen symbolischen Namen für unser Task Repository, der ab sofort zur Identifikation genutzt wird (*CONNECTOR_KIND*). Diese Klasse enthält auch die Methoden, die Mylyn aufruft, um die gewählte Query auszuführen und damit die Tasks aus dem Repository zu laden. Für den Anfang genügt es, einige Methoden mit Leben zu füllen (*MyRepositoryConnector.java*):

```
public String getConnectorKind()
{
  return MyRepositoryConnector.CONNECTOR_KIND;
}

public String getLabel()
{
  return "My Connector";
}
```

Der Wert aus *getLabel()* wird in der Auswahlliste beim Erzeugen eines neuen Task Repositories angezeigt.

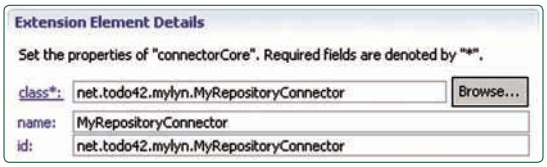
Das Repository UI

Nun kann das zweite Extension-Element *ConnectorUi* definiert werden. Abb. 3 zeigt, wie der Extension Point aussehen sollte. Wir gehen analog zu dem ersten Element vor und lassen uns vom Wizard eine Klasse erzeugen, die nun von *org.eclipse.mylyn.tasks.ui.AbstractRepositoryConnectorUi* abgeleitet ist. Hier implementieren wir erst einmal nur die Methode *getConnectorKind()*. Danach wird es Zeit für einen ersten Test. Im MANIFEST-Editor der PDE können Sie auf dem Reiter *OVERVIEW* Ihr Plug-in per *Launch an Eclipse Application* starten. Öffnen Sie unter WINDOW | SHOW VIEW | MYLYN die TASK REPOSITORIES VIEW. Hier können Sie dann per RECHTE MAUSTASTE | ADD TASK REPOSITORY ein Fenster öffnen, das unseren *My Connector* enthalten sollte. Leider bringt uns ein Klick auf den *NEXT*-Button nicht weiter, wir müssen die nächste Methode in der *ConnectorUi*-Klasse implementieren.

Die Repository Settings

Will man ein neues Task Repository für Mylyn anlegen, wird man einige Attribute wie Host/Port des entsprechenden Servers, gegebenenfalls Zugangsdaten etc. benötigen. Diese müssen auf einer *RepositorySettingsPage* eingegeben werden, die wir nun erstellen. Wir implementieren dazu die Methode *MyRepositoryConnectorUi.getSettingsPage(TaskRepository taskRepository)* und erzeugen eine neue *RepositorySettingsPage* (Listing 1). Diese kann man gleich von *org.eclipse.mylyn.tasks.ui.wizards.AbstractRepositorySettings-*

Abb. 2: ExtensionPoint für ConnectorCore



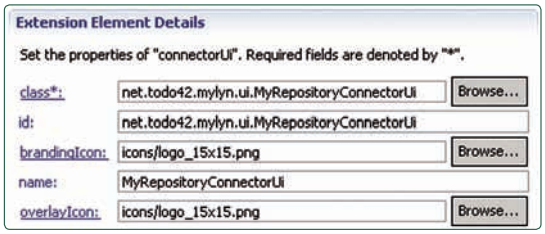


Abb. 3: ExtensionPoint für ConnectorUi

Page ableiten und nicht, wie der Wizard vorschlagen wird, von *ITaskRepositoryPage*. Die existierende, abstrakte Implementierung bringt nämlich schon ein ansehnliches UI zur Eingabe von Attributen wie Server URL, Repository-Bezeichner, Zugangsdaten etc. mit. Wir müssen nur die Methode *isValidUrl(String)* implementieren, um diese Wizard-Seite auch beenden zu können. Listing 2 zeigt eine ausreichende SettingsPage.

