Fachvortrag: Einblicke in Design und Verfahren einer modellbasierten Anwendungsplattform

Inhalt

Ankündigung

Abstract

Die Vorteile des **Low-Code** Ansatzes beschränken sich nicht auf die verbreitet im Fokus stehenden wirtschaftlichen Aspekte, wie dem effektiven und zielgerichteten Einsatz knapper und kostspieliger Entwicklungskapazitäten und der beschleunigten Deckung des schnell wachsenden Lösungsbedarfs. Die zur Umsetzung der Low-Code Technologie erforderliche Einbettung von höherwertigen Informationen aus den Fachdomänen in die laufende Anwendung selbst eröffnet vielseitig nützliche Optionen für das Anwendungsdesign und den Entwicklungsprozess. Am Beispiel unserer Anwendungsplattform geben wir einen Einblick in die Verwendung der eingebetteten Modellinformation mit dem Schwerpunkt auf die programmierfreie Bereitstellung mehrgerätefähiger Benutzeroberflächen und die programmierfreie Anbindung heterogener Datenquellen.

Autor

Jörg Bouillon ist Gründer, CEO und CTO im Firmenverbund der JBnSE GmbH. Seine berufliche Tätigkeit im Bereich der Softwaretechnologie begann als wissenschaftlicher Mitarbeiter und Dozent an der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau. Seine lebenslange Leidenschaft ist die Erarbeitung und der Einsatz zukunftweisender Technologien und Methoden in der Softwareentwicklung. Der Schwerpunkt liegt dabei auf generativen und modellbasierten Ansätzen. Bereits 1999 wurde eine damals auf der Programmiersprache Smalltalk und der **Unified Modeling Language (UML)** basierende Anwendungsplattform der Öffentlichkeit vorgestellt. Mit der seither stürmischen Fortentwicklung sämtlicher Basistechnologien wurden die zu Grunde liegenden Kernkonzepte ständig überprüft und weiterentwickelt.

Firmenprofil

Die JBnSE Service GmbH ist ein Tochterunternehmen der JBnSE GmbH. Im Firmenverbund ist sie Dienstleistungspartner für Projektkunden sowie Vertriebs- und Supportpartner für Softwarelösungen auf Basis der unternehmenseigenen modellbasierten Anwendungsplattform **JBnSE Works**.

Vortragsmanuskript

Einführung

In der gebotenen Kürze möchte ich mich kurz vorstellen. Name und Firma kennen Sie bereits aus der Vortragsankündigung. Hier geht es mir um die wesentlichen Hintergründe und Antriebe meiner Tätigkeit.

Ich bin zwar Gründer, Mehrheitsgesellschafter und Geschäftsführer der JBnSE GmbH, aber meine Leidenschaft und der überwiegende Teil meiner Arbeitszeit gehört der Technologie der Softwareentwicklung.

Stationen dabei waren die Arbeit in einschlägigen universitären wie industriellen Forschungsprojekten. Von Beginn an war ein wesentliches Ziel solcher Vorhaben die Erhöhung der Entwicklungsproduktivität. Mein Thema in einem industriellen Forschungsprojekt, gefördert vom damaligen Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF), war schon 1996 (also vor 27 Jahren) eine (objektorientierte) Softwarearchitektur zur automatisierten Erzeugung und Anbindung heterogener Datenhaltungssysteme.

Mit diesem Hintergrund erscheinen mir persönlich die Inhalte und Ziele hinter dem Begriff Low-Code nicht wirklich neu. Unstrittig neu und erfreulich ist die stark zunehmende Sichtbarkeit, Verbreitung und Akzeptanz, welche die Vielfalt der zugrunde liegenden technologischen Ansätze und die existierenden Produkte unter dieser einfachen und allgemein eingängigen Bezeichnung erfährt. Das ist ein willkommener Grund und Anlass, als Technologe einem guten Marketing meinen Dank auszusprechen.

Unser Fokus auf Technologie und Architektur im Kontrast zu primär wirtschaftlichen Effekten entspricht sehr stark meinen persönlichen Interessen und Triebkräften und ist damit in gewissem Sinne auch zufällig. Dennoch lade ich Sie auch mit dem heutigen Vortrag zu einem genaueren Blick auf die Technik ein. Dazu stelle ich eine These in den Raum und zur Diskussion, welche sich in meinen Erfahrungen bestätigt hat: „Kluge Architektur führt fast im Selbstlauf zu verbesserter Effizienz und auch Qualität“. Bei aller Unschärfe wird das etwas klarer beim Versuch der Umkehr dieser Aussage. Die Formulierung von Zielen wie hier Effizienz und Qualität enthält keine hinreichende Information über zielführende Wege, es kann also in dieser Richtung keinen Automatismus geben. Der kritisch verständige Zuhörer wird nun die Unschärfe des Adjektivs „klug“ feststellen. Aus diesem Dilemma gibt es ein Entrinnen. Klug ist ein Vorgehen und ein Ergebnis im guten Allgemeinverständnis dann, wenn eine stetige Überprüfung und gegebenenfalls Korrektur im Hinblick auf ein angestrebtes Ziel erfolgt.

Mit diesem gedanklichen Hintergrund entwickeln wir nicht zuerst vordergründig ein Low-Code Tool für andere, sondern unsere eigene Unternehmenssoftware unter Verwendung zweckmäßiger generativer, modellbasierter und interaktiver Techniken. Diese Techniken zahlen in ihrer Gesamtheit auf das Low-Code Manifest ein. Seit etwa zwei Jahren entwickeln wir auf dieser Basis individuelle Kundensysteme als Erweiterungs- und Anpassungslösungen. Neben unserem Musikklassenportal gibt es auf diese Weise zwei Lösungen für einen namhaften deutschen Logistikanbieter.

Für die Auswahl eines Low-Code Ansatzes bzw. eines konkreten Produkts halten wir folgende Betrachtung für wesentlich. Ein System kann auf zwei prinzipiell unterschiedliche Arten gesehen werden.

1. Als **Blackbox**: Ein kurzfristiges Ergebnis steht im Mittelpunkt des Interesses. Die Art und Weise der Erzielung dieses Ergebnisses ist von nachrangiger Bedeutung. Der Anwender übernimmt keine besondere Verantwortung für einen langfristig kontinuierlichen Einsatz des Systems. Beispiele: Prototyp, schnell verfügbare Pilotlösung.
2. Als **Whitebox**: Neben einem langfristig tragfähigen Ergebnis ist auch die Art und Weise der Ergebnisbereitstellung (die Technologie) von Bedeutung. Der Anwender möchte oder muss die Verantwortung für einen langfristig tragfähigen Einsatz des Systems übernehmen. Ein späterer Systemwechsel soll aus Aufwands- und Kostengründen vermieden werden. Beispiel: Produktentwicklung, kritische, umfangreiche und langlebige Unternehmenslösungen.

Wir fokussieren strategisch auf letztere Sichtweise und richten unsere Ansprache möglicher Interessenten, Kunden oder Kooperationspartner daran aus.

Gegenstand des weiteren Vortrags sind daher einige technologische Besonderheiten unseres Ansatzes, welche wir mit dem vorgenannten Ziel einer „klugen Architektur“ für interessant halten und welche für unsere eigenen konkreten Ziele ganz wesentlich sind. Hier zunächst eine Übersicht:

- **Business-Schicht** als Low-Code
- Nutzung der UML
- Template Engine
- Geräteunabhängigkeit
- Datenbankunabhängigkeit

Sofern Sie hier bestimmte Punkte vermissen, ist das möglicherweise der Begrenzung von Raum und Zeit innerhalb dieser Veranstaltung geschuldet oder es handelt sich um weniger spezifische Aspekte. Fragen Sie aber gerne nach.

Vertiefung

Business-Schicht als Low-Code

Tatsächlich arbeiten wir mit Programmobjekten in einer verbreiteten Programmiersprache, aktuell Java. Im Unterschied zur konventionellen Entwicklung gibt es jedoch markante Unterschiede:

1. Der Code wird weitestmöglich aus höherwertigen Informationen generiert.
2. Der Umfang des verwendeten Codes wird auf ein architektonisch sinnvolles Minimum reduziert. So wird im Zuge der fachlichen Anwendungsentwicklung grundsätzlich kein algorithmischer Code zur Erstellung von Benutzeroberflächen oder Datenanbindungen geschrieben.

