Konfiguration und Metadaten in Danubia

Vorschlag für eine Projektarbeit. Stephan Janisch, 24.06.2005

Zusammenfassung

Es soll eine Komponente zur Verwaltung von Konfigurations- und Metadaten entwickelt werden. Die Komponente soll Konfigurations- und Metadaten einlesen (z.B. aus Dateien) und eine Abfrage der Daten über eine Java-API ermöglichen. Als Beispielanwendung soll die Komponente in Danubia integriert werden, um Konfigurations- und Metadaten für das Danubia-Framework und für konkrete Danubia-Komponenten zu verwalten.

Anforderungen

Im Kontext von Danubia gibt es eine Reihe von Konfigurationsparametern auf verschiedensten Ebenen: JVM, Danubia-Framework, DeepActor-Framework, Tools/Bibliotheken die von Danubia verwendet werden und Danubia-Komponenten (DeepActor-Modelle sind auch Danubia-Komponenten). Beispiele:

- JVM: Speicher über -Xmx, ...
- Danubia: Verzeichnis für Ergebnisdaten, Export-Schnittstelle als Result rausschreiben - Ja/Nein, Logging enabled - Ja/Nein, ...
- DeepActor: Verzeichnis und Name von Initialisierungsdateien, Datei mit Namen von Constraintklassen, Pfadnamen zu Eventklassen, ...
- Tools/APIs die in Danubia verwendet werden: z.B. Log4J benötigt eine Datei mit einer Logger-Konfiguration, ...
- Danubia-Komponenten: häufig Namen von Verzeichnissen und Dateien, aber auch Konfigurationsparameter die das Laufzeitverhalten beeinflussen (z.B. bei Rivernetwork, Household), beispielsweise Konfiguration für lokale Testläufe vs. Konfiguration auf dem Cluster.

Zu den Metadaten gehören Informationen wie *Name*, *Entwickler*, *Version*, ..., die sowohl auf Framework- als auch auf Danubia-Komponentenebene existieren.

Unklarheiten. Es ist nicht immer klar, was Konfigurations- und was Metadaten sind. Beispiele: Sind Import/Export-Schnittstellen Konfigurationsparameter ? Wo und wie werden nichtfunktionale Anforderungen formuliert (z.B. minimale CPU- und Speicherausstattung der Laufzeitumgebung) ? Durch die Beispiele stellt sich auch die Frage ob die Unterscheidung nur zwischen Meta- und Konfigurationsdaten genügt oder ob man noch weitere Unterscheidungen benötigt.

Ansatz mit Properties. Um zu verdeutlichen welche Funktionalität von der zu entwickelnden Komponente in etwa erwartet wird, skizziert Abb. 1 einen Property-basierten Ansatz. Eine `Property` ist hier ein key-value Paar. `PropertyManager` ermöglicht über einen Identifikator auf Werte von Konfigurationsparametern oder Metadaten zuzugreifen. Der `PropertyManager` kann beispielsweise vom Danubia-Framework verwendet werden um Metadaten einzelner Modelle abzufragen. Er instantiiert die konkreten Properties beispielsweise basierend auf Dateien.

Konfigurations- und Metadaten müssen i.a. verschiedene `key`-spezifischen Anforderungen berücksichtigen, z.B.

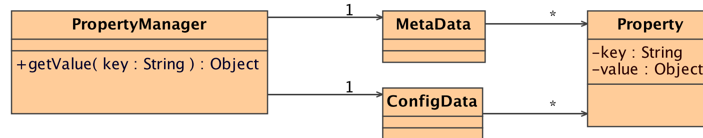

Abbildung 1: Beispiel basierend auf Properties.

- Versionsnummer hat Form X.Y.Z
- resultdata.enabled hat den Wert `true` oder `false`
- Name ist ein String
- Zeitschritt ist HOUR, MONTH oder YEAR
- Startdate $\leq$ Enddate
- Klasse ist zur Laufzeit auf dem Klassenpfad zu finden ...

Diese Anforderungen lassen sich teilweise durch eine geeignete Typisierung von `value` ausdrücken. Die zu entwicklende Komponente muss es ermöglichen derartige Anforderungen an Konfigurationsparameter und Metadaten zu formulieren und soll Mechanismen anbieten konkrete Konfigurations- und Metadaten bzgl. der jeweiligen Anforderungen zu validieren.

Eine zweite wichtige Anforderung ist Erweiterbarkeit: beispielsweise sollen Entwickler von Danubia-Komponenten nicht nur Werte für vorgegebene `keys` definieren sondern auch die Möglichkeit haben eigene `keys` zu definieren. Es soll auch einem Danubia-Entwickler möglich sein Anforderungen bzgl. der Werte selbst-definierter `keys` zu formulieren und validieren.

Technische und Methodische Anforderungen.

- Vorkenntnisse: Java 1.5, UML, FOOSE (Konzeptionell), JUnit
- Analyse: mind. informell, evtl. Use Cases
- Design: UML mit Inv/Pre/Post (mind. informell, evtl. teilweise OCL/JML)
- Implementierung: Java 1.5 mit Unittests die systematisch aus Spec abgeleitet wurden

Literatur

Zur allgemeinen Motivation dieser Projektarbeit: *Java Properties Purgatory Part 1/2* von S.D. Halloway, Aug 2002 (Google oder unter `http://www.informit.com/articles/`).

Daneben wäre es interessant und wichtig zu untersuchen wie Konfigurations- und Metadaten in anderen Frameworks wie CORMAS, Cougaar oder ZEUS[1] bzw. in Komponenten-Technologien wie EJB, COM, ... verwaltet werden. Möglicherweise liefern auch allgemeine Bücher wie *Component Software* von Szyperski oder *Catalysis* von D'Souza Hinweise auf Konzepte und Implementierungen zu diesem Thema.

Aus der speziellen Sicht von Umweltsimulationen kommen weitere konkrete Anforderungen dazu, die in der Dissertation von Endejan beschrieben werden. Evtl. liefert nicht nur die Disseration sondern auch die Referenzen dort brauchbare Hinweise auf zu berücksichtigende Anforderungen bzw. auf existierende Lösungen.

[1] Die Framework-Beispiele stammen aus der *MAS Tools and Techniques* Übersicht von AgentLink (http://www.agentlink.org, Review of Software Products for Multi-Agent-Systems, June 2002) und beziehen sich von daher immer auf MAS. Natürlich könnte man auch nach allgemeinen Framework-Implementierungen im Bereich Komponentenbasierter SW-Entwicklung suchen.