COPPER Best Practices

Version 1.0.1

Wann sollte man überhaupt COPPER verwenden?

Allgemein genau dann, wenn man von der COPPER Notation oder den COPPER-Features profitieren kann. Ein wesentliches Feature von COPPER ist die sehr gute Unterstützung für asynchrone Verarbeitung mittels einer Java-Notation:

Wenn Service Calls zu externen Systemen oder zur Datenbank asynchron erfolgen können oder müssen, bietet COPPER den Vorteil einer sequentiellen Notation. Diese Java-Notation ermöglicht ein asynchrones Warten ohne den aktuellen Thread zu binden/blockieren.

```
@Override
public void main() throws InterruptException {
	System.out.println("started");
	final String cid = mockAdapter.foo("foo");
	wait(WaitMode.ALL, 5000, cid); // asynchronous wait
	Response<?> r = getAndRemoveResponse(cid);
	System.out.println("finished - response="+r);
}
```

Der aktuelle Thread wird beim COPPER wait sofort freigegeben und kann so weitere Workflow-Instanzen ausführen. Erst wenn die asynchrone Antwort eintrifft, wird erneut ein freier Thread für die Weiterverarbeitung der Workflow Instanz verwendet. Dies ermöglicht es, mehr Workflow-Instanzen mit asynchroner Kommunikation parallel auszuführen, als Betriebssystem-Threads zu Verfügung stehen. Dies ist insbesondere dann ein Vorteil, wenn die Antwortzeiten auf die asynchronen Requests überwiegen lang sind und somit ein großer Teil der Threads mit Warten beschäftigt wäre. Darüber hinaus ist die Verarbeitung schneller und effizienter, weil das Betriebssystem nicht so viele Thread-Context-Wechsel durchführen muss.

Neben dieser verbesserten Skalierbarkeit, bietet COPPER auch feinere Mechanismen zur Priorisierung von Workflow-Instanzen an. Jede Workflow-Instanz hat einen numerischen Wert als Priorität. Diese Priorität kann zur Laufzeit dynamisch geändert werden und wirkt sich darauf aus, ob eine Workflow-Instanz gegenüber einer anderen Instanz bevorzugt behandelt wird. Es ist ferner möglich COPPER so zu erweitern, dass eigene spezielle Priorisierungsregeln verwendet werden.

Wann sollte man einen Transient-Workflow erstellen und wann besser einen Persistent-Workflow?

Transiente Workflows sind, wie der Name schon andeutet, nicht in einem permanenten Medium wie einer Datenbank gespeichert, sondern existieren nur in dem Java-Prozess der sie erzeugt und gestartet hat. Transiente Workflows verwendet man oft dann, wenn man nur lesende Requests verarbeiten muss, oder wenn Workflow-Instanzen nicht über die Lebenszeit des entsprechenden Java-Prozesses hinaus existieren müssen. Im Allgemeinen also genau dann, wenn man keines der im Folgenden beschriebenen Features der Persistenten Workflows benötigt.

Persistente Workflow existieren nicht nur in dem Java-Prozess der sie erzeugt und gestartet hat, sondern werden zu bestimmten Zeitpunkten in einem permanenten Medium, üblicherweise einer Datenbank, persistiert. Ihr Zustand kann dann zu einem späteren Zeitpunkt wiederhergestellt werden.

Man verwendet persistente Workflows in der Regel dann, wenn eine oder mehrere der folgenden Anforderungen zu erfüllen ist:

- Langläufer
    - Workflow Instanzen existieren über einen langen Zeitraum, möglicherweise Tage, Wochen oder sogar Monate. In diesem Fall müssen die Workflow-Instanzen das Stoppen und Starten der Anwendung überleben.
- Crash Recovery
    - Workflow Instanzen müssen auch den Crash der Anwendung überleben. Dazu werden nach einzelnen Verarbeitungsschritten sogenannte Synchronisations-Punkte bzw. Checkpoints in die Datenbank geschrieben. Bei COPPER wird für eine persistente Workflow-Instanz immer genau dann ein Checkpoint geschrieben, wenn ein COPPER wait oder resubmit ausgeführt wird.
- FailOver bzw. Lastverteilung bei mehreren Knoten
    - Die Anwendung soll auf einem Cluster, also mehreren verbundenen Knoten laufen. Dabei soll bei Ausfall eines Knotens dessen Last automatisch auf die verbliebenen Knoten verteilt werden. Auch im Regelbetrieb wird die Last auf alle Knoten verteilt.
- Manuelles und Automatisches Retry
    - Im Fehlerfall müssen Workflow-Instanzen ausgesteuert werden und später automatisch oder manuell wieder angestoßen werden.

