6. Konzeption

Simplicity is the ultimate sophistication.

(Leonardo da Vinci)

Inhaltsangabe

In diesem Kapitel wird die Konzeptionsphase dieser Arbeit behandelt. Zunächst wird auf das Vorgehensmodell für die Entwicklung des Konzepts und die Implementierung des Prototyps eingegangen. Darauf folgt eine Auflistung der identifizierten Stakeholder und ihren Interessen. Im nächsten Abschnitt wird der entwickelte Papierprototyp detailliert erläutert und etwaige Designentscheidungen begründet. Im Rahmen eines Ziele-Workshops in Zusammenarbeit mit GDIS[1] wurden die identifizierten Interessen neu gewichtet und letzte Anpassungen am Papierprototyp vorgenommen. Es folgt eine Anforderungsliste für das gesuchte Modell und den zu implementierenden Webservice für Rechenkerne. Schließlich wird das entwickelte Variabilitätsmodell anhand eines Klassendiagramms vorgestellt und die allgemeine Funktionsweise des Webservice mit Hilfe des EMI-Schichtenmodells erklärt.

6.1. Vorgehensmodell

Für das Vorgehen in dieser Arbeit wurde der Ansatz der iterativen Software-Entwicklung gewählt [31]. In diesem zeichnet sich eine Iteration durch die Tätigkeiten *Analysieren*, *Entwerfen*, *Codieren* und *Testen* aus. Tatsächlich wurde in dieser Arbeit jedoch etwas kleinschrittiger vorgegangen. Am Anfang stand die Identifizierung der Stakeholder und ihrer Interessen, sowie die Entwicklung eines ersten Prototyps.

[1] Generali Deutschland Informatik Services GmbH

Anschließend wurden die Interessen sukzessive verfeinert und der Papierprototyp in mehreren Iterationen Stück für Stück angepasst. Parallel dazu wurden verwandte Arbeiten studiert und die Umsetzung von Variabilität in ähnlichen Systemen untersucht.

Als die Architektur der *Enterprise Measurement Infrastructure* feststand, welche anfangs parallel zu dieser Arbeit entwickelt wurde, konnten konkrete Anforderungen an den Webservice formuliert werden. Auf Basis dessen wurde ein Variabilitätsmodell konzipiert und ein Prozess erarbeitet, der den Austausch der spezifizierten Variabilität zwischen Rechenkern und Dashboard via Webservice erlaubt.

Während der anschließenden Implementierungsphase, wurden kleinere Änderungen am Variabilitätsmodell vorgenommen, dessen Notwendigkeit in der Konzeptionsphase nicht bedacht wurden oder nicht vorhergesehen werden konnten.

6.2. Stakeholder und ihre Interessen

Nach der Einarbeitungsphase und Erlangung des entsprechenden Fachwissens über die Funktionsweise und Architektur von MeDIC und MeDIC-Dashboard, wurden die Stakeholder und ihre Interessen ermittelt.

Projektleiter
Ein Projektleiter ist der typische Nutzer von MeDIC-Dashboard. Das Dashboard unterstützt ihn insbesondere bei der Kontrolle des Softwareentwicklungsprozesses, indem Metriken zu Zustand und Fortschritt des jeweiligen Softwareprojekts visualisiert werden.

- Es wird eine Möglichkeit gewünscht die Dashboard-Items zu modifizieren (einzustellen).
- Die Einstellungsmöglichkeiten sollen sowohl die zugrundeliegende Metrik und deren Berechnung als auch die grafische Darstellung betreffen.
  Kommentar: Zum Beispiel die Umstellung von Balken- auf Liniendiagramm.
- Es soll möglich sein, die Einstellungen der Dashboard-Items eines Dashboard-Containers einfach und intuitiv vornehmen zu können.
- Dazu soll die Dashboard-Ansicht nicht verlassen werden müssen.
  Kommentar: Möglich zum Beispiel via Popover[2] oder Modal-Dialog[3].
- Der Nutzer muss sich einen Überblick über alle veränderlichen Attribute der Dashboard-Items verschaffen können.
- Zusätzlich soll es Standardwerte für die Attribute geben.

