Jenkins

Currently in german only.

Nicht mehr alles aktuell.

Was ist zu tun, wenn Build fehlgeschlagen?

Es gibt mehrere Möglichkeiten, warum ein Build fehlschlägt. Hier eine kleine Auswahl an bekannten Problemen:

- Odysseus ist an sich nicht kompilierbar - wurde also schon fehlerhaft eingecheckt. Lösung: reparieren und neu einchecken.
- Vor allem beim Löschen und/oder Verschieben von Bundles, kann es sein, dass ein Fehler a la "konnte .project nicht finden" auftaucht. Hier reicht es i.A. den Arbeitsbereich zu löschen. Dazu in Jenkins den Build-Schritt "Odysseus Hourly Build" aufrufen und im Menü auf "Arbeitsbereich" (in Englisch Workspace) klicken und dann sollte dort "Arbeitsbereich löschen" stehen. Dadurch wird dann der Arbeitsbereich gelöscht und alles neu ausgecheckt, wenn man dann anschließend auf "Jetzt Bauen" klickt. Man muss natürlich eingeloggt sein.
- Wenn ein Feature umbenannt oder gelöscht wurde, kann es sein, dass nicht alle Komponenten gefunden werden. Dazu am besten in der feature.xml vom Bundle de.uniol.inf.is.odysseus.updatesite schauen, ob alle Features, die dort angegeben sind, auch existieren. Diese feature.xml gibt nämlich an, welche Features beim Bauen berücksichtigt werden sollen. Wurde also ein Feature gelöscht, dann muss es hier auch entfernt werden.
- Die ID einer Product-Definition (unter "Overview" in der *.product Datei) muss einzigartig sein. Wer also z.B. das "Odysseus Studio 2 (MemoryStore).product" kopiert, der muss auch die ID ändern!
- "No suitable provider for component" --> Nachschauen ob die ID und der Name des Bundles identisch sind

Der Build-Prozess

Der in Jenkins definierte Build-Prozess besteht aus folgenden Teil-Schritten:

- **Fetch and Build Target:**
  Lädt und importiert die Target Platform (Eclipse / Equinox, GEF etc. - also alles was im Prinzip externe Features sind, auf die Odysseus aufbaut), die in der Datei "target-platform.target" definiert ist, die sich im Projekt updatesite befindet. Dabei lädt Jenkins die entsprechenden Bundles aus den Update-Sites herunter, die in der Target-Platform definiert sind! Darunter befinden sich auch alle notwendigen Sachen für verschiedene Plattformen (Win, Mac, Linux) aka "delta-pack".

- **Odysseus Nightly Build**
  Wird jeden Tag um 0 Uhr ausgeführt und startet eine Build-Pipeline zum Bauen aller Produkte in bestimmten Konfigurationen, indem es die folgenden vier Build-Jobs nacheinander ausführt.

- **Build Odysseus Server**
  Erzeugt ein ausführbares Produkt aus der Produktdefinition, die als Datei "Odysseus Product Platform Server.product" im Projekt de.uniol.inf.is.odysseus.product.server.starter liegt und installiert zusätzlich das Feature de.uniol.inf.is.odysseus.product.server.feature. Siehe unten zu Details zur Kombination aus Platform und installiertem Feature.

- **Build Odysseus Studio**
  Erzeugt ein ausführbares Produkt aus der Produktdefinition, die als Datei "Odysseus Product Platform Studio.product" im Projekt de.uniol.inf.is.odysseus.product.studio.starter liegt und installiert zusätzlich das Feature de.uniol.inf.is.odysseus.studio.feature. Siehe unten zu Details zur Kombination aus Platform und installiertem Feature.

- **Build Odysseus Server + Studio**
  Erzeugt ein ausführbares Produkt aus der Produktdefinition, die als Datei "Odysseus Product Platform Studio.product" im Projekt de.uniol.inf.is.odysseus.product.studio.starter liegt und installiert zusätzlich das Feature de.uniol.inf.is.odysseus.product.serverandstudio.feature. Siehe unten zu Details zur Kombination aus Platform und installiertem Feature.

- **Build Fresh Update Site**
  Erzeugt eine Update-Site durch die ein bereits installiertes Odysseus um weitere Features erweitert werden und aktualisiert werden kann, indem die Update-Site http://odysseus.informatik.uni-oldenburg.de/update/ unter "Help > Install New Software" innerhalb von Odysseus verwendet wird. Dabei enthält die Update-Site alle Features, die in der feature.xml im Projekt de.uniol.inf.is.odysseus.updatesite definiert sind.

