

Mannschaftssport –

UML-Modellierung in verteilten Teams

Autor: Thorsten Sturm, Gentleware AG

Während die Arbeit in verteilten Teams bei der Programmierung bereits gängige Praxis ist, beschränken sich die meisten Projekte bei der Modellierung auf wenige, überwiegend lokal verfügbare Mitarbeiter. Die fehlende Werkzeugunterstützung für verteiltes Modellieren ist dabei einer der Hauptgründe. Dieser Artikel gibt eine Übersicht über die notwendigen Fähigkeiten, die Modellierungswerkzeuge verteilten Teams zur Verfügung stellen sollten.

Nahezu jedes aktuelle Softwareprojekt wird in Teams entwickelt. Für die meisten Programmiersprachen stehen Werkzeuge zur Verfügung, die verschiedene Aspekte der verteilten Programmierung mit hilfreichen Lösungen unterstützen. Neben Systemen, die den eigentlichen Quelltext, Anforderungen oder Fehlermeldungen verwalten, werden auch automatisierte Testumgebungen und sogar verteilte Editoren (siehe Link-Sammlung) angeboten.

Die am Markt verfügbaren Modellierungswerkzeuge konzentrieren sich vorwiegend auf die Unterstützung des einzelnen Entwicklers. Ansätze auch Teams bei der Modellierung ihrer Lösungen zu unterstützen, bestehen in den meisten Fällen aus Möglichkeiten komplexe Projekte in zahlreiche Unterprojekte aufzuteilen und der Anbindung an die bereits genannten Versionsverwaltungssysteme. Modellierungsumgebungen, die speziell auf die Bedürfnisse von modellierenden Teams zugeschnitten sind, sucht man bisher vergeblich. Dabei weisen die bisher verfügbaren Lösungen eklatante Schwächen auf, die ihre Nutzbarkeit in verteilten Umgebungen nachhaltig beeinträchtigen können.

„Der Mannschaft fehlt es an Teamgeist“

Die Aufteilung eines größeren Projektes in mehrere Unterprojekte (Partitionierung) erfordert eine sehr genaue Analyse der möglichen Schnittstellen. Die entstandenen Unterprojekte benötigen größtmögliche Unabhängigkeit voneinander, um Konflikte zwischen Änderungen an verschiedenen Unterprojekten zu vermeiden. Trotz aller Sorgfalt lassen sich absolut unabhängige Unterprojekte leider nur in den seltensten Fällen erstellen. Eine genaue Absprache über die vorzunehmenden Änderungen ist daher weiterhin unerlässlich. Einmal auftretende Konflikte lassen sich vielfach nur sehr mühsam und unter großem Zeitaufwand wieder auflösen, da sie nicht sofort erkannt und behandelt werden können.

Ein kleines Beispiel soll illustrieren, welche Probleme der verteilten Modellierung von den verfügbaren Lösungen aufgeworfen werden. Unser Beispielprojekt ist eine kleine, inhaltslose Applikation, die aus zwei Paketen besteht. Während *sampleBasics* unsere eigentliche Applikation darstellt, liefert *sampleUtils* einige unterstützende Klassen (siehe Abbildung 1). An unserem Projekt sind zwei Entwickler beteiligt. Ihre Aufgabe besteht in der Ergänzung einer neuen Funktion zu unserer Applikation.

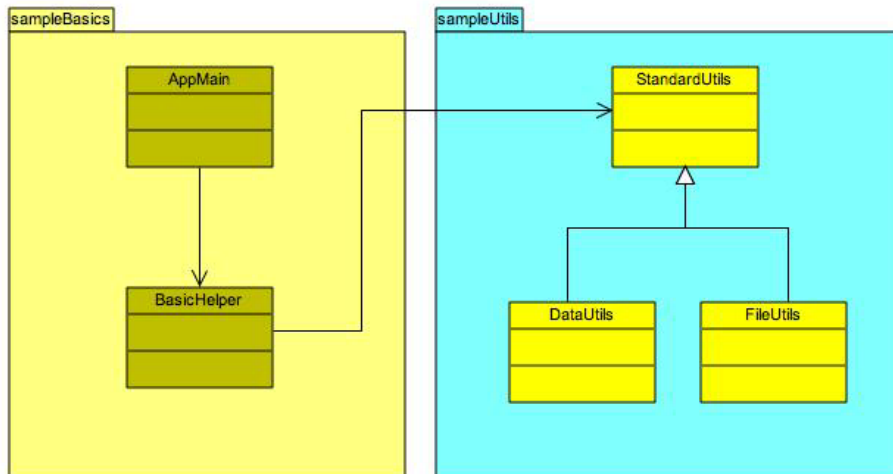


Abbildung 1. Struktur des Beispielprojektes

Wir versuchen nun unser Projekt in zwei Unterprojekte aufzuteilen. Ein häufig verwendetes Muster ist die Trennung anhand von Paketstrukturen. Dem folgend trennen wir unser Projekt auf und erhalten je ein Unterprojekt für die Pakete *sampleBasics* und *sampleUtils*.