[2] http://twitter.github.io/bootstrap/javascript.html#popovers
[3] http://twitter.github.io/bootstrap/javascript.html#modals

- Falls es Standardwerte für die veränderlichen Attribute gibt, soll es möglich sein die Dashboard-Items auf diese Werte zurückzusetzen.
- Eine Hilfefunktion soll weitere erklärende Informationen zu den Attributen liefern und möglicherweise Beispielausprägungen anzeigen.
- Darüber hinaus wäre eine Art Plausibilitätsprüfung der eingestellten Parameter hilfreich.
  Kommentar: Liefert die Metrik mit den eingestellten Parametern überhaupt Messdaten?

Dashboard-Administrator (Metrik-Experte)

Ein Metrik-Experte besitzt die Rolle des Administrators in MeDIC und interagiert nicht direkt mit der Dashboard-Oberfläche. Seine Aufgabe ist die Überwachung, Verwaltung und Erweiterung der Funktionalität des Dashboards. Insbesondere geht es darum Erkenntnisse über die Verwendung von Metriken innerhalb eines Unternehmens zu gewinnen.

- Wieviele unterschiedliche Benutzer (Projektleiter) melden sich pro Woche/Monat am Dashboard an?
- Anzahl aktiver Benutzer mit laufendem Projekt vs. Anzahl inaktiver Nutzer mit laufendem Projekt.
- Übersicht über alle Projekt-Dashboards und wie stark diese modifiziert wurden (Top/Worst 5).
  Kommentar: Der Grad der Modifikation eines Dashboards könnte z.B. mit der Abweichung zur Dashboard-Vorlage und den Standardeinstellungen der Dashboard-Items gemessen werden (mehr dazu in Abschnitt 6.3.2).
- Eine Art Rangliste/Tabelle von Dashboard-Vorlagen in Benutzung mit Angabe, wie häufig diese angepasst wurden und zusätzlich wie stark diese durchschnittlich (Median) modifiziert wurden.
- Rangliste der am häufigsten und wenigsten genutzten Dashboard-Items (Metriken).
- Rangliste meist modifizierter Dashboard-Items (Metriken).

Entwickler

Zu den Aufgaben eines Entwicklers zählt die Implementierung von Interaktionsoberflächen, Visualisierungen, Rechenkernen (Metriken) und Datenadaptern zur Integration heterogener Systeme.

- Es muss auf einfache Weise möglich sein, Rechenkerne und Dashboard-Items um Variabilität zu erweitern oder Varianten hinzuzufügen.
- Dabei soll eine Lösung gefunden werden, die es *nicht* erfordert für jede Metrik-Variation einen neuen Rechenkern zu implementieren.

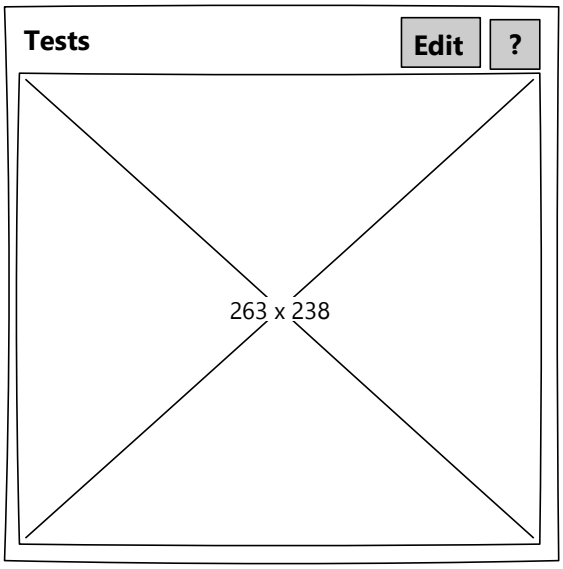

Abbildung 6.1.: Dashboard-Container in MeDIC-Dashboard

6.3. Papierprototyp

Damit die Integration von Variabilität in MeDIC-Dashboard gelingen kann, muss der strukturelle Aufbau eines Dashboards verstanden werden (siehe dazu auch Abschnitt 3.2). Darauf aufbauend wurde über mehrere Iterationen hinweg ein Papierprototyp entwickelt, der die Interessen der Stakeholder berücksichtigt. So konnten die Interessen auf explorative Weise weiter präzisiert werden. Die Anforderungsermittlung für ein konsistentes Variabilitätsmodell, dass in die Systemlandschaft von MeDIC passt, folgt in Abschnitt 6.4.