Der Begriff Business-Schicht hat sich in der Terminologie der Softwareentwicklung für den fachlich relevanten Teil bzw. Kern einer Software etabliert, im Unterschied zur technisch notwendigen Infrastruktur. Die Begriffsbildung ist insofern etwas missverständlich und unglücklich, da es nicht auf den Bezug zu einem Geschäft im eigentlichen Wortsinn ankommt, sondern tatsächlich allgemein Fachlichkeit in Abgrenzung zur Technik gemeint ist.

Die Business-Schicht ist der wertvolle und stabile Kern einer Software. Das gilt relativ, denn obwohl auch das sogenannte Business einem beschleunigten Wandel unterliegt, ist die technologische Entwicklung stets schneller und oft disruptiv. Insofern gilt es, die Investitionen in das fachliche Verständnis von technologischen Veränderungen weitgehend abzuschirmen und so zu schützen.

Nutzung der UML

In Bezug auf die Rolle des Entwicklers/Programmierers kommt es letztlich auf das Fachwissen an, welches aus den Fachanforderungen in die Anwendungsfunktionen übertragen wird. Neben dem Informationsfluss zum Realisierungszeitpunkt einer Funktion ist dabei die dauerhafte Ablage, die Historienführung und die Kombinationsfähigkeit im Sinne von Wissensbausteinen von Bedeutung.

Wir setzen dazu weiter auf die **Unified Modeling Language (UML)** und dabei zuerst auf den schon lange relativ stabilen Kern der Klassendiagramme zur Beschreibung der grundlegenden logischen Datenstrukturen. Die UML findet hier ausdrückliche Erwähnung, da mit ihr der Rahmen einer Business-Schicht generiert wird, welche vor allem konsequent unabhängig von der angeschlossenen Datenhaltungstechnologie ist. D.h. konkret auch, dass Daten nicht zwangsweise in einer relationalen Datenbank abgelegt werden müssen.

Der Codegenerierungsprozess stellt sich schematisch wie folgt dar.

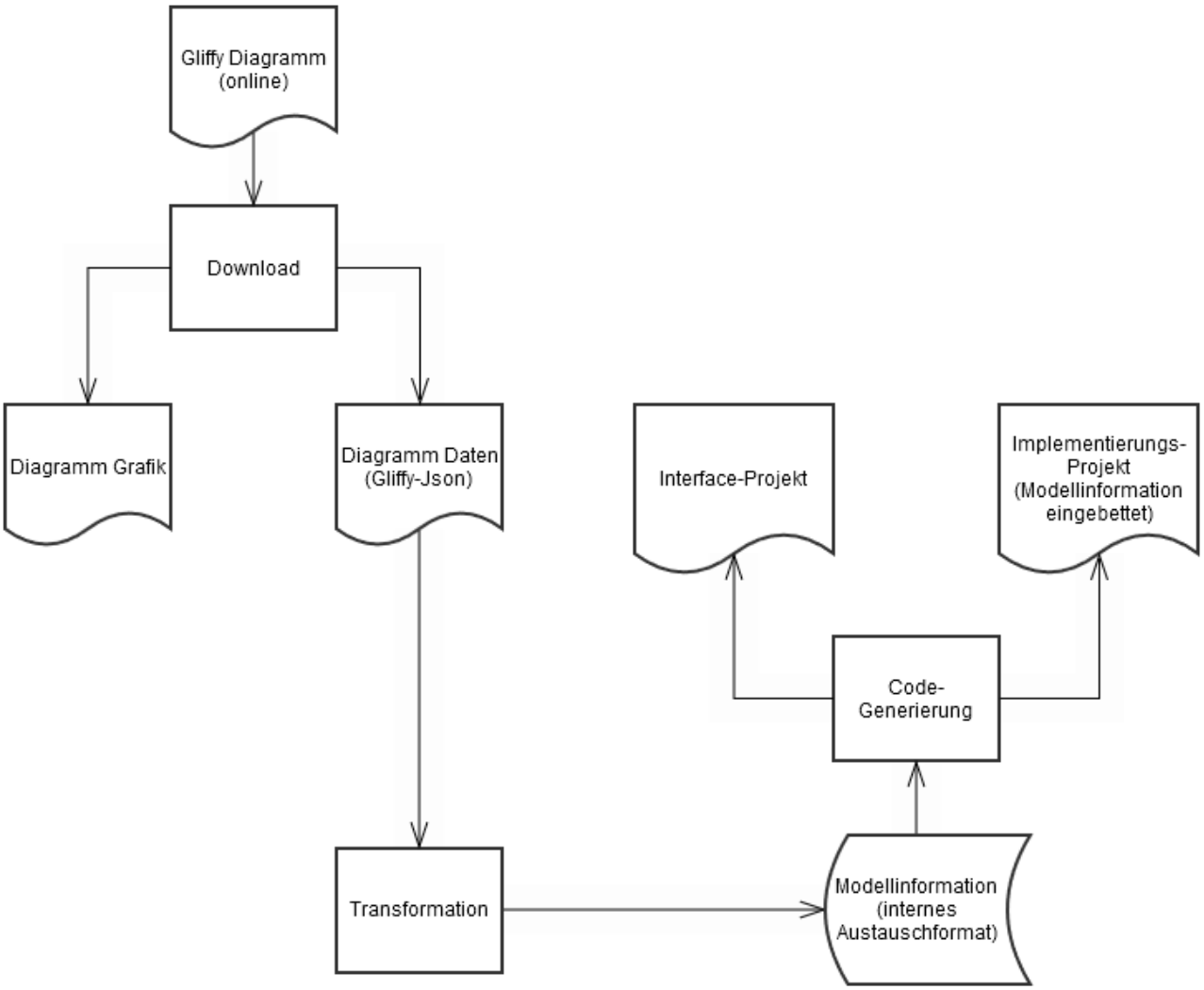

Abbildung 1. Codegenerierungsprozess

Durch die Generierung wird prinzipiell kein Code für Benutzeroberflächen oder zur Datenhaltung erzeugt. Diese Funktionen bleiben spezialisierten und austauschbaren Komponenten vorbehalten. Auf diese kommt der Vortrag im weiteren Verlauf zurück.

Zunächst betrachten wir die Struktur der generierten Programmbibliotheken. Charakteristisch ist hier die Einteilung in inhaltliche/fachliche Säulen (Module) und die konsequente Trennung von Schnittstelle und Implementierung in Schichten.

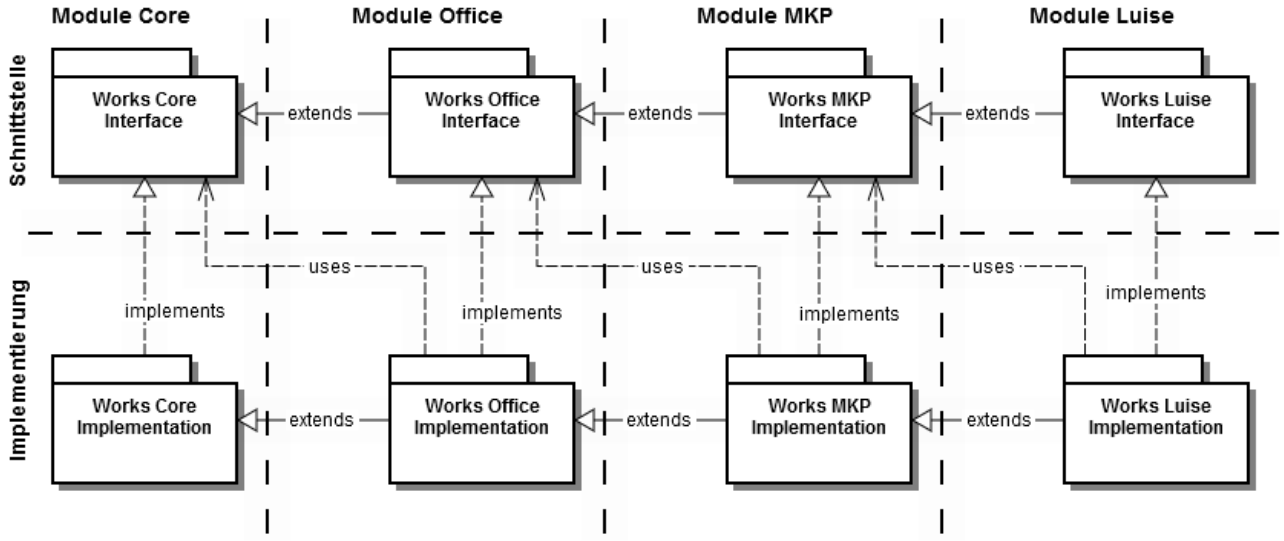

Abbildung 2. Modellbibliotheken in Modulen und Schichten

Bisher umgesetzte Individuallösungen auf Basis unserer Plattform setzen regelmäßig auf dem **Modul Office** auf, d.h. sie bestehen aus Erweiterungen und Modifikationen der vorhandenen Office-Funktionen. Exemplarisch dafür sind die beiden **Module MKP und Luise** in der Abbildung oben. MKP enthält dabei sämtliche Standardfunktionen unseres Musikklassenportals. Unter dem Arbeitstitel Luise existiert eine exklusive Erweiterung bzw. Anpassung für einen spezifischen Einsatzfall.