Mithilfe dieser *SettingsPage* können wir nun ein Task Repository anlegen. Mylyn fragt uns nach dem erfolgreichen Anlegen auch gleich, ob wir eine Query definieren wollen. Damit kommen wir zum nächsten Schritt, dem Verbinden zum entfernten Bug/Issue Tracker.

Der Bug/Issue Tracker, Teil 1

Der Einfachheit halber werden wir für die genutzte Abstraktion eines Bug/Issue Trackers annehmen, dass dieser über eine Menge von Queries verfügt, die bereits angelegt wurden und jeweils eine bestimmte Teilmenge aller Bugs/Issues ermitteln können. Somit entfällt die Notwendigkeit, selbst ein reichhaltiges UI zu implementieren, um solche Queries zu erzeugen. Stattdessen genügt es, die Menge aller Queries mit ID und Namen vom Bug/Issue Tracker zu laden und den Anwender dann genau eine Query zur Benutzung auswählen zu lassen. Im Gegensatz zu diesem leichtgewichtigen Ansatz bringen einige Connectors ein reichhaltiges UI für diese Funktionalität auf den so genanten Query Pages mit.

Die Funktionalität des jeweiligen Bug/Issue Trackers sollten Sie zuerst in einem Interface beschreiben und dann – vielleicht in einem Singleton – implementieren. Mithilfe des Interface kann man später für (Unit-)Tests Mock-Objekte erstellen oder zwischen alternativ implementierten Zugangsmethoden wählen (z. B. Command Line API, HTTP-Requests etc.). Ein solches Interface enthält typischerweise Methoden, wie sie hier zu sehen sind (*IRepository.java*):

```
public interface IRepository
{
  public boolean connect();

  public List<Query> getQueries();

  public List<Issue> getIssues(String queryID);

  public Issue getIssueDetails(String taskID);

  public boolean editIssue(Issue issue);

  public String createIssue(Issue issue);
}
```

Basierend auf der oben gemachten Annahme, dass diese Queries bereits existieren, müssen sie nun abgerufen und auf einer Wizard-Seite in einer *SelectBox* zur Auswahl angezeigt werden.

Die Query Page

Eine Query Page wird in der Methode *MyRepositoryConnectorUi.getQueryWizard* erzeugt (Listing 1). Da Mylyn wieder eine abstrakte Implementierung einer solchen Page mitbringt, leiten wir

Listing 1

MyRepositoryConnectorUi.java

```
public class MyRepositoryConnectorUi extends
                    AbstractRepositoryConnectorUi
{
  public MyRepositoryConnectorUi()
  {
  }

  public String getConnectorKind()
  {
    return MyRepositoryConnector.CONNECTOR_KIND;
  }

  public ITaskRepositoryPage getSettingsPage
                    (TaskRepository taskRepository)
  {
    return new RepositorySettingsPage(taskReposi
                                          tory);
  }

  public IWizard getQueryWizard(TaskRepository
        taskRepository, IRepositoryQuery queryToEdit)
  {
    RepositoryQueryWizard wizard = new RepositoryQu
                        eryWizard(taskRepository);
    QueryPage queryPage = new QueryPage
                    (taskRepository, queryToEdit);
    wizard.addPage(queryPage);
    return wizard;
  }
}
```

Listing 2

RepositorySettingsPage.java

```
public class RepositorySettingsPage extends
                    AbstractRepositorySettingsPage
{
  private static final String TITLE =
                    "My Repository Settings";
  private static final String DESC = "Please insert your
                                        values";

  public RepositorySettingsPage(TaskRepository
                                    taskRepository)
  {
    super(TITLE, DESC, taskRepository);
  }

  public String getConnectorKind()
  {
    return MyRepositoryConnector.CONNECTOR_KIND;
  }

  protected boolean isValidUrl(String name)
  {
    if ((name.startsWith(URL_PREFIX_HTTPS)
        || name.startsWith(URL_PREFIX_HTTP)) &&
                                !name.endsWith("/"))
    {
      try
      {
        new URL(name);
        return true;
      }
      catch (MalformedURLException e)
      {
      }
    }
    return false;
  }
}
```