Ein Dashboard besteht aus einer Menge von sogenannten Dashboard-Containern, die für die Darstellung von Diagrammen verantwortlich sind. Die Anordnung der Dashboard-Container innerhalb des Dashboards erfolgt nach dem Layout-Schema einer Tabelle. Die genaue Position kann durch die Angabe der Zeile und Spalte bestimmt werden. Abbildung 6.1 zeigt den groben Aufbau eines Dashboard-Containers.

In der Titelleiste befindet sich oben links der Name des Dashboard-Containers, welcher einen Hinweis auf die visualisierten Metriken geben soll. Oben rechts ist eine Schaltfläche mit einem „?“-Symbol zu sehen. Diese zeigt beim Überfahren mit der Maus Informationsbedürfnisse in Form von Fragen an, die von den verknüpften Metriken erfüllt werden. Dadurch erhält der Nutzer Aufschluss darüber, welchen Zweck die visualisierten Metriken erfüllen.

Der wichtigste Bereich wird durch einen Platzhalter in Form von einem großen „X“ dargestellt. In diesem Bereich befinden sich die Dashboard-Items. Jedes Dashboard-Item steht für ein Diagramm. Momentan gibt es drei verschiedene Typen:

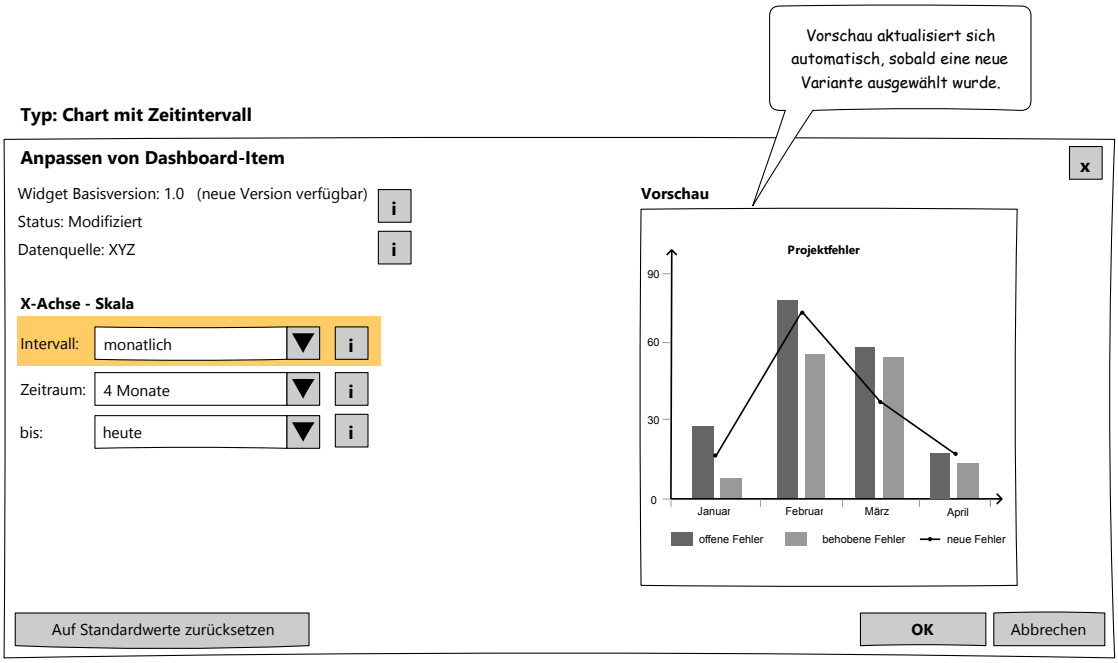

Abbildung 6.2.: Papierprototyp – Einstellungsdialog für `CartesianChart`

1. `CartesianChart`
   Hiermit wird die Darstellung von Balken- und Liniendiagrammen realisiert.
2. `BulletGraph`
3. `Table`

Die Anordnung der Dashboard-Items erfolgt, wie schon bei den Dashboard-Containern, nach dem Layout-Schema einer Tabelle. So sieht es das zugrundeliegende Datenmodell vor. Die Klasse `DashboardItemContainer` enthält eine Liste von Listen von Dashboard-Items. Dies bringt sowohl eine hohe Flexibilität als auch eine hohe Komplexität mit sich. In der Praxis gibt es momentan jedoch nur drei Konstellationen. Entweder der Dashboard-Container enthält nur ein Dashboard-Item (vom Typ `CartesianChart` oder `Table`) oder er enthält eine Liste von `BulletGraph`-Objekten, die vertikal untereinander angeordnet sind (siehe Abbildung 3.1).