Die **Module Core und Office** haben eine Sonderstellung. Sie bilden die Schnittstelle zwischen der Laufzeitumgebung und den fachlichen Anwendungsmodulen. Eine grundlegende Besonderheit unseres Ansatzes ist, dass auch die Datenstrukturen des Laufzeitsystems über die UML modelliert und auf diesem Wege erweiterbar sind. Auch bezüglich der Anbindung von Benutzeroberflächen und Datenhaltung stehen für die Daten der Laufzeitumgebung, wie Modelldaten und Oberflächendefinitionen, die gleichen Techniken wie für die eigentlichen Anwendungs- oder Nutzdaten der Fachdomäne zur Verfügung.

Das Modul Core beschränkt sich auf die Kernaufgaben der Laufzeitumgebung. Exemplarisch zeigt das folgende Diagramm die aktuelle Modellierung des Typsystems dort.

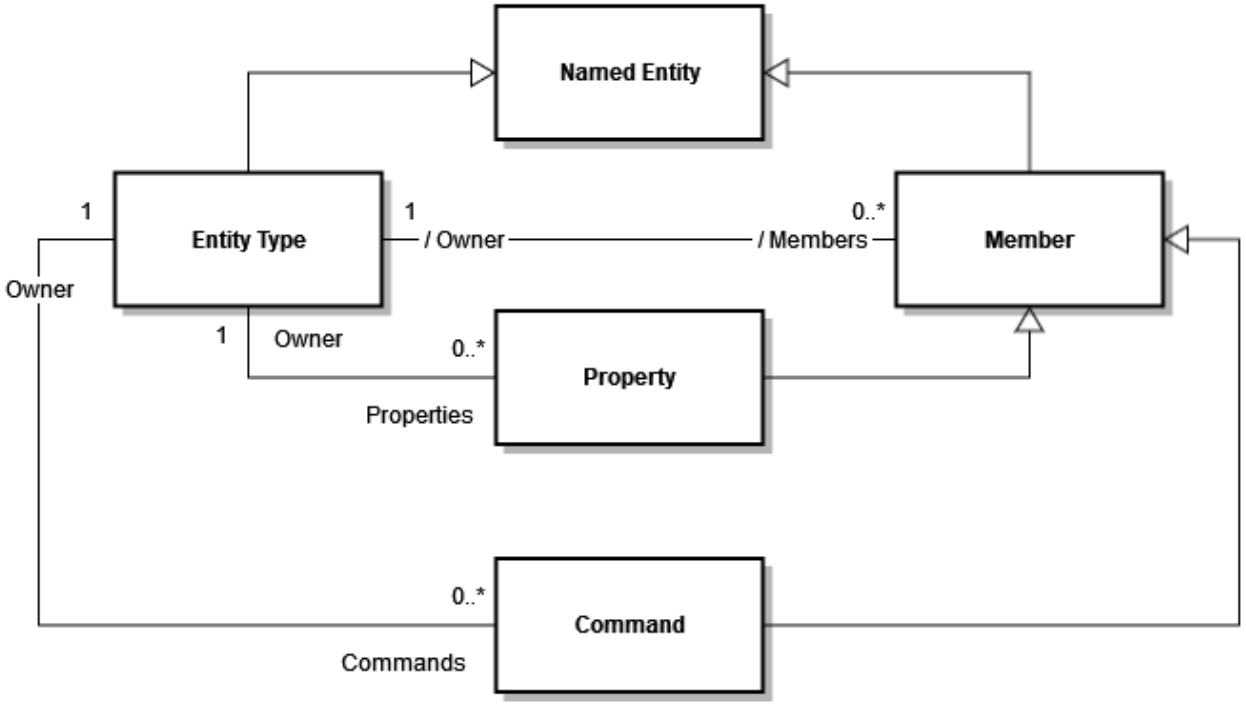

Abbildung 3. Modellierung des Typsystems

Ein weiterer wesentlicher Kernbestandteil ist die Grundstruktur der Benutzeroberfläche. Das Template-Modell legt die Grundlage für die Arbeit der Template-Engine zur Erzeugung zusammengesetzter komplexer Benutzeroberflächen. Besonders hier ist die einfache Erweiterbarkeit durch Modellierung wichtig, so können leicht weitere Parameter zur Verwendung in dem Typ- und Kontextbezogenen Auswahlverfahren ergänzt werden.

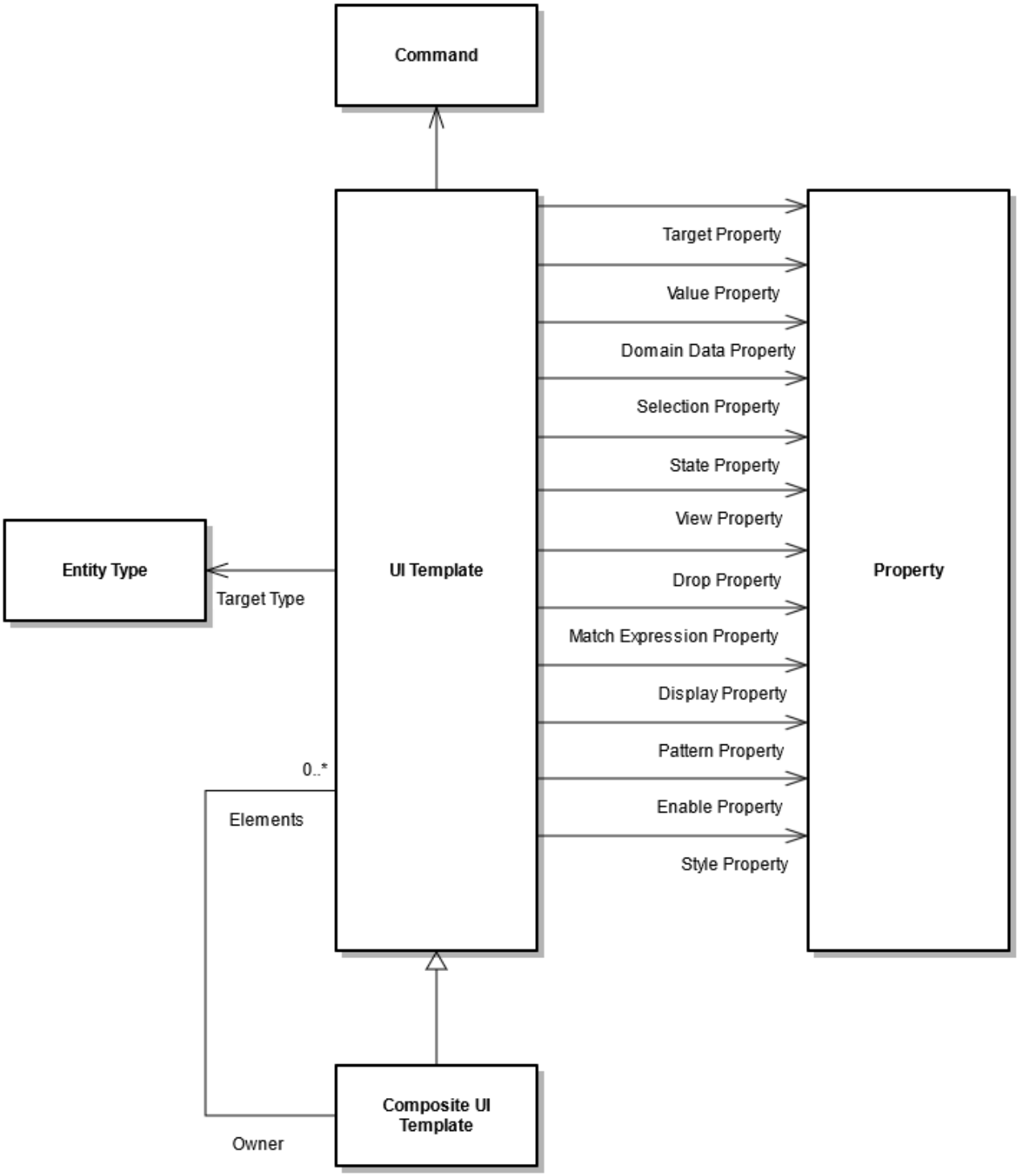

Abbildung 4. UI Templates

Das Modul Office bildet die Basis für allgemeine Verwaltungsanwendungen. Diese Anwendungsklasse stellt derzeit das Hauptanwendungsgebiet unserer Plattform dar. Das Modul enthält das Modell für komplexe, hierarchisch strukturierte Benutzerarbeitsplätze (Workspaces) wie sie aus verbreiteten Desktop-Anwendungen bekannt sind.