Welche Daten kommen in den Workflow?

Daten in einem Workflow sind die Felder bzw. Members der Workflow Java-Klasse, das *Data Object*s im Workflow (siehe de.scoopgmbh.copper.Workflow.getData()) und alle Daten die bei der Ausführung einer Workflow-Methode auf dem Stack liegen, also lokale Variablen und Methoden-Parameter. Bei den transienten Workflows gibt es aus technischer Sicht keine Einschränkungen bzgl. der Verwendung von beliebigen Datentypen innerhalb des Workflows.

Bei persistenten Workflows gibt es einige technische und organisatorische Einschränkungen die beachtet werden sollten. Dazu zuerst kurz eine Aufzählung aller Daten, die beim Erzeugen eines Checkpoints mit der Workflow Instanz gespeichert werden:

- Alle Felder im Workflow, die nicht *transient* deklariert ist,
- Das zum Workflow gehörige *Data-Object*
- Alles was bei Ausführung des COPPER-wait auf dem Stack liegt. Das sind die lokalen Variablen und auch alle Methoden Parameter.

Daraus lassen sich die folgenden Best Practices ableiten:

- Nur die wirklich benötigten Daten in den Workflow legen. Weniger Daten erzeugen einen kleineren Footprint und das ermöglicht eine bessere Performance, denn alle Daten müssen im Rahmen eines Checkpoints gespeichert werden.

- Vermeide lokale Variablen in Workflow-Methoden, die direkt oder indirekt ein COPPER wait verwenden, also einen Checkpoint schreiben.
    - Verschiebe Code-Blöcke mittels Refactoring in eigene Methoden
    - Setze nicht mehr benötigte lokale Variablen oder Parameter auf null. Dadurch müssen diese nicht persistiert werden.
- Minimiere die Abhängigkeiten
    - Verwende keine extern definierten Schnittstellen-Datenstrukturen
        - auf eigene Datenstrukturen mappen
    - Verwende weniger Datentypen, dadurch entstehen weniger Abhängigkeiten, wodurch eine später evtl. nötige Migration von Workflow-Instanzen vereinfacht wird.
- Referenzen auf Service Beans innerhalb eines Workflows müssen als transient deklariert werden. Die Beans können und sollten von COPPER per AutoWire gesetzt werden.
- COPPER verwendet als Default die Java Object Serialization. Das bedeuted, dass alle Klassen von Daten-Objekten das Java Interface Serializable implementieren müssen.

Welcher Code kommt in den Workflow?

Der Workflow implementiert die fachliche Ablauf-Semantik (high level). Im Workflow Code, also den Methoden, sollte daher möglichst nur die Flusskontrolle der logischen Einzelschritte des Workflows enthalten sein. Ein Beispiel:

- Rufe System A auf und warte auf Anwort A
- Schreibe einen AuditTrail-Eintrag
- Abbruch, wenn A mit einem Fehler geantwortet hat
- Für alle Elemente in der Antwort von A:
    - Übermittele das Element an B und warte auf Antwort von B
    - Schreibe einen AuditTrail-Eintrag
- …

Die eigentliche Verarbeitung, also beispielsweise das Mapping von einer internen auf eine externe Schnittstelle, sollte außerhalb des Workflows liegen.

Es gilt außerdem, genauso wie für die Daten, dass die Abhängigkeiten zu anderem SourceCode so klein wie möglich gehalten werden sollte. Abhängigkeiten zu externen Schnittstellen sollten möglichst vermieden werden.

Die Verwendung von Loggern, zum Beispiel Simple Logging Fascade for Java oder Log4J ist nicht problematisch.

Um die Abhängigkeiten in einem COPPER Projekt zu minimieren und den SourceCode sauber aufzuteilen, bietet sich die folgende Projekt-Struktur an.

Wie soll ich mein COPPER Projekt strukturieren?

Getrennte Subprojekts für

- Workflows

    - Enthält die COPPER Workflows. Hat nur eine Abhängigkeit zum SubProjekt „Interne Schnittstellen“. Dadurch werden Abhängigkeiten zu deren Implementation oder zu externen Schnittstellen oder Datenstrukturen verhindert.
- Interne Schnittstellen
    - Enthält die intern verwendeten Schnittstellen und Datenstrukturen
- Externe Schnittstellen
    - Enthält alle extern verwendeten Schnittstellen und Datenstrukturen, zum Beispiel aus WSDL generiertes JAX-WS und JAXB Binding.
- Implementierung der Schnittstellen
    - Enthält die Implementierung der intern verwendeten Schnittstellen. Hierin können zum Beispiel auch Adapter enthalten sein, die die Service Calls auf externe Schnittstellen kapseln.