Die „Edit“-Schaltfläche oben rechts in Abbildung 6.1 ist ein neues Steuerelement dieses Prototyps und in MeDIC noch nicht vorhanden. Die Idee ist, dass über diese Schaltfläche ein Einstellungsdialog geöffnet werden kann, der die Anpassung der verknüpften Metriken erlaubt.

6.3.1. Einstellungsdialog

Abbildung 6.2 zeigt den Entwurf eines Einstellungsdialogs für einen Dashboard-Container mit `CartesianChart`. Der Dialog ist zweigeteilt. Links befinden sich Eingabefelder, um Einstellungen vorzunehmen. Rechts ist eine Diagramm-Vorschau, die sich automatisch aktualisiert, sobald eine neue Variante ausgewählt wurde. Das Beispiel zeigt drei Dropdown-Listenfelder zur Konfiguration der veränderlichen Attribute „Intervall“, „Zeitraum“ und „bis“. Sie sollen dazu dienen, die Auflösung der Messreihe des dazugehörigen Balken- oder Liniendiagramms einzustellen. Rechts neben den Dropdown-Listenfeldern befinden sich kleine quadratische Schaltflächen mit einem ï"(wie Information), die auf Wunsch weitere Informationen zu dem entsprechenden Attribut aufzeigen sollen. Möglich wären Erklärungen zur Bedeutung des entsprechenden Attributs, Standardwerte, Beispielausprägungen oder Gültigkeitsbereiche. Neben den in den Abbildung 6.2 aufgeführten Dropdown-Listenfeldern, sind außerdem andere Eingabefelder für verschiedene Typen von Attributen denkbar. Dazu zählen Checkbox-Gruppen zur Mehrfachauswahl und Textfelder (vergleiche Abschnitt 6.5 Variabilitätsmodell).

Eine frühere Iteration des Prototyps hat außerdem die Einstellung des Minimums und des Maximums der Y-Achse des Diagramms vorgesehen. Diese Idee wurde allerdings verworfen, da Graphen null-basiert sein sollten, um die Darstellung für quantitative Daten nicht zu verzerren und zu verfälschen [16]. Das Maximum kann automatisch anhand des höchsten Messwertes bestimmt werden.

Des Weiteren wurde über ein *Widget*-Konzept erarbeitet. Alle Einstellungen für einen Dashboard-Container sollen in Form von Widgets vorgegeben werden können. Ein *Widget* ist ein Dashboard-Container mit einer vordefinierten Menge von enthaltenen Dashboard-Items (Diagrammen) mit bestimmter Reihenfolge und Voreinstellungen. Widgets können von Metrik-Experten vorkonfiguriert werden und sind mit den Informationsbedürfnissen verknüpft, die sie befriedigen.

Dennoch soll der Projektleiter[4] (Benutzer) die Möglichkeit haben, für eine bestimmte Auswahl an Attributen Einstellungen vorzunehmen. Um die vom Benutzer nachträglich modifizierten Attribute kenntlich zu machen, wird eine farbliche Hinterlegung des jeweiligen Eingabefelds vorgeschlagen. Außerdem steht im Anzeigefeld *Status* im oberen linken Teil der Abbildung der aktuelle Zustand des Widgets. Mögliche Werte sind „Modifiziert“ oder „Voreinstellung“. Über die Schaltfläche „Auf Standardwerte zurücksetzen“ unten links, kann das Widget in den Ausgangszustand zurückversetzt werden.

Weiterhin ist es möglicherweise sinnvoll eine Versionierung der Widgets einzuführen. Wenn sich Unternehmensziele ändern, kann dies dazu führen, dass Metriken im Rahmen ihrer definierten Variabilität angepasst werden müssen.