Wesentliche Elemente sind dabei eine hierarchische App- und Applet-Struktur zur Darstellung der beiden Hauptnavigationsebenen in der Anwendung.

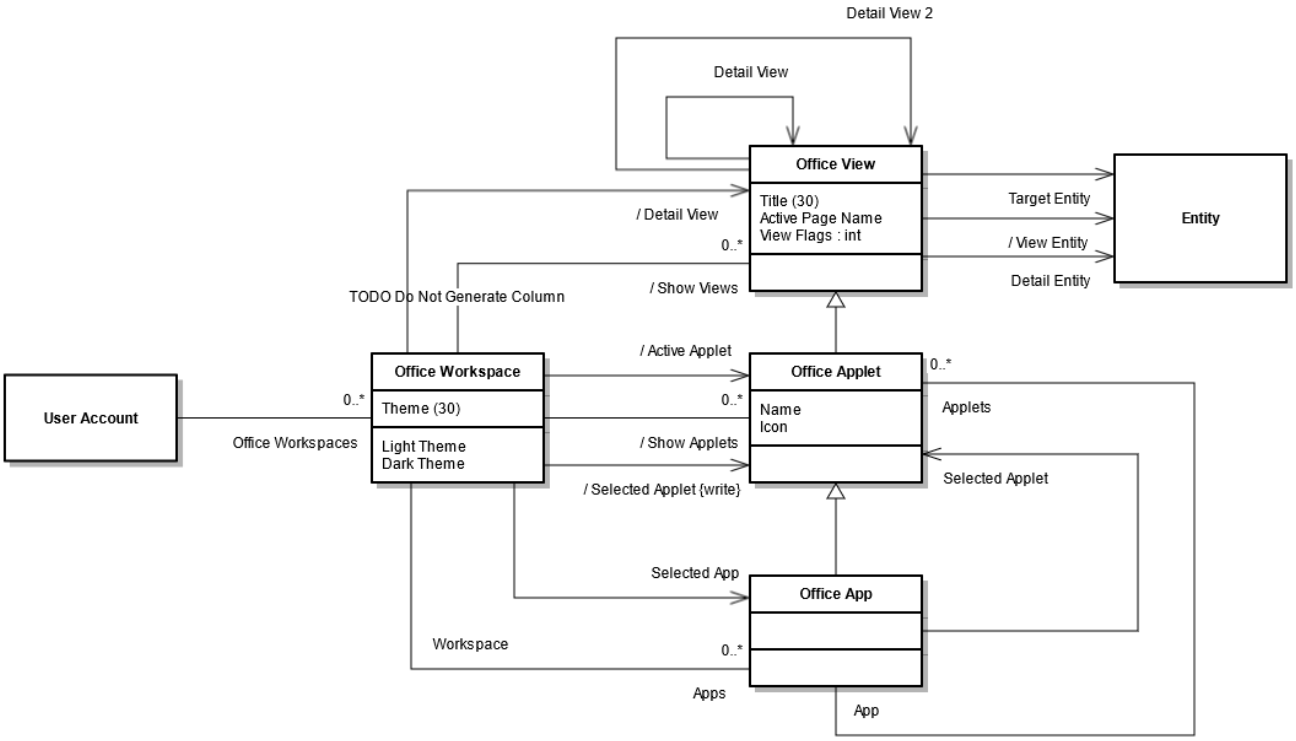

Abbildung 5. Office App und Office Applet

Hinzu kommt eine allgemeine Basis für strukturierte Such- und Selektionsverfahren zur Navigation in die eigentlichen Anwendungsdaten, sozusagen die Nutzlast der Anwendung.

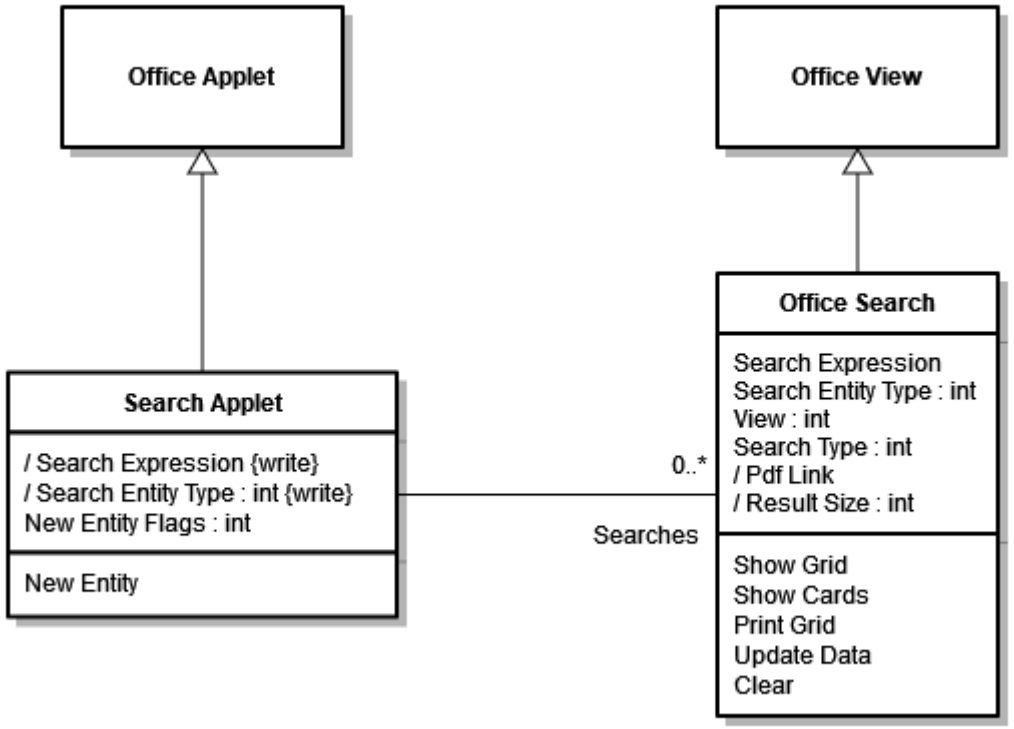

Abbildung 6. Search Applet, Office View, Office Search

Konkret wird es dann durch Ableitung spezifischer Strukturen wie nachfolgend exemplarisch gezeigt für die Suche und Bearbeitung von Asset (Inventar) und Instrument.