Listing 3

MyRepositoryConnector.performQuery

```
public IStatus performQuery(TaskRepository
    taskRepository, IRepositoryQuery query, TaskData-
    Collector resultCollector, ISynchronizationSession
                    event, IProgressMonitor monitor)
{
  IStatus result = Status.OK_STATUS;
  monitor.beginTask("Querying repository",
                    IProgressMonitor.UNKNOWN);
  IRepository repository = MyRepository.
                    getInstance(taskRepository);
  if (!repository.connect())
  {
    result = new RepositoryStatus(taskRepository.
      getRepositoryUrl(), IStatus.ERROR, "net.todo42.
                                mylynconnector",
        RepositoryStatus.ERROR_REPOSITORY,
                    "Can't connect to repository");
  }
  else
  {
    List<Issue> issues = repository.getIssues(query.
                                getAttribute("id"));
    for (Issue issue : issues)
    {
      TaskData data = taskDataHandler.createPartialTask
                    Data(taskRepository, monitor, issue);
    data.setPartial(true);
      resultCollector.accept(data);
    }
    result = Status.OK_STATUS;
  }
  monitor.done();
  return result;
}
```

unsere Query Page hiervon ab: *org.eclipse.mylyn.tasks.ui.wizards.AbstractRepositoryQueryPage*.

Die vollständige *QueryPage.java*, alle anderen Source-Dateien und auch das komplette Eclipse-Projekt finden Sie auf der Heft-CD und unter [5]. In Mylyn repräsentiert ein Objekt vom Typ *org.eclipse.mylyn.tasks.core.IRepositoryQuery* eine Query. Ein solches Objekt verfügt über eine Map von Attributen. Wir speichern die ID der gewählten Query in dieser Map ab und lassen Mylyn das Query-Objekt verwalten. Später, wenn es darum geht, mithilfe einer solchen Query die existierenden Tasks zu erfragen, werden wir dieses Objekt wieder von Mylyn bekommen und können dann auf die darin gespeicherten Informationen zugreifen.

Wir haben nun die notwendigen Vorarbeiten abgeschlossen. Mit dem Beenden der Auswahlseite einer Query versucht Mylyn, diese auszuführen. Dazu wird die Methode *MyRepositoryConnector.performQuery* (Listing 3) aufgerufen. Diese Methode führt die angegebene Query aus und lädt so Tasks aus dem entfernten Bug/Issue Tracker. Hierbei sollten nur die wichtigsten Attribute dieser Tasks geladen werden. Diese dienen dazu, den Task in einer übersichtlichen Form in der Mylyn-Tasklist darzustellen. Zu diesen Attributen gehören eine eindeutige Kennung des Tasks, eine Zusammenfassung des Inhalts (Summary) sowie die Priorität, die in der Mylyn-Tasklist durch unterschiedliche Symbole visualisiert werden kann. Wenn Ihr Repository auch ein Fälligkeitsdatum eines Bug/Issue kennt, können Sie dieses ebenfalls schon hier laden. Mithilfe eines solchen Attributs kann der Task in der Tasklist farblich hervorgehoben werden, um damit seine Dringlichkeit anzuzeigen. Die Beschränkung auf einige wichtige Daten wirkt sich natürlich auch positiv auf die Performance aus und stellt das reibungslose Darstellen der Tasks in der Tasklist sicher. Als Hinweis für Mylyn, dass die *TaskData*-Objekte nach dem ersten Durchlauf erst partiell mit Attributen gefüllt sind und einer weiteren Synchronisation mit dem Repository bedürfen, setzen wir *taskData.setPartial(true)*. Das veranlasst Mylyn, die vollständigen Daten durch einen Synchronisationsjob und dem Aufruf *MyRepositoryConnector.getTaskData* in einem zweiten Schritt zu laden. Die erzeugten *TaskData*-Objekte werden noch der Resultatsmenge hinzugefügt, und wir sind bereits fertig. Listing 3 zeigt die Methodenimplementierung.