Jede Nacht um 0 Uhr beginnt der Build-Prozess, indem der Build "Odysseus Nightly Build" (und damit die Build-Pipeline) gestartet wird. Da das "fetchen" bzw. importieren der Target Platform recht lange dauert und sich diese nur selten ändert, wird der Teil-Build "Fetch and Build Target" nicht täglich ausgeführt, sondern muss beim Ändern der Target-Platform per Hand geändert werden.

Zusätzlich erfolgt noch jede Stunde ein Test-Build:

- **Odysseus Hourly Build**
  Dieser Job checkt nur jede Stunde, ob Odysseus gebaut werden kann oder ob es Probleme gibt. Dabei werden im Prinzip alle Features und Bundles wie in Eclipse mit "Building Workspace" compiliert und daraus eine Update Site erzeugt. Diese Update Site ist unter http://odysseus.informatik.uni-oldenburg.de/hourlyupdate/ zu finden und kann z.B. für kurzfristige Updates genutzt werden.

Aufbau des Produkts: Platform + Hauptfeature

Problem ist, dass man das Kern-Produkt einer RCP-Anwendung nicht einfach durch Updates aktualisieren kann, da die Abhängigkeiten nicht aufgelöst werden können. Dies liegt u.a. daran, dass die Anwendung selbst noch läuft wenn man auf "check for updates" geht und es nicht möglich ist, eine Version zu aktualisieren, während die andere als Teil der RCP-Anwendung noch läuft.

Aus diesem Grund wird Odysseus so gebaut, dass es eine Basis-Platform als Hauptanwendung gibt, die nicht aktualisierbar ist und der Rest - als das komplette eigentliche Odysseus - als ein zusätzlichen Feature (Hauptfeature) hinzu installiert wird. Da das Hauptfeature hinzu installiert wird, ist es aktualisierbar und damit ist dann auch ganz Odysseus über die Updatesite aktualisierbar.

Also gibt es zwei Teile: eine fast leere Platform und ein Hauptfeature, dass Odysseus beinhaltet und je nach Produkt (Server, Studio, Server+Studio) andere Kombinationen von anderen Features enthält. Diese beiden Teile werden in der *buckminster_product.properties* definiert.

Wie erwähnt, enthält die Platform nur eine Basis zum Ausführen von Odyssues. Eine Product-Definition legt dabei die Platform fest. Dazu wird ein Platform-Feature verwendet, damit dieses Platform-Feature auch in anderen Product-Definitionen verwendet werden kann. Ein Platform-Feature beinhaltet letztlich nur die Platform (Equinox, ggf. RCP etc.) und zusätzlich ein starter-bundle. Dieses starter-Bundle beinhaltet die OSGi-Application, die Odysseus startet.Gestartet wird Odysseus im Prinzip dadurch, dass alle Bundles gestartet werden - also wird Odysseus nie direkt aufgerufen.

Wichtig ist hierbei jedoch, dass die Platform keine Abhängigkeiten zu Odysseus haben darf! Daher dürfen die Bundles rcp.starter, studio.starter und server.starter auch keine Abhängigkeiten zu Odysseus haben! Dies gilt natürlich auch in die andere Richtung.

Dies ist wichtig, da sonst die Platform ebenfalls bspw. ein odysseus-core benötigt, dass dann wieder Teil der Basis-Platform wäre und dann nicht mehr aktualisierbar ist.

Die ganzen Bundles von Odysseus (core, planmanagement etc.) werden über Hauptfeatures kombiniert. Jenkins installiert im Prinzip dieses Hauptfeature, als wenn der Benutzer "install new software" über das Menü verwendet hätte.