Im Rahmen der Modellierung erkennt der für *sampleUtils* zuständige Entwickler, dass die Klasse *StandardUtils* überflüssig ist und durch eine neue Klasse namens *PlainUtils* ersetzt werden sollte. Das Modell wird im Unterprojekt entsprechend geändert. In der Zwischenzeit hat der andere Entwickler im Modell für *sampleBasics* eine neue Klasse *BasicUtils* eingeführt, die von *StandardUtils* erbt. Die Klasse *AppBasics* hat zudem eine neue Methode *getUtils()* erhalten, die ein Objekt vom Typ *StandardUtils* zurückliefert.

Sobald beide Unterprojekte wieder zusammengeführt werden, entstehen offensichtlich ein Konflikte. Diese würden noch dadurch verschärft werden, wenn eine Implementierung von *getUtils()* Gebrauch vom bereits gelöschten Typ *StandardUtils* machen würde. Bereits durch relativ einfache Veränderungen an Unterprojekten ergeben sich Konflikte, deren Auflösung wertvolle Entwicklungszeit kostet.

Der Konflikt hätte natürlich auch vermieden werden können. Durch genaue Absprache der beiden Entwickler untereinander und häufiges Zusammenführen der Unterprojekte wäre das Problem wahrscheinlich umgangen worden. Leider erfordert diese Form der Konfliktvermeidung zusätzlichen Aufwand und wird nur unzureichend durch Werkzeuge unterstützt.

Die Nutzung von Versionsverwaltungssystemen wirft weitere Probleme auf. Typischerweise arbeiten diese Systeme auf Basis von Dateien. Bei Quelltext-Dateien sind Vergleich, Zusammenführung und Konfliktbehandlung von unterschiedlichen Datei-Versionen relativ einfach, da es sich um reine Textdateien handelt. Je nach benutztem Dateiformat sind Modell-Dateien bedeutend empfindlicher und jede ungewollte Veränderung kann unter Umständen dazu führen, dass das gesamte Modell unbrauchbar wird.

„Der Star ist die Mannschaft“

Schon aus recht einfachen Szenarien wird schnell deutlich, dass die meisten Probleme durch das Auftreten von Konflikten entstehen. Die Vermeidung von Konflikten sollte somit das vordringliche Ziel einer Werkzeugunterstützung für verteilte Modellierung sein. Aber wie erreicht man das?

Neben der räumlichen Verteilung der beteiligten Entwickler spielt bei der Entstehung der Konflikte besonders die asynchrone Zusammenführung der unterschiedlichen Änderungen eine wichtige Rolle. Vermeidet man die zeitliche Differenz zwischen Durchführung der Änderung und Zusammenführung mit anderen Änderungen, vermeidet man auch einen Großteil der Konflikte.

Letztlich ist allerdings eine zentrale Instanz notwendig, die entscheiden kann, ob eine Änderung konfliktfrei durchgeführt werden kann. Der aktuelle und korrekte Stand des Modells liegt zu jedem Zeitpunkt in dieser Instanz vor und kann von dort aus auch gespeichert werden. An dieser Stelle

können dann auch Versionsverwaltungssysteme problemlos ansetzen. Da es nur eine speichernde Instanz gibt, treten bei der Versionierung keine Konflikte auf und die Modell-Datei verbleibt in einem konsistenten, funktionsfähigen Zustand.

Keine noch so ausgefeilte Konfliktbehandlung ersetzt hingegen die Kommunikation zwischen den beteiligten Entwicklern. Auch wenn im besten Fall alle Entwickler in einem Raum sitzen, sollte eine angemessene Unterstützung für die Kommunikation zwischen den Entwicklern bereits vom Modellierungswerkzeug geboten werden. Chat-Systeme oder Instant Messenger haben sich bereits als hilfreiche und einfach zu handhabende Werkzeuge bei der Unterstützung verteilter Entwicklung erwiesen.