Listing 4

AttributeMapper.java

```
public class AttributeMapper extends
                                TaskAttributeMapper
{
 public String mapToRepositoryKey(TaskAttribute
                                parent, String key)
 {
  String result = null;

  if (key.equals(TaskAttribute.SUMMARY))
  {
    result = RepositoryAttribute.SUMMARY.getKey();
  }
  else if (key.equals(TaskAttribute.PRIORITY))
  {
    result = RepositoryAttribute.PRIORITY.getKey();
  }
  else if (key.equals(TaskAttribute.DATE_CREATION))
  {
    result = RepositoryAttribute.CREATION_DATE.
                                        getKey();
  }
  if (result != null)
  {
    return result;
  }
  else
  {
    return super.mapToRepositoryKey(parent, key).
                                        toString();
  }
 }
```

Listing 5

MyRepositoryConnector.getTaskData

```
public TaskData getTaskData(TaskRepository
        taskRepository, String taskId, IProgressMonitor
                        monitor) throws CoreException
{
  TaskData result = null;
  monitor.beginTask("Querying repository for issue "
                + taskId, IProgressMonitor.UNKNOWN);
  IRepository repository = MyRepository.
                        getInstance(taskRepository);
  if (!repository.connect())
  {
    result = null;
  }
  else
  {
    Issue issue = repository.getIssueDetails(taskId);
    result = taskDataHandler.createFullTaskData(taskR
                        epository, monitor, issue);
  }
  monitor.done();
  return result;
}
```

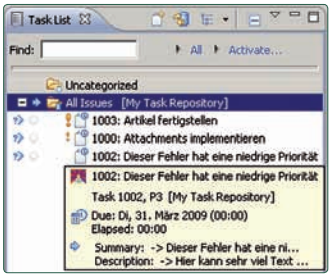


Abb. 4: Mylyn Tasklist des Beispielprojekts

Die verwendete Query ist die zuvor in der *QueryPage* selektierte, deren ID als Attribute aus dem Mylyn-Query-Objekt bezogen und als Parameter der Methode übergeben wird. Die Implementierung der Methode *IRepository.getIssues(String id)* hängt nun sehr stark vom verwendeten Bug/Issue Tracker ab.

Der Bug/Issue Tracker, Teil 2

Um dieses Tutorial nicht unnötig zu verkomplizieren, verwenden wir eine sehr einfache Abstraktion für unseren Bug/Issue Tracker. Die Methode *getQueries()* liefert genau eine Query zurück, und *getIssues(String queryId)* liefert immer die gleiche Menge an Issues zurück. Diese Einschränkung zielt nur darauf ab, nicht auf Details bestimmter Bug/Issue Tracker eingehen zu müssen. Für die verwendeten Mechanismen des Mylyn Connectors selbst stellen sie keine Einschränkung dar.

Fassen wir kurz das bisherige Geschehen zusammen: Als Erstes haben wir einen RepositoryConnector registriert und konnten somit ein neues Task Repository für Mylyn anlegen. Anschließend wurde über eine Wizard-Seite eine Query ausgewählt, die vom Repository-Server Tasks/Issues geladen hat. Jedes geladene Issue

Listing 6

EditorPageFactory.java

```
public boolean canCreatePageFor(TaskEditorInpu
                                        t input)
{
  if (input.getTask().getConnectorKind().equals
        (MyRepositoryConnector.CONNECTOR_KIND))
  {
    return true;
  }
  else
  {
    return false;
  }
}

public FormPage createPage(TaskEditor parentEditor)
{
  return new EditorPage(parentEditor);
}
```

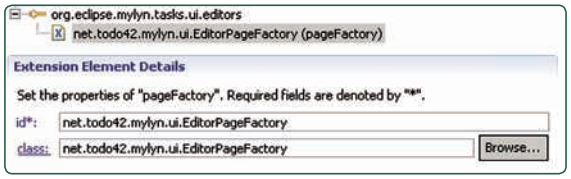


Abb. 5: ExtensionPoint für die EditorPageFactory

mit seinen Repository-spezifischen Attributen muss nun transformiert werden.