Der grobe Aufbau der Build-Prozesse (s.o.) zu den Teilen, ist dabei wie folgt (die Zuordnung wird über die buckminster_product.properties konfiguriert):

- **"Build Odysseus Server"** nimmt das Product mit der ID "*de.uniol.inf.is.odysseus.product.server* - das ist die Datei "Odysseus Product Platform Server.product" als Basis-Platform
  - *de.uniol.inf.is.odysseus.product.server* beinhaltet das Feature *de.uniol.inf.is.odysseus.product.server.platform.feature*
  - *de.uniol.inf.is.odysseus.product.server.platform.feature* definiert die Platform für das Server-Produkt durch die Bundles: Equinox/Eclipse + *de.uniol.inf.is.odysseus.product.server.starter*
  - *de.uniol.inf.is.odysseus.product.server.starter* startet alle installierten Bundles - also auch Odysseus, dass jedoch durch das Hauptfeature installiert wird.
  - Zu dem Product-Definition installiert "Build Odysseus Server" zusätzlich das Feature *de.uniol.inf.is.odysseus.product.server.feature*
  - *de.uniol.inf.is.odysseus.product.server.feature* bündelt alle notwendigen Features von Odysseus (core, planmgt, relational...)

- **"Build Odysseus Studio"** nimmt das Product mit der ID "*de.uniol.inf.is.odysseus.product.studio* - das ist die Datei "Odysseus Product Platform Studio.product" als Basis-Platform
  - *de.uniol.inf.is.odysseus.product.studio* beinhaltet das Feature *de.uniol.inf.is.odysseus.product.studio.platform.feature*
  - *de.uniol.inf.is.odysseus.product.studio.platform.feature* definiert die Platform für das Studio-Produkt durch die Bundles: Equinox/Eclipse + *de.uniol.inf.is.odysseus.product.studio.starter*
  - *de.uniol.inf.is.odysseus.product.studio.starter* startet die GUI (also auch das Splash + Login etc.) und alle installierten Bundles - also auch den Client-Teil von Odysseus, dass jedoch durch das Hauptfeature installiert wird.
  - Zu dem Product-Definition installiert "Build Odysseus Studio" zusätzlich das Feature *de.uniol.inf.is.odysseus.studio.feature*
  - *de.uniol.inf.is.odysseus.studio.feature* bündelt alle notwendigen Client(!!)-Features von Odysseus (Odysseus-RCP, GEF etc.)

- **"Build Odysseus Server + Studio"** nimmt das Product mit der ID "*de.uniol.inf.is.odysseus.product.studio* - das ist die Datei "Odysseus Product Platform Studio.product" als Basis-Platform
  - *de.uniol.inf.is.odysseus.product.studio* beinhaltet das Feature *de.uniol.inf.is.odysseus.product.studio.platform.feature*
  - *de.uniol.inf.is.odysseus.product.studio.platform.feature* definiert die Platform für das Studio-Produkt durch die Bundles: Equinox/Eclipse + *de.uniol.inf.is.odysseus.product.studio.starter*
  - *de.uniol.inf.is.odysseus.product.studio.starter* startet alle installierten Bundles - also auch Odysseus, dass jedoch durch das Hauptfeature installiert wird.
  - Zu dem Product-Definition installiert "Build Odysseus Server + Studio" zusätzlich das Feature *de.uniol.inf.is.odysseus.product.serverandstdio.feature*
  - *de.uniol.inf.is.odysseus.product.serverandstudio.feature* bündelt alle notwendigen Features von Odysseus (core, planmgt, relational...) und zusätzlich das Feature *de.uniol.inf.is.odysseus.studio.feature* (s.o) für die GUI (Odysseus RCP, GEF etc.)

Man kann hierbei erkennen, dass Studio und Server+Studio sich nur dadruch unterscheiden, dass jeweils ein anderes Hauptfeature hinzu installiert wird.

Da das eigentliche Odysseus über das Hauptfeature "nachinstalliert" wird, startet das Product "Odysseus Product Platform Server.product" bzw. "Odysseus Product Platform Studio.product" nicht Odysseus - es kennt Odysseus nicht einmal und darf es auch nicht kennen!!! Das heißt, dass man zum Starten per Hand (z.B. aus Eclipse heraus zum Entwickeln) ein eigenes Produkt (a la das alte Memory Store Product) brauch, das sowohl das Platform-Feature als auch das Hauptfeature beinhaltet!!

Kontrolle des Build-Prozesses

Per Browser

Man kann Jenkins unter
http://odysseus.informatik.uni-oldenburg.de:8080
aufrufen.

Unter Eclipse

Es gibt ein Eclipse-Plugin, welches normalerweise mit Mylyn mitinstaliert wird.
Wer mylyn noch nicht hat, der kann diese Update-site benutzen:
http://download.eclipse.org/mylyn/releases/latest
Dann gibt es unter Windows/ show views / Other / mylyn die view „builds“ – diese öffnen und dann kann man einen neuen Build-Server hinzufügen.
Dort dann http://odysseus.informatik.uni-oldenburg.de:8080 und seinen benutzer + password eintragen. Dann noch den harken bei Build Plans setzen, die man sehen will.
Damit kann man auch den Build aus Eclipse starten.