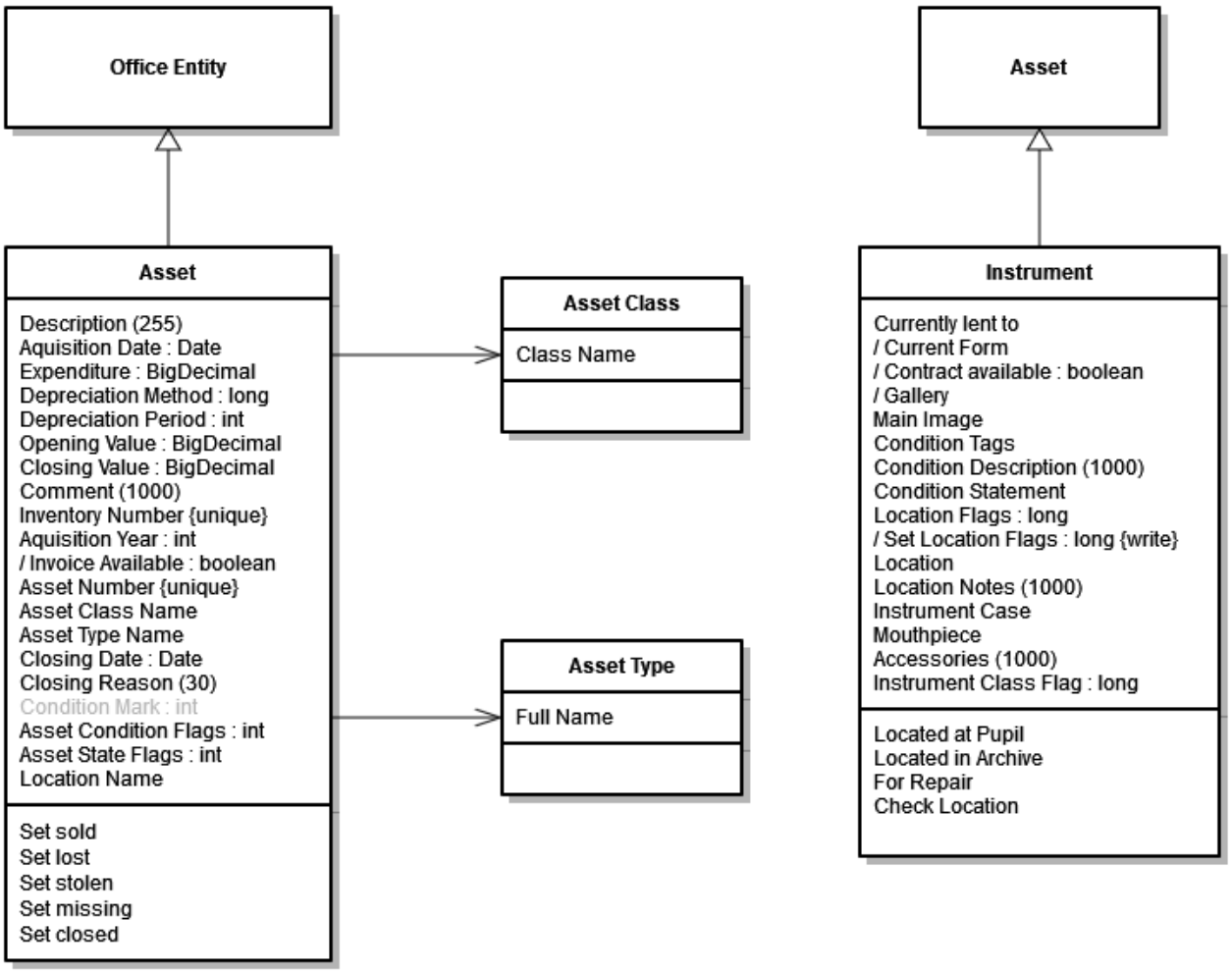

Abbildung 7. Office Entity → Asset → Instrument

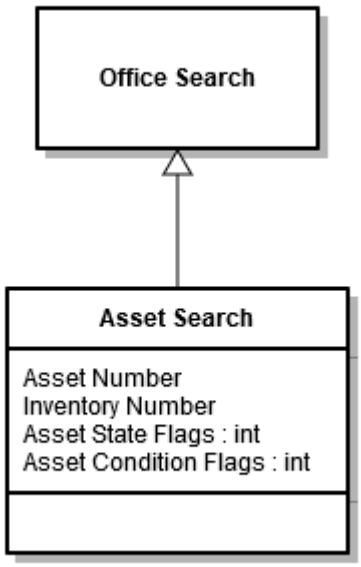

Abbildung 8. Office Searach → Asset Search

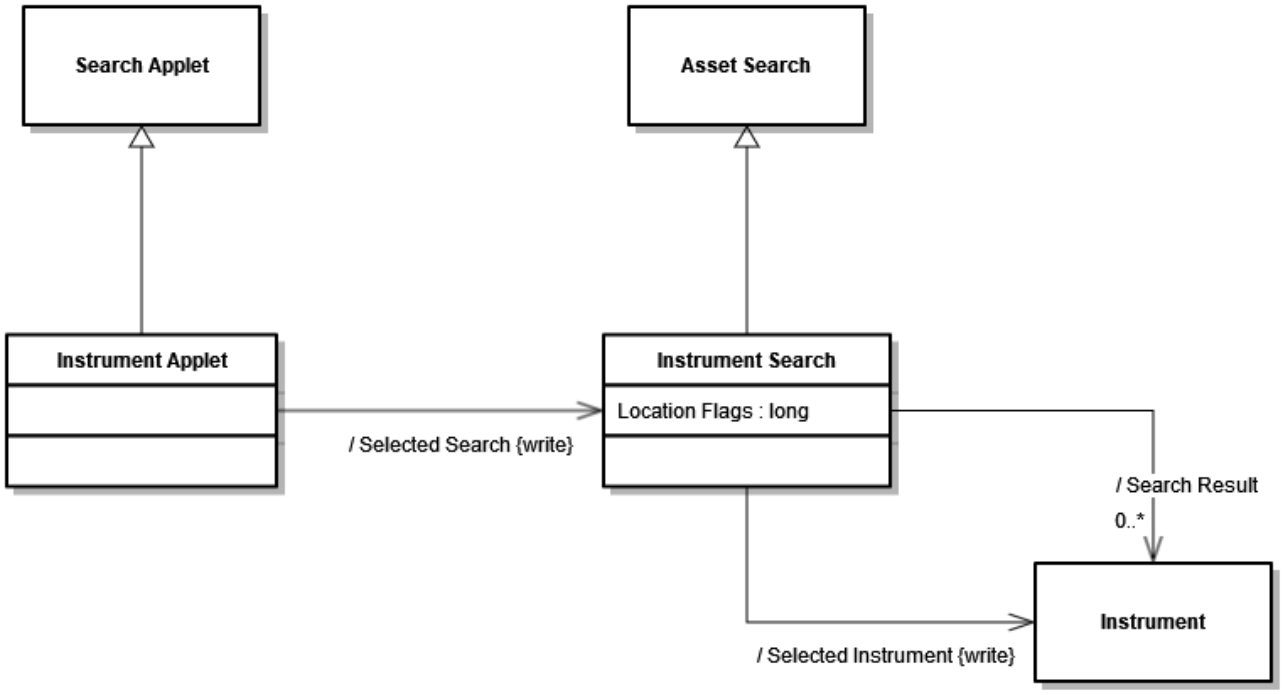

Abbildung 9. Instrument Applet, Asset Search → Instrument Search

Die gezeigten Diagramme sind nur ein kleiner Ausschnitt und natürlich nicht ausreichend zur Erlangung eines vollständigen Verständnisses. Es soll aber ein Eindruck gegeben werden, wie mit bewährten objektorientierten Verfahren und Techniken komplexe Anwendungsstrukturen exakt beschrieben und verständlich dokumentiert werden können.

Template Engine

Ein nächstes charakteristisches Merkmal unserer Plattform ist der Einsatz einer innovativen geräteunabhängigen Template Engine. Prinzipiell gibt es weder klassisch algorithmischen Code zur Erzeugung von Oberflächen und auch keine schon auf ein bestimmtes Ausgabeformat zugeschnittene Templates auf Basis von HTML oder XML. Die Vorlagen unserer Template Engine sind rein deklarativ und liegen in der Datenbank, ihre Struktur ist in UML modelliert und damit erweiterbar. Sie werden entweder automatisch erzeugt, interaktiv bearbeitet oder im Programmcode deklariert.

Die folgende Abbildung zeigt sehr vereinfacht das Funktionsprinzip unserer Template Engine. Obwohl das nicht auf den ersten Blick offensichtlich ist, bilden sowohl die Anwendungsdaten links als auch die fertige Oberfläche rechts eine Baumstruktur. Ausgangspunkt und Wurzelelement ist im Falle einer typischen Office-Anwendung ein Arbeitsplatzobjekt (Workspace). An den Enden der Baumstruktur finden sich die Blattelemente wie einzelne Eingabefelder oder Listenzellen. Dazwischen liegen mehrere Ebenen aus verschiedenen Struktur- und Layoutelementen. Die Template Engine läuft nun den Baum der Anwendungsdaten beginnend am Wurzelelement ab, sucht für jedes Element passende Templates zur Darstellung und bringt diese zur Anwendung. Die Templateauswahl erfolgt über einen Matching-Algorithmus unter Verwendung von Kriteriengruppen wie Objekt- oder Datentyp, Kontext, Status und Berechtigungen. So kann zum Beispiel die Position und Sichtbarkeit von Formularen oder einzelner Elemente in Abhängigkeit von Datentypen, Workflow-Status, Benutzerberechtigungen etc. gesteuert werden. Kriterien und Matching-Algorithmus sind bei Bedarf erweiterbar.