Wenden wir das oben beschriebene Szenario auf eine entsprechend ausgestattete Modellierungsumgebung an, werden die damit verbundenen Vorteile sofort sichtbar. Wir haben weiterhin zwei Entwickler, die getrennt voneinander die Pakete *sampleBasics* und *sampleUtils* bearbeiten. Die Reaktion auf das Löschen der Klasse *StandardUtils* und ihrer gleichzeitigen Nutzung als Oberklasse unterscheidet sich allerdings deutlich. Da die Klasse *StandardUtils* bereits gelöscht wurde, kann sie nicht als Oberklasse oder Typ genutzt werden. Sollten beide Änderungen zeitlich zusammen fallen, entscheidet die Reihenfolge, mit der sie bei der steuernden Instanz ausgeführt werden, darüber, ob ein Konflikt entsteht. Wird zunächst die Erstellung der Vererbungsbeziehung und danach die Löschung der Klasse *StandardUtils* bearbeitet, entsteht kein Konflikt. Es ist davon auszugehen, dass der betroffene Entwickler sich die Löschung der betroffenen Klasse wohl überlegt hat. Im umgekehrten Fall entsteht ein Konflikt, da eine Vererbungsbeziehung zu einem bereits gelöschten Typ aufgebaut werden soll. Der auslösende Entwickler erhält eine Warnung und kann gezielt auf diesen Konflikt reagieren. Zusätzliche Absprachen zwischen den Entwicklern, beispielsweise über einen Instant Messenger, hätten allerdings auch hier den Konflikt bereits im Vorwege verhindern können. Zu jedem Zeitpunkt unseres kleinen Beispiels war das Gesamtmodell in einem konsistenten, konfliktfreien Zustand. Eine Speicherung des Projektes und dessen spätere Weiterbearbeitung ist problemlos möglich.

Mit Poseidon for UML Enterprise Edition gibt es am Markt seit kurzem ein UML-Modellierungswerkzeug, das sich speziell mit der Unterstützung von verteilter Modellierung befasst. Verschiedene Entwickler können parallel an einem Projekt arbeiten. Die Änderungen werden an eine zentrale Instanz geschickt und dort verarbeitet. Wenn kein Konflikt entsteht, wird die Änderung an alle beteiligten Entwickler verschickt und in deren lokaler Poseidon-Instanz sofort sichtbar. Sollte ein Konflikt entstehen, erhält der betroffene Entwickler umgehend eine Rückmeldung darüber und kann angemessen reagieren. Da Änderungen am Modell zeitnah und sehr kleinräumig übertragen werden, ist die Anzahl der entstehenden Konflikte ausgesprochen gering und ihre Auflösung sehr einfach zu bewerkstelligen.

Über die verteilte Modellierung hinaus unterstützt die Enterprise Edition die Kommunikation zwischen räumlich entfernten Entwicklern. Weiterhin besteht auch die Möglichkeit, die zentral gespeicherten Projekte mit einem Versionsverwaltungssystem verwalten zu lassen.

„Geschlossene Mannschaftsleistung“

Insgesamt lässt sich feststellen, dass die Werkzeugunterstützung von verteilter Modellierung noch sehr wenig verbreitet ist. Obwohl die notwendigen Konzepte relativ klar und einfach sind, scheinen erst wenige Werkzeughersteller von der Wichtigkeit der Modellierung im Team überzeugt. Trotzdem gibt es bereits heute am Markt hochwertige Lösungen, die Teams, wie verteilt sie auch sein mögen, zu einer gemeinsamen Modellierungsumgebung verhelfen, die selbst bestehenden Lösungen für die verteilte Programmierung überlegen ist.

Links:

JUnit www.junit.org

CVS www.cvs.org

Bugtracker Overview www.a-a-p.org/tools_tracking.html

SubEthaEdit www.codingmonkeys.de/subethaedit/

Poseidon for UML Enterprise Edition www.gentleware.com/products/

Über den Autor:

Thorsten Sturm (E-Mail: thorsten.sturm@gentleware.com) ist Leiter der Entwicklungsabteilung der Gentleware AG und verantwortlich für die Weiterentwicklung des Modellierungswerkzeugs Poseidon for UML. Gleichzeitig ist er als Teamleiter zuständig für die Entwicklung der teamfähigen Enterprise Edition von Poseidon.