Features über Update-Site installieren

Wie oben beschrieben, kann man eine ausführbare Version von Odysseus unter downloads herunterladen und diese dann nachträglich um zusätzlichliche Features (z.B. CEP) erweitern, indem man die Updatesite über Help > Install New Software innerhalb von Odysseus verwendet. Odysseus lässt sich dann (wenn ein neuer Build gemacht wurde) auch aktualisieren.

Neues Feature zum Build-Prozess für Update-Site hinzufügen

Um ein zusätzliches Feature hinzuzufügen müssen folgende Schritte gemacht werden:

Neues Feature

Die entsprechenden Features müssen in der feature.xml im Bundle de.uniol.inf.is.odysseus.update.p2 unter "Included Features" hinzugefügt werden. Damit werden sie schon automatisch mit in die Update-Site integriert und sind dann "nachinstallierbar" bzw. aktualisierbar (wie unten beschrieben). Man kann die Features bei Bedarf noch in der category.xml in kategorisieren. Hierbei darauf achten, dass das Feature eine andere ID (in der feature.xml unter Overview) als das Bundle hat (jede ID muss unique sein)! Also am besten wie das Bundle selbst auch auf ".feature" enden lassen.

Produkt ändern (Features hinzufügen)

Es gibt nur zwei Product-Definition, die von Jenkins benutzt werden (s.o. bei "Aufbau des Produkts"). In verschiedenen Konfigurationen werden dann nur die notwendigen Features hinzu installiert. Entsprechend gibt es bspw. ein Feature "de.uniol.inf.is.odysseus.product.server.feature", welches alle notwendigen Features für das Server-Product zusammenfasst. Dieses wird dann zu dem Basis-Produkt (Odysseus Product Platform Server.product) hinzugepackt. Dieses ist in der buckminster_product.properties in dem updatesite-Bundle definiert. Möchte man also sein Feature hinzufügen, muss folgendes gemacht werden:

- Feature in feature.xml von updatesite Bundle hinzufügen
- Feature in feature.xml von update.p2 Bundle hinzufügen
- Feature dem "Product Feature" hinzufügen, zu welchem Produkt es später gehören soll. Also z.B. de.uniol.inf.is.odysseus.product.serverandstudio.feature für das All-in-one server+studio-Produkt.

Achtung!
Die Product-Konfigurationen (also alle *.product) werden von Jenkins ignoriert. Lediglich die "Odysseus Product Platform Studio.product" und "Odysseus Product Platform Server.product" werden als Basis verwendet. Diese Produkt darf jedoch nicht irgendwie abhängig zu Odysseus core oder anderen Features sein!!!

Man muss natürlich aufpassen, dass alle notwendigen Bundles/abhängigen Features angegeben sind und das sich das Ganze in trunk befindet oder Teil der Target-Platform sind.

Neues Produkt zum Build-Prozess hinzufügen

Wer ein komplett eigenes Produkt bauen lassen will, der sollte unter "Ein ganz neues Produkt bauen lassen" gucken. Wer nur seine eigene Konfiguration bzw. Zusammenstellung von Features auf Basis von Odysseus haben möchte, der sollte unter "Ein Odysseus-Produkt mit anderen Feature-Sets bauen lassen" schauen.

Ein Odysseus-Produkt mit anderen Feature-Sets bauen lassen

Diese Schritte bieten sich an, wenn man nur - z.B. im Vergleich zum Server+Studio-Product - zusätzliche Features hat, die immer automatisch vorinstalliert sein sollen.