Durch Anwendung der Templates entsteht serverseitig ein Zwischenergebnis, das **Server DOM**. Obwohl der Begriff **DOM (Document Object Model)** tatsächlich aus der Begriffswelt des Browsers stammt, hat das **Server DOM** konzeptionell keinen Bezug zu HTML. Wie das **Browser DOM** bildet es jedoch auf der Serverseite die logische Struktur der Benutzeroberfläche ab. Diese Struktur wird beim initialen Oberflächenaufbau vollständig und im Laufe der späteren Benutzerinteraktionen inkrementell zum Client übertragen. Dort wird es im Falle eines Browser-Clients durch eine spezielle JavaScript-Bibliothek in das **Browser DOM** überführt.

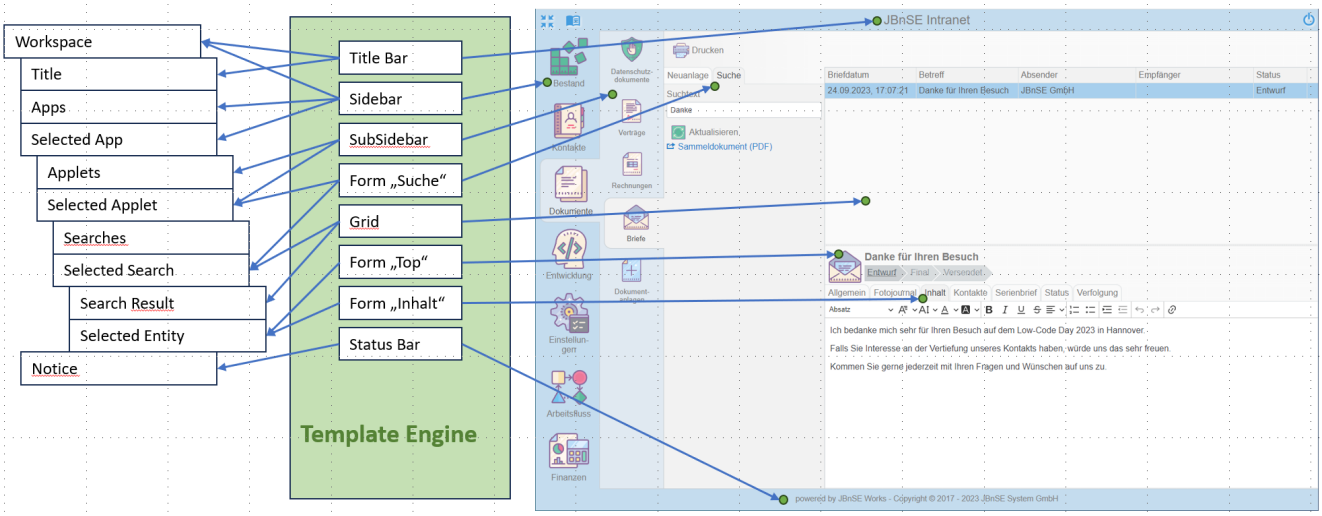

Abbildung 10. Anwendungsdaten → Template Engine → Server DOM → Browser DOM

Die konkreten Datenflüsse und Verarbeitungsschritte werden durch das folgende Diagramm veranschaulicht.

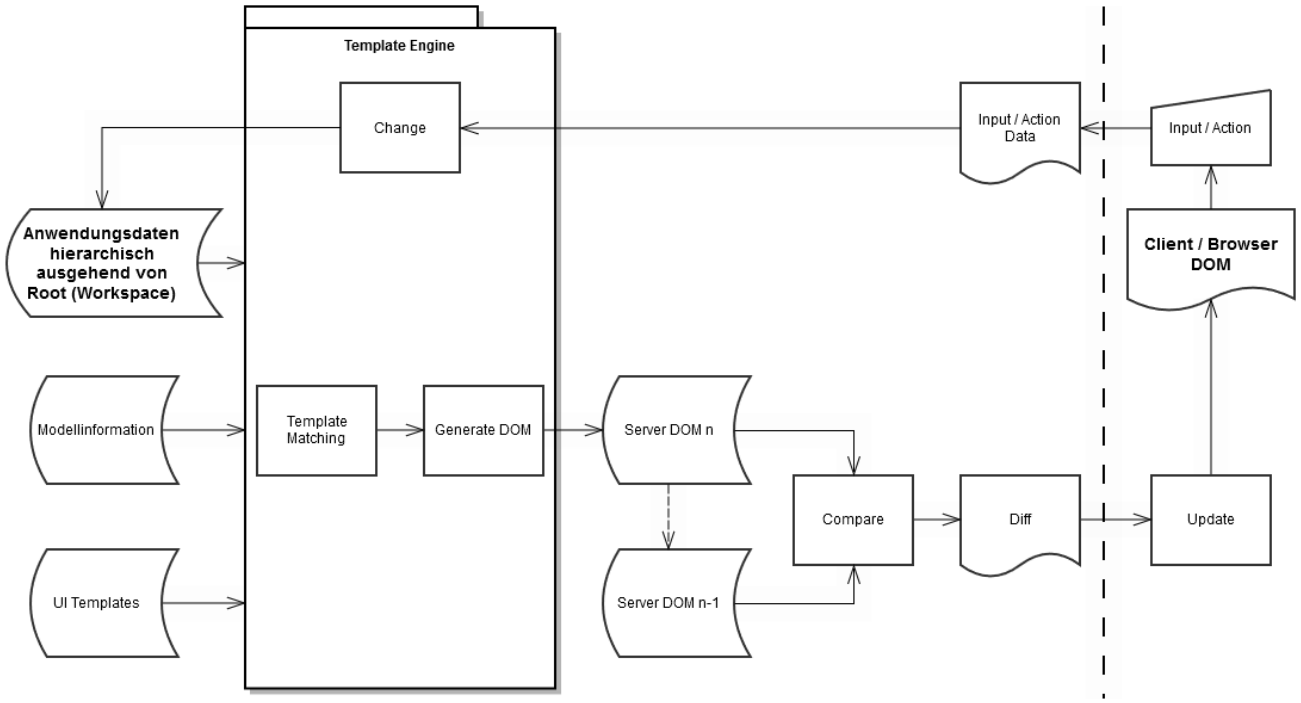

Abbildung 11. Datenfluss Template Engine

Geräteunabhängigkeit

Durch die Einführung eines universellen Server DOMs ergibt sich als nächstes wesentliches

Merkmal unserer Plattform die prinzipielle Unabhängigkeit von einzelnen Endgerätetechnologien. Das so eingeführte Kommunikationsprotokoll ist wie folgt definiert.

Die Kommunikation zwischen Server und dem **Thin Rich Client** erfolgt durch ein Client-unabhängiges Kommunikations- bzw. Interaktionsprotokoll. Bezüglich seiner wesentlichen Merkmale trägt es die Bezeichnung **Universal differential Interaction Protocol**, Abkürzung/Akronym **UdIP**. Die namensgebenden Merkmale sind:

Universal

Universell bzw. geräteunabhängig, d.h. das Protokoll unterstützt z.B. Browser-Clients gleichermaßen wie native Clients z.B. auf Basis von **JavaFX**.

differential

Es werden jeweils nur minimale Änderungen der Benutzeroberfläche vom Server zum Client übertragen. Die Änderungen werden grundsätzlich automatisch **ohne Zutun des Anwendungs-Entwicklers** ermittelt.

Interaction

Das Protokoll deckt alle Interaktionen zwischen Oberflächen-Client und Server ab. Der Server liefert initiale bzw. geänderte Inhalte als Antwort auf Benutzeraktionen und -eingaben.

Die in den folgenden beispielhaften Abbildungen gezeigten Oberflächen unserer hauseigenen Unternehmenssoftware zeigen zwei technisch vollkommen unterschiedliche Oberflächen. Zunächst unseren ursprünglichen auf JavaFX basierenden Client. Bereits diesen haben wir zur später parallel bzw. nachfolgend vorgesehenen Implementierung als Browser-Client auf unser geräteunabhängiges Protokoll gestützt.