Wie versioniere ich meine Workflows?

Nachdem ein Workflow in einer produktiven Umgebung deployed wurde, werden oft noch Änderungen wie zum Beispiel Bugfixes und Weiterentwicklungen an dem Workflow gemacht. Ist der Workflow persistent, dann stellt sich die Frage, ob bereits in einer Umgebung existierende Workflow-Instanzen nach dem Deployment der neuen Workflow-Version noch lauffähig sind.

Bei COPPER ist es grundsätzlich so, dass eine persistente Workflow-Instanz den entsprechenden Workflow über dessen Java-Klassennamen referenziert. Wird ein Workflow verändert, und neu deployed, dann verwenden alle ab diesem Zeitpunkt neu erzeugten Workflow-Instanzen und alle bereits existierenden und noch aktiven Workflow-Instanzen diesen neuen Workflow.

Dieses Feature kann hilfreich sein, wenn eine Änderung sich auch auf existierende Workflow-Instanzen auswirken soll. In diesem Fall ist Sorge zu tragen, dass nur abwärtskompatible Änderungen am Workflow gemacht werden. Das Dokument „COPPER-workflow-compatibility-rules.pdf“ enthält eine Aufzählung von kompatiblen Änderungen.

Eine solche Änderung sollte mit Vorsicht durchgeführt werden. COPPER überprüft beim Deployment nicht, ob vorhandene Workflow-Instanzen noch kompatibel sind. Bei Inkompatibilitäten kann es dementsprechend zu Laufzeitfehlern kommen, wenn die Workflow-Instanz wieder aktiviert wird. Im Fehlerfall setzt COPPER die Workflow-Instanz dann in einen entsprechenden Fehler-Zustand und setzt die Ausführung bis zu einem manuellen Retry aus. Dies ermöglicht es, die fehlerhafte Instanz zu korrigieren und später wieder auszuführen.

Einfacher ist es, wenn eine Änderung sich nicht auf existierende Workflow-Instanzen auswirken soll. In diesem Fall erstellt man eine Kopie des Workflows bzw. von dessen Java-Klasse. Dazu ist es auch notwendig einen neuen Klassen/Workflow-Namen zu vergeben. Bewährt hat es sich, eine fortlaufende Versionsnummer im Klassennamen zu verwenden, z.B. FooWorkflow_001, FooWorkflow_002, usw.

Wichtig ist, dass der alte Workflow solange mit im Deployment enthalten ist, solange noch entsprechende Workflow-Instanzen in einer Umgebung vorhanden sind. Erst wenn es in keiner Umgebung mehr Instanzen eines Workflows gibt, kann und soll der Workflow gelöscht werden.

Damit an den Stellen im SourceCode, an denen Workflow-Instanzen erzeugt und gestartet werden, nicht nach der Einführung einer neuen Workflow-Version der dazugehörige Sourcecode geändert

werden muss, gibt es in COPPER die Möglichkeit einem Workflow eine WorkflowDescription-Annotation hinzuzufügen. Diese enthält einen Alias und Versionsnummern für Major- und Minor-Version und den Patchlevel.

Es ist dann möglich, eine Workflow-Instanz für einen Alias und eine bestimmte Version oder die neueste Version zu erzeugen, z.B. so

```
engine.run(new WorkflowInstanceDescr<Data>("wfAliasName", new Data(...), null, null, null, new WorkflowVersion(5,1,0));
```

Die Best Practices zum Thema Versionierung lassen sich wie folgt zusammenfassen:

- Nach dem Deployment in eine produktive Umgebung sollte ein Workflow in der Regel nicht mehr geändert werden.
- Muss ein produktiver Workflow dennoch geändert/gepatcht werden, gelten die Kompatibilitäts-Regeln in Dokument: „COPPER-workflow-compatibility-rules.pdf“
- Der Klassenname oder das Package eines Workflow enthält üblicherweise eine Versionsnummer, bspw. FooWorkflow_001
- Neue Versionen eines Workflows werden als Kopie der vorhergehenden Version mit hochgezählter Versionsnummer erstellt.
- Workflows können durch Verwendung der „WorkflowDescription“ Annotation mit einem gemeinsamen „Alias“ und Version-Informationen versehen werden. Über die Verwendung des Alias kann dann eine Workflow-Instanz für eine bestimmte Version oder die neuste Version für ein Major- oder Minor-Release gestartet werden (siehe „public void run(WorkflowInstanceDescr<?> wfInstanceDescr)“)