1. Ein Feature erstellen, das unter "Included Features" alle Features enthält, die man haben möchte. Je nachdem ob man einen Client, Server oder Monolithic haben will, sollte man hier eines der drei folgenden Features dort aufnehmen, um die jeweilige Basis-Funktionalitäten von Odysseus bereits zu haben:
    a. Für Server: de.uniol.inf.is.odysseus.server.feature
    b. Für Client: de.uniol.inf.is.odysseus.client.feature
    c. Für Monolithic: de.uniol.inf.is.odysseus.monolithic.feature
2. Dann fügt man seine eigenen Features unter "Included Features" hinzu.
3. Das Feature muss eine eindeutige ID haben - z.B. "my.example.feature"

4. In der "buckminster_product.propertiers" im Ordner tooling im Bundle "*.updatesite" brauch man einen neuen Eintrag, der als "key=value"-Pair gesehen werden kann. Entsprechend sollte der Key in der Datei stets unique sein.
    a. Das Pair gibt die Liste (kann auch leer sein) von Features an, die hinzuinstalliert werden sollen. Der Key sei bspw. "product.example.features". Ein Feature-Eintrag entspricht dabei immer der ID des Features mit dem Anhang ".feature.group".
    Hier geben wir also "product.example.features=myexample.feature.feature.group" an.
    b. Optional kann man noch eine Location angeben, die das Zielverzeichnis angibt, wohin das fertige Produkt abgelegt wird. Sonst wird "product.destination" für das gebaute Produkt und "product.zipdestination" für das gepackte als root-Verzeichnis verwendet (später kommt eh noch ein Unterordner hinzu). Man muss dann natürlich eigene Keys definieren.
5. Dann die buckminster.cspex öffnen. Hier gehören im Wesentlichen immer zwei Einträge (die <public> Einträge) zusammen. Eins baut das Produkt mit Hilfe von Ant und das andere packt es dann in einem ZIP-File. Ob man letzteres brauch, muss man jeweils entscheiden - macht zum herunterladen jedoch Sinn...
6. Am besten kopiert man sich ein Paar. Dies sollte abhängig von der Produkt-Art sein: Client, Server oder Monolithic - also in Abhängigkeit von Schritt 1!
    a. Für Server kopiert man sich die Einträge "create.server.product" und "create.server.product.zip"
    b. Für Client kopiert man sich die Einträge "create.studio.product" und "create.studio.product.zip"
    c. Für Monolithic kopiert man sich die Einträge "create.serverandstudio.product" und "create.serverandstudio.product.zip"
7. Im Folgenden nehmen wir Monolithic als Basis an und nehmen daher die beiden Einträge "create.serverandstudio.product" und "create.serverandstudio.product.zip".
8. Nach dem kopieren nennen wir diese um - z.B. in "create.example.product" und "create.example.product.zip".
9. Dann muss man das ganze noch anpassen, in dem man die variablen (die ${...} Dinger 😉) mit den Werten ersetzt, die man zuvor in der buckminster_product.properties gewählt hat.
    a. Dies ist im Wesentlichen die Feature-Liste, wie sie unter Schritt 3a definiert haben. So wird aus <property key="features" value="${product.serverandstudio.features}" /> wird dann <property key="features" value="${product.example.features}" />
    b. Wenn man oben noch eigene Locations definiert hat, muss man hier noch "${product.destination}" und "${product.zipdestination}" ändern - so, wie man den key unter 4b gewählt hat.
    c. In der create.example.product.zip muss nun der Eintrag <attribute name="create.serverandstudio.product" /> in <attribute name="create.example.product" /> geändert werden, damit beim Auslösen von create.example.product.zip automatisch create.example.product vorher gestartet wird.
    d. Des Weiteren sollte der Name des Produkts, wie es nachher auf der Platte liegt, geändert werden, damit es nicht andere überschreibt! Also z.B. in

        <products alias="action.output" base="${product.zipdestination}/**example**">

            <path path="odysseus.**example**.${target.ws}.${target.os}.${target.arch}.zip" />