Der TaskDataHandler

Das Erzeugen von Mylyn-konformen *TaskData*-Objekten erledigt ein TaskDataHandler vom Typ *org.eclipse.mylyn.tasks.core.data.AbstractTaskDataHandler*. Diesen erzeugen wir im RepositoryConnector und stellen ihn mit der Methode *getTaskDataHandler()* zur Verfügung. Unser TaskDataHandler verfügt über zwei öffentliche Methoden, *createPartialTaskData* und *createFullTaskData*. Innerhalb dieser Methoden wird die Hierarchie von *TaskAttribute*-Objekten erzeugt, die Mylyn Tasks kennzeichnen. Solche Attribute verfügen über eine Reihe von Metainformationen wie Typ, Label, readOnly etc., die primär für das generische Erzeugen eines Editor-UI gedacht sind. Die Attribute selbst sind naturgemäß sehr stark vom verwendeten Bug/Issue Tracker abhängig. Daher ist es auch notwendig, mithilfe eines *AttributeMapper* zwischen diesen beiden Welten zu vermitteln. Der TaskDataHandler selbst setzt abschließend noch die tatsächlichen Werte der einzelnen Attribute basierend auf den Werten aus den Repository-spezifischen Attributen unseres *Issue*. Wenn Sie Ihrem Connector noch den Support des Rich Editor spendieren, dann muss Ihr TaskDataHandler zusätzlich noch Methoden wie *initializeTaskData* und *postTaskData* implementieren. Diesen werden die Daten aus dem Editor in Form eines *TaskData*-Objekts übergeben. Sie wandeln diese in ein Repository-spezifisches Objekt (hier *Issue*) um und rufen dann damit ihre Bug/Issue-Tracker-Interface-Methoden *createIssue* bzw. *editIssue* auf. Wir gehen später noch darauf ein.

Der AttributeMapper

Das Mapping von Repository-spezifischen Attributen in Mylyn-Attribute implementieren wir in einer von *org.eclipse.mylyn.tasks.core.data.TaskAttributeMapper* abgeleiteten Klasse. Zuerst können wir hier ein *enum* mit den Repository-spezifischen Attributen deklarieren. Dabei werden Eigenschaften vergeben, die sich später auf die Erzeugung des entsprechenden TaskAttributs auswirken und sich in den *TaskAttributeMetaData* wiederfinden (*TaskDataHandler.createAttribute*). Die Methode *mapToRepositoryKey* (Listing 4) wird immer dann aufgerufen, wenn für ein Mylyn-spezifisches Attribut das korrespondierende Repository-Attribut ermittelt werden soll.

Nachdem wir nun das Mapping zwischen Repostitory-spezifischen und Mylyn-Task-Attributen definiert haben, können wir zum letzten Schritt kommen, dem Laden aller weiteren Bug/Issue-Details und dem Aufbereiten in *TaskData*-Objekte. Hierzu ruft Mylyn die Methode *MyRepositoryConnector.getTaskData* auf (Listing 5). Die Implementierung ähnelt der von *performQuery*. Auch hier werden zuerst die Daten aus dem Remote Bug/Issue Tracker geladen, nun jedoch nicht mehr nur einige wenige Attribute, sondern alle, die wir darstellen möchten. Das Repository-spezifische Objekt *Issue* wird mithilfe des TaskDataHandler in ein vollständig gefülltes *TaskData*-Objekt umgewandelt. Nach einem Doppelklick auf einen selektierten Task in der *Mylyn Task List* öffnet sich der *Mylyn Task Editor*. Auf diesem sollte nun der Reiter *Planning* und nach der Aktivierung eines Tasks auch der Reiter *Context* zur Verfügung stehen, auf dem Sie weitere Eigenschaften des Tasks einstellen können (z. B. Schedule Date).