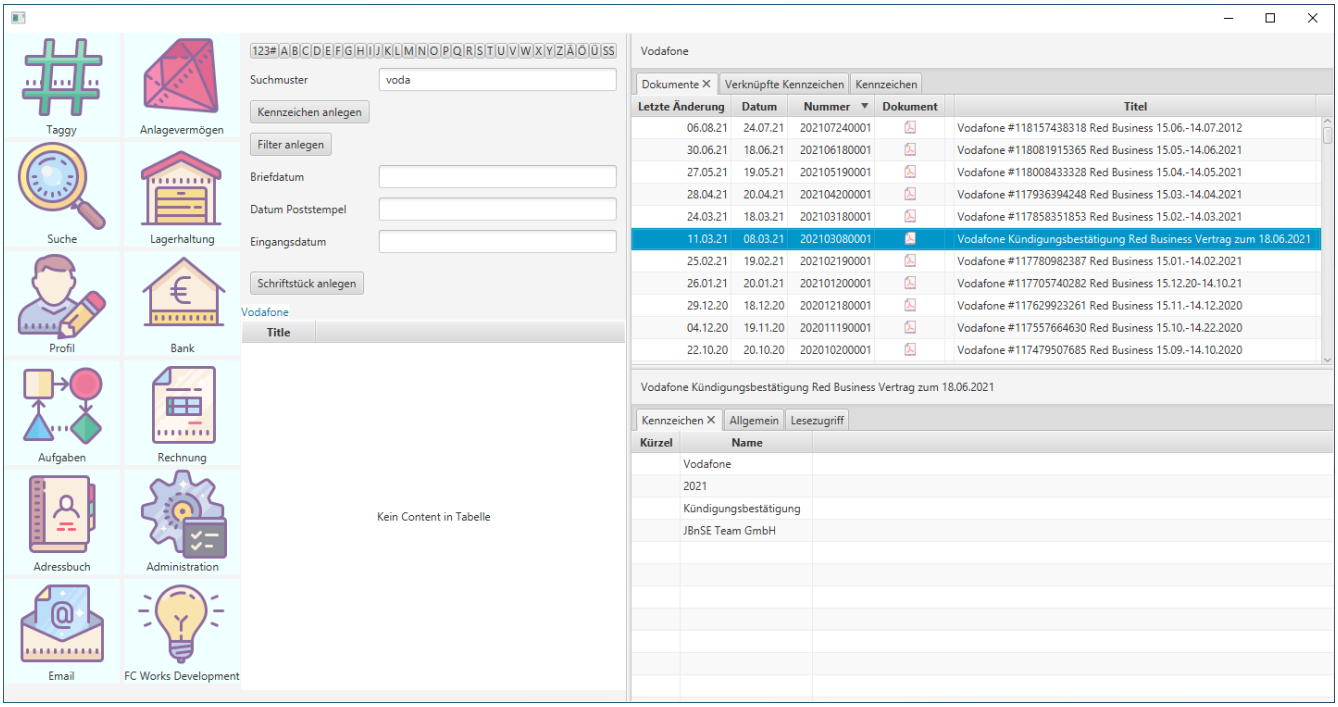

Abbildung 12. JavaFX-Client bis zur Abkündigung von JavaFX durch den Hersteller

Nachfolgend unseren heutigen Browser-Client basierend auf HTML5, CSS und JavaScript. Dieser Client enthält einen Protokoll-Monitor, den wir hier zur Veranschaulichung der differenziellen Funktionsweise des Protokolls eingeblendet haben. Es gibt zwei prinzipiell unterschiedliche

Aktionstypen als Antwort des Servers auf eine Benutzerinteraktion. Die folgende Abbildung zeigt eine **strukturelle Veränderung** des DOMs bzw. der Oberfläche, hier das Hinzufügen eines aktiven Tab-Reiters mit weiteren Inhalten.

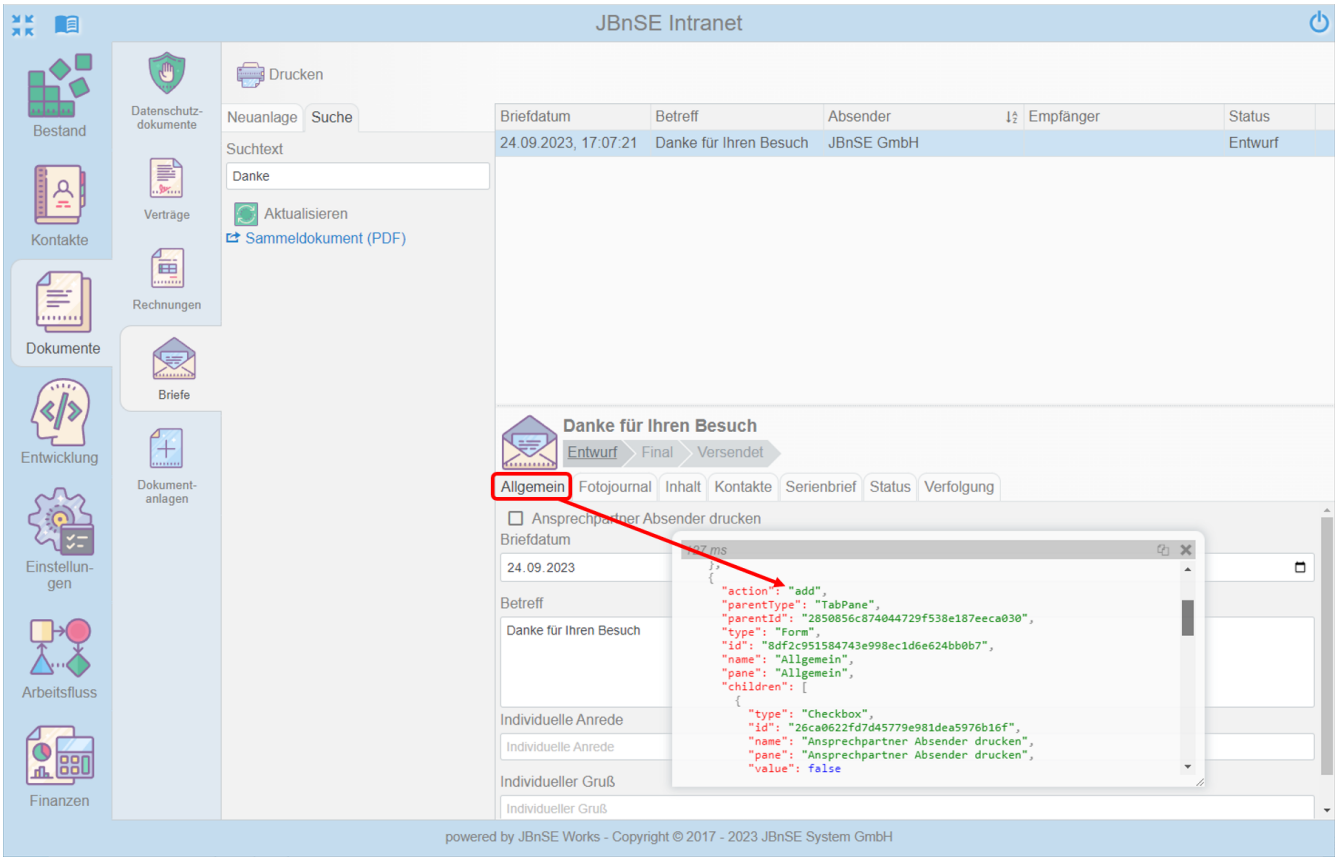

Abbildung 13. Protokollmonitor: Auswahl Tab-Reiter "Allgemein"

Die nächste Abbildung zeigt eine einfachere **inhaltliche Änderung** an zwei betroffenen Stellen nach einer Benutzereingabe.