        </products>
        Das erste gibt dann den Unterordner an, der unter Schritt 4b erwähnt wird.
    e. *Nur zum Verständnis*: Das "iu" ist die sogenannte "installable unit". Hier verwenden wir das existierende wieder. Bauen wir also ein monolithic-Produkt mit unseren eigenen Features, dann nehmen wir auch hier das Produkt "product.serverandstudio.id". "product.serverandstudio.id" ist wiederum in der buckminster_product.properties definiert und ist das Produkt mit der ID "de.uniol.inf.is.odysseus.product.studio". Das entspricht dann der "Odysseus Product Platform Studio.product" im Bundle "der.uniol.inf.is.odysseus.ci.products" - Dieses Produkt ist so nicht ausführbar und enthält im Wesentlichen nur einen leeren OSGi-Container (also Equinox) + RCP, GEF etc.**Hinweis:** Dieses Produkt kennt Odysseus nicht und hat auch keine Abhängigkeiten! Dies ist notwendig, damit die hinzuinstallierten Features, die man unter Schritt 1 angibt, nichts mit dem Produkt hier zu tun haben und einfach nur "hinzuinstalliert" werden - dadurch können die hinzuinstallierten Features über den Updatemechanismus konfliktfrei aktualisiert werden.
10. Das ganze dann commiten.
11. Dann Jenkins öffnen und am besten einen "neuen Job anlegen" und die Option "Kopiere bestehenden Job" auswählen. Dort "Build Odysseus Server+Studio" (oder entsprechend Server oder Client - je nach Schritt 1) auswählen und "OK" klicken.
12. Im Step "Run Buckminster" im Feld "Kommandos" ändert man nun die Einträge mit "perform" ab. Der Teil am Ende hinter dem # gibt das Kommando entsprechend aus der buckminster.cspex an, das ausgeführt werden soll. Entsprechend ändert man diese wie folgt ab:
    a. der Eintrag, der auf "#site.p2.publish" endet, kann entfernt werden - dieser sorgt nur dafür, dass die Updatesite aktuell ist - welches jedoch schon von anderen Build-Jobs gemacht wird.
    b. Dann gibt es für jedes Betriebssystem für jeweils 32 und 64bit einen Eintrag, der jeweils auf #create.server.product.zip endet. Diese ändert man am Ende auf den Namen aus Schritt 8 um - also hier auf "create.example.product.zip"
    c. Möchte man nur ein bestimmtes Setting - z.B. Windows mit 64bit, dann kann man die anderen 5 Einträge auch löschen.
13. Hat man eine andere Ausgabe-Location in der buckminster.cspex bzw. in der buckminster_product.properties gewählt, dann muss man noch ggf. den ersten Schritt im Job "Shell ausführen" anpassen, sodass auch dort die entsprechenden Ordner bereinigt werden.
14. Das ganze Speichern und den Job ausführen.

Ein ganz neues Produkt bauen lassen

Um ein komplett eigenes Produkt von Jenkins bauen zu lassen, muss folgendes gemacht werden. **Achtung**: Teile des Produkts sind möglicherweise nicht über den Update-Mechanismus aktualisierbar!

1. Dem Produkt (also der *.product Datei) eine unique ID geben (z.B. "mein.example.product"), die bisher noch nicht von einem anderen Product verwendet wird (Paketnamen + product hat sich hier bewährt).
2. In der "buckminster_product.propertiers" unter tooling im Bundle updatesite brauch man zwei neue einträge, die jeweils als "key=value" pair gesehen werden können. Entsprechend sollte der Key unique sein.
    a. Man brauch einen Key, um die Product-ID zu definieren, z.b. "product.example.id". die ID ist dann gleich mit dem aus der *.product-Datei. Das ergibt dann "product.example.id=mein.example.product"
    b. Ein zweiter Key ist eine Liste (kann auch leer sein) von Features, die hinzuinstalliert werden sollen. Der Key sei bspw. "product.example.features". Ein Feature-Eintrag entspricht dabei immer der ID des Features mit dem Anhang ".feature.group".
    Nur die hinzuinstallierten Features sind später aktualisierbar!
    c. Optional kann man noch eine Location - also ein Zielverzeichnis angeben, sonst wird "product.destination" für das gebaute Produkt und "product.zipdestination" für das gepackte verwendet. Man muss dann natürlich eigene Keys definieren.
3. Dann die buckminster.cspex öffnen. Hier gehören im wesentlichen immer zwei einträge zusammen. Eins baut das Produkt mit Hilfe von ant und das andere packt es dann in einem ZIP-File. Ob man letzteres brauch, muss man jeweils entscheiden.

4. Am besten man kopiert sich einmal die beiden einträge "create.serverandstudio.product" und "create.serverandstudio.product.zip" und nennt sie um - z.B. in "create.example.product" und "create.example.product.zip". Dann muss man das ganze noch anpassen, in dem man die variablen (die ${...} Dinger 😉) mit den Werten ersetzt, die man zuvor in der buckminster_product.properties gewählt hat.
    a. also aus <property key="iu" value="${product.serverandstudio.id}" /> wird dann <property key="iu" value="${product.example.id}" />
    b. und aus <property key="features" value="${product.serverandstudio.features}" /> wird dann <property key="features" value="${product.example.features}" />
    c. Wenn man oben noch eigene locations definiert hat, muss man hier noch "${product.destination}" und "${product.zipdestination}" angeben.
    d. In der create.example.product.zip muss nun der Eintrag <attribute name="create.serverandstudio.product" /> in <attribute name="create.example.product" /> geändert werden, damit beim Auslösen von create.example.product.zip automatisch create.example.product vorher gestartet wird.
    e. Des Weiteren sollte der Name des Produkts, wie es nachher auf der Platte liegt, geändert werden, damit es nicht andere überschreibt! Also z.B. in