An dieser Stelle sei noch auf die beiden Methoden *MyRepositoryConnector.hasTaskChanged* und *MyRepositoryConnector.updateTaskFromTaskData* hingewiesen. Diese sind notwendig, um Änderungen festzustellen und die beiden genutzten Task-Modelle ineinander überführen zu können. Fehlt eine dieser Methoden, bleibt die Tasklist trotz fehlerfrei geladener Tasks merkwürdig leer. Ansonsten sollten Sie nun eine Tasklist erhalten, wie sie Abbildung 4 zeigt.

Tasks, Tasks, Tasks

In Mylyn werden Tasks auf zwei verschiedene Arten modelliert: mit den schwergewichtigen *TaskData* und den leichtgewichtigen *ITask*. Erstere enthalten zur Laufzeit die Repositoy-spezifischen Attribute, die in einem TaskDataHandler (Listing 4) erzeugt werden. Dabei wird eine Hierarchie von *TaskAttribute-Objekten* erstellt, die die spezifischen Eigenschaften eines Bugs/Issue Ihres Task Repositories widerspiegeln. Diese *TaskAttribute* wiederum enthalten bestimmte Metadaten vom Typ *TaskAttributeMetaData*, die genutzt werden, um bei Verwendung eines Rich Editors zum Bearbeiten eines solchen Tasks auf generische Art und Weise die passenden Editor UI Controls zu erzeugen. Dazu werden gewisse Typinformationen beigefügt, die in *org.eclipse.mylyn.tasks.core.data.TaskAttribute* definiert sind. Die wichtigsten Vertreter sind sicherlich TYPE_SHORT_TEXT, TYPE_LONG_RICH_TEXT und TYPE_SINGLE_SELECT für ein einfaches, kurzes Textfeld, eine Text-Area für formatierbaren Text und eine *SelectBox*. Mylyn selbst verwendet für die Tasklist-Objekte vom Typ *org.eclipse.mylyn.tasks.core.ITask*. Diese verwenden konkrete Mylyn-spezifische Attribute wie Summary, Creation und Due Date oder Priority. Mithilfe eines Mappers vom Typ *org.eclipse.mylyn.tasks.core.data.TaskAttributeMapper* müssen Sie ein Mapping zwischen Task-Repository-spezifischen und Mylyn-Attributen angeben (Listing 4). Neben diesen beiden vorgegebenen Typen steht es Ihnen natürlich frei, Ihre Repository Tasks in einer eigenen Klassenhierarchie zu modellieren. Im vorliegenden Beispiel wurde dafür die Klasse *net.todo42.model.Issue* genutzt. Diese dient als Datencontainer für (bespielhafte) Attribute unseres Task Repositories.

Web Template Connector

Von der Mylyn Incubator Update Site [6] kann ein Generic Web Template Connector bezogen werden. Dieser ist prinzipiell für alle Bug/Issue Tracker geeignet, die sich über ein Web-Frontend bedienen lassen. Eine Anleitung zum Einrichten gibt es unter [7]. Dieser Connector ist in der Lage, alle notwendigen Informationen durch das Parsen der HTML-Seiten mithilfe von Regular Expressions zu erhalten. Er kann nach bestimmten Kriterien konfiguriert werden, und auch das Thema Anmeldung über *user/password* wurde berücksichtigt. In der Praxis jedoch zeigen sich schnell Probleme, wenn die zu parsenden HTML-Seiten nicht valides XML darstellen. Diese können dann nicht verarbeitet werden und eine Benutzung scheitert. Dass dieser Connector auch nach einiger Zeit noch nicht sein Incubator-Status verlassen hat, macht deutlich, dass hier die Probleme wohl im Detail stecken. Eine einfache Grails-Webanwendung, die vom Autor zum Erstellen und Verwalten von Aufgaben einmal erstellt wurde, lies sich jedoch sofort per Web Template Connector einbinden.