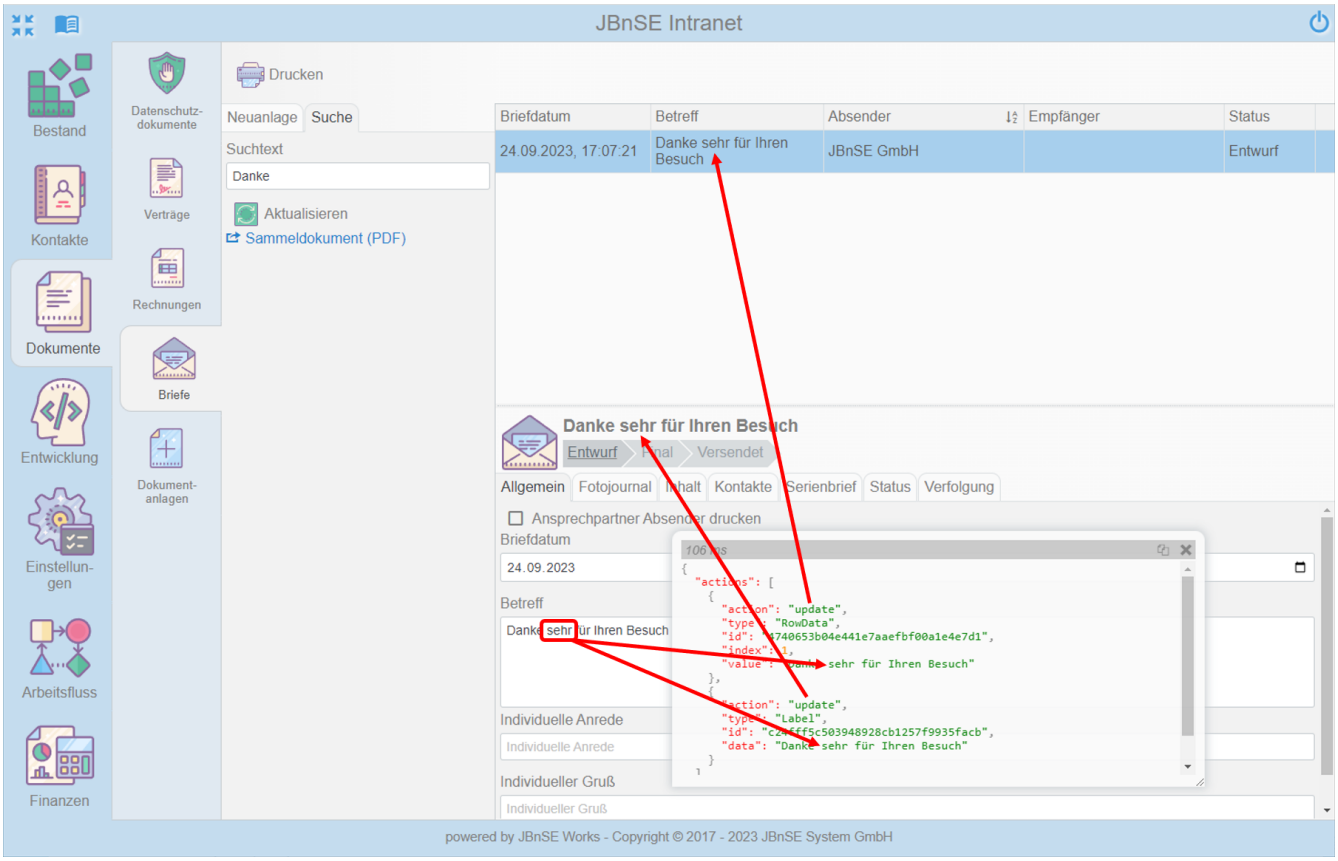

Abbildung 14. Protokollmonitor: Eingabe "sehr" im Feld Betreff

In Konsequenz des Zusammenspiels von Template Engine und Kommunikationsprotokoll entfällt jede manuelle Programmiertätigkeit zur Aktualisierung der Benutzeroberfläche bei beliebigen auch feingranular verteilten Änderungen von Anwendungsdaten. Unabhängig vom Umfang der Änderungen erfolgt die Aktualisierung im Regelfall innerhalb einer einzigen Transaktion. Damit sinkt als Zusatzeffekt die erforderliche Kommunikationsbandbreite und die Gesamtantwortzeit auf ein Minimum.

Datenbankunabhängigkeit

Nicht zuletzt bereitet die Generierung von unabhängigen Business-Objekten (Business-Schicht) aus UML und die umfassende Einbettung der Modellinformationen den Weg zur konsequenten Datenbankunabhängigkeit. Die folgende Abbildung zeigt die Beziehungen der beteiligten Pakete. Die Schnittstelle zwischen Business-Schicht und Datenhaltung befindet sich vollständig im Paket **Works Storage Interface**. Es wird deutlich, dass weder ein fachliches Anwendungsmodul eine Abhängigkeit zur Datenhaltung hat, noch umgekehrt eine bestimmte Datenhaltungskomponente eine Abhängigkeit zu einem Fachmodul. Die gesamte Funktionalität der Datenhaltung ist in den jeweiligen Storage-Komponenten grundsätzlich ausschließlich durch Bezug auf das Works Storage Interface und dort speziell auf die eingebettete Modellinformation implementiert.

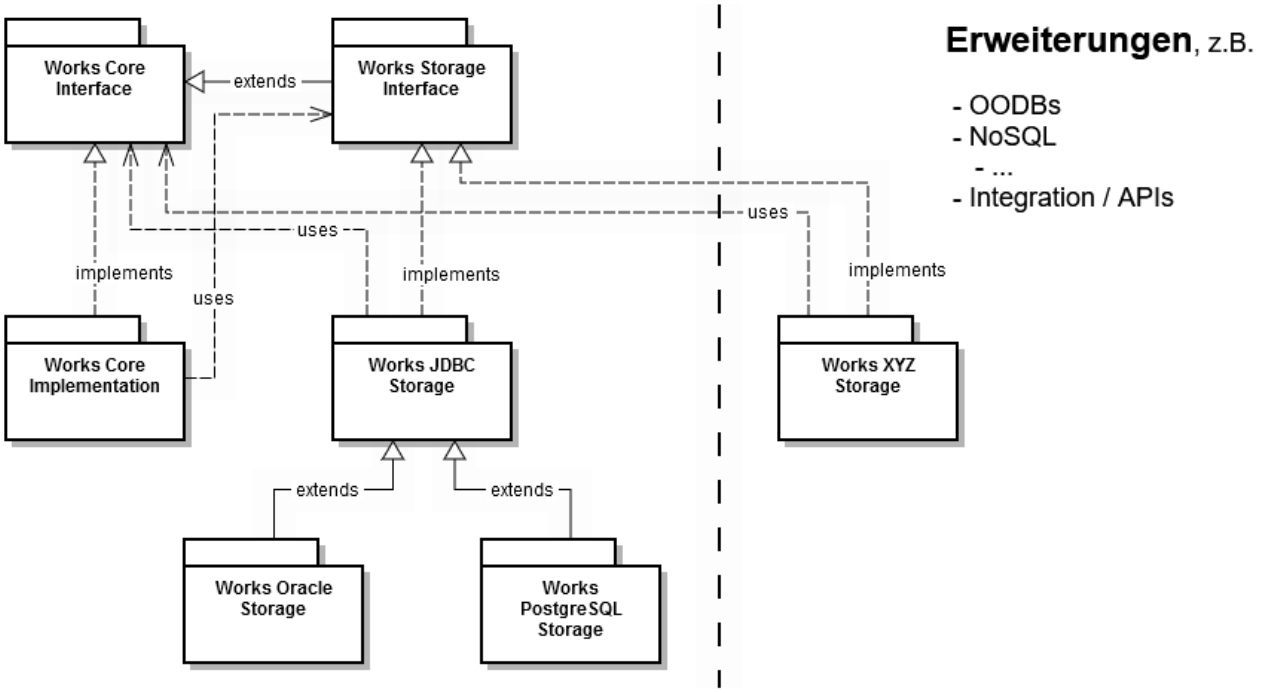

Abbildung 15. Storage Services

Mit der beschriebenen Datenbankunabhängigkeit schließt sich auch der Kreis einer langfristig tragfähigen flexiblen Anwendungsarchitektur. Die Datenbankunabhängigkeit war tatsächlich das erste Ziel eines modellbasierten generativen Ansatzes. Der Einsatz dieser Techniken für die programmierfreie Erzeugung der Benutzeroberfläche kam erst zu einem späteren Zeitpunkt hinzu, ist allerdings nicht weniger bedeutsam und wirkungsvoll.

Anknüpfend an das eingangs verwendete Adjektiv „klug“ für eine Anwendungsarchitektur hat sich langfristig gezeigt, dass klug vor allem bedeutet: flexibel. **Flexibel und robust** gegenüber erwartbaren technologischen Änderungen, Brüchen oder paralleler Existenz und Integrationsfähigkeit disjunkter Technologien vor allem im Bereich von Datenhaltung und Benutzeroberfläche.

Insofern empfehlen wir unsere Plattform als Option für Anwender mit **langfristiger Verantwortung für Kontinuität und Investitionssicherheit**.