        <products alias="action.output" base="${product.zipdestination}/**example**">

            <path path="odysseus.**example**.${target.ws}.${target.os}.${target.arch}.zip" />

        </products>
5. Das ganze dann commiten.
6. Dann Jenkins öffnen und am besten einen "neuen Job anlegen" und die Option "Kopiere bestehenden Job" auswählen. Dort "Build Odysseus Server" auswählen und "OK" klicken.
7. Im Step "Run Buckminster" im Feld "Kommandos" ändert man nun die Einträge mit "perform" ab. Der Teil am Ende hinter dem # gibt das Kommando entsprechend aus der buckminster.cspex an, das ausgeführt werden soll. Entsprechend ändert man diese wie folgt ab:
    a. der Eintrag, der auf "#site.p2.publish" endet, kann entfernt werden - dieser sorgt nur dafür, dass die Updatesite aktuell ist - welches jedoch schon von anderen Build-Jobs gemacht wird.
    b. Dann gibt es für jedes Betriebssystem für jeweils 32 und 64bit einen Eintrag, der jeweils auf #create.server.product.zip endet. Diese ändert man am Ende auf den Namen aus Schritt 4 um - also hier auf "create.example.product.zip"
    c. Möchte man nur ein bestimmtes Setting - z.B. Windows mit 64bit, dann kann man die anderen 5 Einträge auch löschen.
8. Hat man eine andere Ausgabe-Location in der buckminster.cspex bzw. in der buckminster_product.properties gewählt, dann muss man noch ggf. den ersten Schritt im Job "Shell ausführen" anpassen, sodass auch dort die entsprechenden Ordner bereinigt werden.
9. Das ganze Speichern.

Interne Funktionsweise

Im Folgenden noch ein paar Informationen zum Aufbau und zur Funktionsweise.

- Jenkins selbst kann keine Eclipse-Anwendungen bauen. Dazu ist ein zusätzliches Tool namens "Buckminster" (http://www.eclipse.org/buckminster/) zuständig, welches den headless Build ermöglicht. Das heißt, dass Jenkins selbst zwar eine Buckminster-Integration hat, letztendlich aber auch nur Buckminster aufruft. Das headless Buckminster wurde über das Director-Tool (ist im Prinzip ein Tool, um mit p2-Updatesites ohne Frontend umgehen zu können) installiert. Beides befindet sich unter /var/lib/jenkins/tools/Buckminster/. Buckminster wiederrum hat zwei plugins installiert, die nötig sind, damit man Eclipse-RCP-Anwendungen bauen kann: der headless core und der headless pde Teil von Eclipse. Es gibt auch ein Buch, welches einiges zu Buckminster erklärt: BuckyBook.pdf. Aber hier möchte ich auch darauf hinweisen, dass es sich hier um einen Headless-Build in Kombination mit Jenkins handelt und das Buch hauptsächlich Buckminster als IDE-Tool behandelt - also wie es händisch über Eclipse verwendet und ausgeführt werden kann)

Der grobe Ablauf des Builds ist dann wie folgt:

- "Fetsch and Build Target Platform" muss vorher einmal gemacht worden sein. Da dies der Fall ist und sich selten etwas ändert, wird dieser Schritt nur manuell ausgeführt (siehe unten). Der automatische Build geht dann wie folgt los:
- Als erstes wird ein revert und svn update in den Workspace gemacht: /opt/jenkins/workspace/odysseus
- Dann werden die Ausgabe-Verzeichnisse unter /opt/jenkins/output/nightly gelöscht
- Dann wird die de.uniol.inf.is.odysseus.creatermap/src/de/uniol/inf/is/odysseus/creatermap/CreateRMap.java compiliert und ausgeführt. Diese erzeugt eine lokale - neue site.rmap. Diese Datei sagt Buckminster im Prinzip welche Bundles und Features verwendet werden sollen und wo sie sich im Dateiverzeichnis (im Workspace) befinden. Die Java-Datei CreateRMap ist daher nur ein Programm, welches eine site.rmap für alle Bundles und Features erzeugt, die sich unter trunk befinden (so dass die site.rmap nicht immer per Hand angepasst werden muss). Wichtig hierbei ist wie oft erwähnt, dass die Bundles, Features und Produkte eindeutige (unique) IDs haben.
- Anschließend wird Buckminster (also der headless RCP/Eclipse-Build ausgelöst). Dieser verwendet die Target-Platform, die mit dem Build-Schritt "Fetch and Build Target Platform" erzeugt wurde und zusätzlich die site.rmap. Das heißt, dass das headless-Build im Prinzip ein temporäres Sammlung von Bundles erzeugt, in denen zum einen die Bundles sind, die über die Target-Platform heruntergeladen wurden und zum anderen auch alle Bundles sind, die in der site.rmap definiert sind (welches alles in trunk sind, da diese ja dynamisch generiert wird). Beides (Target-Platform und site.rmap) wird also als erstes importiert.
- Dann wird der build ausgeführt, indem die Features gebaut werden, die in der feature.xml in de.uniol.inf.is.odysseus.update.p2 definiert wurden. Hierbei erkennt man, dass natürlich dann auch alle notwendigen Bundles eines dort definierten features entweder Teil der Target-Platform sein müssen oder selbst als Bundle im trunk vorliegen müssen.
- Das Ergebnis wird nach /opt/jenkins/output/nightly/result geschrieben. Die Update-Site wird dann unter /opt/jenkins/output/nightly/update erzeugt. Dies ist auch exakt das Verzeichnis, welches über einen dynamischen Link von http://odysseus.informatik.uni-oldenburg.de/update/ aufgerufen wird.
- Dann werden die Produkte gebaut. Dazu nimmt Buckminster (welches ja von Jenkins ausgeführt wird) die buckminster.cspex und ruft die dort angegebenen Actions (je Produkt zwei actions: einmal das Bauen des Produktes selbst und einmal das Ergebnis zippen) auf. Diese rufen wiederrum jeweils das ANT-File product.ant auf, welches dann einen headless-build mit Hilfe von Equinox (ruft also im Prinzip eine Jar mit Java auf) ausführt.
- Wenn ein Produkt gebaut wurde, dann wird es nach /opt/jenkins/output/nightly/downloads geschrieben. Jeweils in einen Unterordner, der in der buckminster.cspex mit (action.output) definiert ist. Dieses Verzeichnis ist im Apache (siehe Lampp) als http://odysseus.informatik.uni-oldenburg.de/download/studio/nightlybuild/ eingehängt und somit extern zugreifbar.
- Ist ein Produkt fertig, stößt die Build-Pipeline den nächsten Build für ein anderes Produkt an

Wird "Fetch and Build Target Platform" manuell ausgeführt, dann wird zunächst

- der Ordner "trunk/ci" des SVN ausgecheckt (in den Ordner /var/lib/jenkins/workspace/shared2)
- dann wird die Datei "target-platform.target" aus dem Projekt de.uniol.inf.is.odysseus.updatesite geöffnet. und die Target-Platform wird anhand der Update-Sites, die in target-platform.target definiert sind, heruntergeladen (dazu wird das verzeichnis /var/lib/jenkins/output/temp2 als temporäres Verzeichnis genutzt und das Ergebnis wird nach /var/lib/jenkins/output/result2 geschrieben
- danach wird das Ergebnis (also alle Bundles der Target-Platform, die heruntergeladen wurden) archiviert - das heißt, dass die Target-Platform dann im Prinzip für zukünftige (andere) Builds persistiert wird.
- Anschließend werden die anderen Builds angestoßen (siehe oben)

Additional Build Processes

In addition to the aforementioned build processes, Jenkins performs the following builds.

Docker Image

This build job takes a Dockerfile and creates a Docker image of the latest Odysseus Server build. To do so, a custom file is defined in *Config File Management* that contains all processing steps (see Manage Jenkins -> Managed files -> Custom file). This custom file is copied to the workspace during the build as *Dockerfile*. The build process starts a Docker build by issuing the following command:

```
docker build -t odysseus/latest .
```

After a successful build, the artifact will remain in the workspace. Marco Grawunder can we copy the artifact somewhere to provide it as a download or pull it to dockerhub?