Der Rich Editor

Zum Abschluss möchten wir nun noch unsere Tasks direkt in Eclipse bearbeiten. Wir müssen dazu die folgenden Required-Plug-ins aufnehmen: *org.eclipse.ui.forms* und *org.eclipse.mylyn.commons.ui*. Danach erweitern wir den Extension Point *org.eclipse.mylyn.tasks.ui.editors* und definieren eine Page Factory. Abbildung 5 zeigt das Ergebnis.

Die Klasse lassen wir uns wieder vom Wizard erzeugen. Die wichtigsten zu implementierenden Methoden sind in Listing 6 zu sehen. Sie definieren die Zuständigkeit genau für unseren Connector und erzeugen die EditorPage. Die EditorPage selbst bleibt einfach und kann ohne zusätzliche Controls auskommen. Alle anzuzeigenden Details steuern wir über die entsprechenden Task-Attribute. Um nun ein Repository-spezifisches Attribut im Rich Editor bearbeiten zu können, sind folgende Schritte notwendig:

- Definieren Sie Ihr Repository-spezifisches Attribut in *AttributeMapper.RepositoryAttribute*
- Definieren Sie ein Mapping zum Mylyn-Attribut in *AttributeMapper.mapToRepositoryKey*
- Erzeugen Sie das *TaskAttribut* im *TaskDataHandler.createFullTaskData*

Im ersten Schritt legen Sie Eigenschaften für das UI Control fest, das für Ihr Attribut generiert wird. Hierzu zählen ein Label vor dem Control, der Typ (kurzes Textfeld, SelectBox, etc.), und die Angabe, ob es *readOnly* sein soll. Einige Attribute müssen als *HIDDEN* markiert werden. Das bezieht sich auf die Darstellung im Attribut-Part der EditorPage. Hierunter fallen solche, die standardmäßig auf anderen Parts (z. B. Description) angezeigt werden sollen. Spielen Sie ruhig ein wenig mit den Einstellungen für die *enum*-Werte in *AttributeMapper.RepositoryAttribute* und betrachten die Resultate.

Ihr Editor verfügt über einen Submit-Button zum sofortigen Absenden der Änderungen, der zu einem Aufruf von *TaskDataHandler.postTaskData* führt. Unabhängig davon stellt Mylyn auch einen Offline-Modus zur Verfügung: Mit Ctrl + S lassen sich die Änderungen lokal abspeichern. Unterschiede zum Stand im entfernten Bug/Issue Tracker werden erkannt, in der Tasklist und im Editor visualisiert und bei einer erneuten Synchronisation auch nicht gleich überschrieben.

Neben dem Rich Editor mit seiner Offline-Fähigkeit stellt Mylyn noch eine klassische Onlinebearbeitung zur Verfügung. Diese Möglichkeit kann natürlich nur genutzt werden, wenn Ihr genutzter Bug/Issue Tracker eine Bearbeitung der Bugs/Issue in einer Weboberfläche mitbringt. Es muss dann für jeden Task eine eindeutige URL geben, die sich aus einem Teil der Repository-URL und einem Task-spezifischen Teil zusammensetzt, z. B. *http://issuetracker/edit=<ID>*. Wenn Sie einen solchen URL als *TaskAttribute.TASK_URL* in jedem *TaskData*-Objekt anlegen, kann man in der Tasklist mit rechte Maustaste | Open with Browser den Task in der für Ihren Bug/Issue Tracker typischen Weboberfläche im Eclipse-Browser öffnen.

Fazit

Dieser Artikel hat einen Weg aufgezeigt, wie durch Implementieren eines Mylyn Connector die Einträge eines Bug/Issue Trackers in die Mylyn Tasklist integriert werden können. Dies kann erstaunlich einfach realisiert werden und ist sicher auch ein Grund für die große Zahl an verfügbaren Connectoren. Wenn man sich mit dem Mapping zwischen Task-Attributen in der Mylyn- und Nicht-Mylyn-Welt einmal vertraut gemacht hat, ist der Rest (fast) normale Eclipse-Plug-in-Technik.

Dominik Hirt ist Diplom-Informatiker und bereits seit 1995 Java-Entwickler. Er arbeitet als Software Engineer bei der Wincor Nixdorf AG. Darüber hinaus gilt sein Interesse praktischen Verbesserungen am Softwareentwicklungsprozess und Model-driven Software Development. Er ist erreichbar unter dominik.hirt@todo42.com.

>> Links & Literatur

[1] http://ianskerrett.wordpress.com/2009/04/06/eclipsecon-2009-demographics/

[2] http://www.eclipse.org/downloads/packages/compare-packages

[3] http://wiki.eclipse.org/index.php/Mylyn/Extensions#Task_Repository_Connectors

[4] http://www.eclipse.org/mylyn/downloads/

[5] http://www.todo42.net/mylynConnector

[6] http://download.eclipse.org/tools/mylyn/update/incubator

[7] http://wiki.eclipse.org/Mylyn_User_Guide#Generic_Web_Templates_Connector

Mik, was gibt´s Neues bei Mylyn?

Mik Kersten: Seit Ganymede haben wir 988 Erweiterungen implementiert und Fehler behoben, von denen 249 als Patches von unserer sehr aktiven Community eingereicht wurden. Es gibt 23 beachtenswerte Neuerungen, unter denen wohl das Mylyn-Connector-Discovery-Portal und WikiText das größte Echo ausgelöst haben. WikiText erweitert Mylyns Task-Editor um leichtgewichtige Funktionen zur Bearbeitung von Wiki Markup. Das bedeutet, dass Benutzer sowohl von den Vorteilen von WYSIWYG als auch von der Bearbeitung von unformatiertem Text profitieren. Wie auch andere Mylyn-Komponenten, kann WikiText als eigenständige Programmierschnittstelle zur Konvertierung von Media Wiki oder Confluence-Dokumenten in das Eclipse-Hilfeformat oder in das PDF-Format verwendet werden. Das Connector-Discovery-Portal ermöglicht mit wenigen Klicks die Installation von Erweiterungen zur Integration von Task-Managementsystemen. Über drei Dutzend Mylyn-Konnektoren für die am häufigsten eingesetzten Werkzeuge zur Änderungs- und Projektverfolgung sind heute verfügbar, und über das Connector-Discovery-Portal sind Erweiterungen aus dem Eclipse Ecosystem einfach zugänglich. Und schließlich sind noch die Verbesserungen der Benutzerschnittstelle zu erwähnen, die in den Task-Editor eingeflossen sind. Editoren werden schneller geöffnet, sind einfacher zu bedienen und sehen besser aus. Das Beste ist, dass zusätzlich zu der bestehenden Funktionalität wie Offlinebearbeitung von Tasks, Änderungsbenachrichtigung und jetzt WikiText, alle realisierten Überarbeitungen automatisch mit den bestehenden Konnektoren funktionieren, sobald die Mylyn-Version auf Galileo aktualisiert wird.

Zu den Neuerungen bei Mylyn lesen Sie auch das Interview mit Steffen Pingel unter: www.jaxenter.de/pingel.

Mik Kersten ist CEO von Tasktop Technologies, Project Lead von Mylyn und Mitglied des Eclipse Architecture Council and Board of Directors. Er ist Erfinder des Task-orientierten Interfaces und entwickelte Mylyn im Rahmen seines PhD an der University of British Colombia. Sein Blog: http://tasktop.com/blog.