[4] Die Begriffe Benutzer und Projektleiter werden im folgenden synonym verwendet.

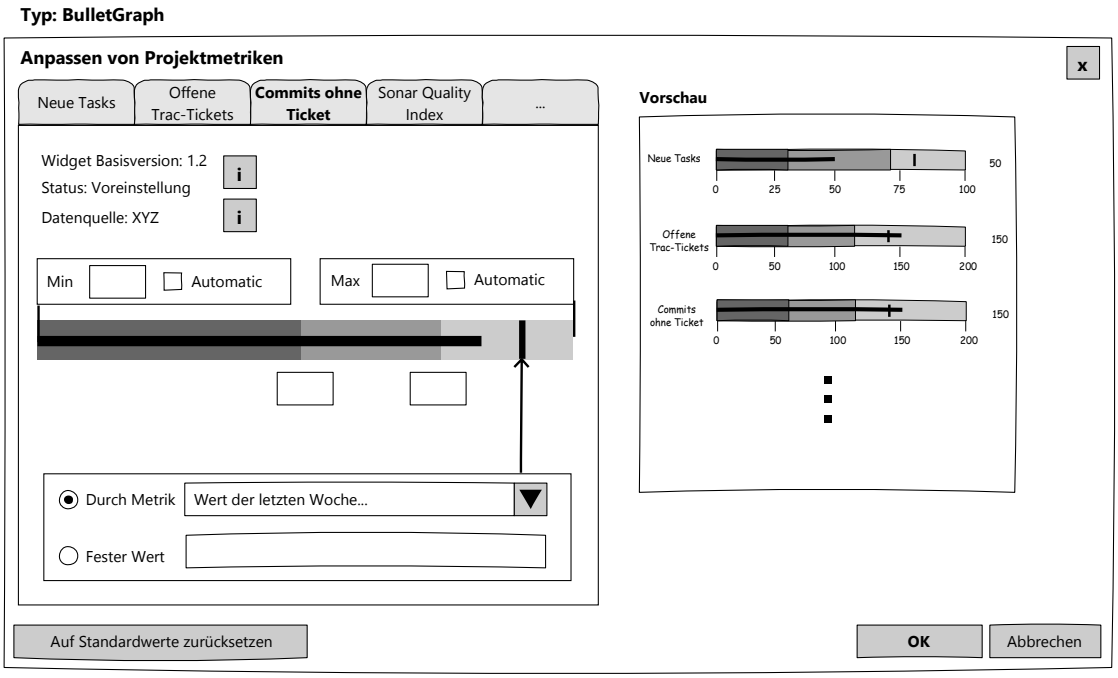

Abbildung 6.3.: Papierprototyp – Einstellungsdialog für eine Liste von `BulletGraph`-Objekten

Ein weiteres interessantes Detail für den Benutzer ist das Anzeigefeld *Datenquelle*. Es soll sichtbar machen welches System die Daten zur Berechnung der Metrik bereitstellt (z.B. Sonar[5], Hudson[6] oder Trac[7]).

Schließlich ist es möglich, die Einstellungen mit „Ok“ zu übernehmen oder mit „Abbrechen“ bzw. dem Fenster-Schließen-Symbol oben rechts den Vorgang abzubrechen, so dass die Änderungen nicht angewendet werden. In beiden Fällen wird der Einstellungsdialog geschlossen.

Abbildung 6.3 zeigt einen spezialisierten Dialog für eine Liste von `BulletGraph`-Elementen. Genau wie in Abbildung 6.2 gibt es eine Zweiteilung in Eingabefelder links und Diagramm-Vorschau rechts. Das Layout der Eingabefelder ist genau auf die Eigenschaften eines `BulletGraph`-Diagramms zugeschnitten. Es können die sogenannten Qualitätsintervalle und der Vergleichswert definiert werden. Der Vergleichswert kann entweder durch eine Metrik oder wahlweise auch durch einen festen Wert spezifiziert werden. Der Metrik-Experte soll die Möglichkeit haben bestimmte Attribute für die Anpassung durch den Benutzer zu sperren. Die dazugehörigen Eingabefelder sollen in diesem Fall ausgegraut werden.

[5]Sonar ist ein Entwicklungswerkzeug zu Berechnung von Code-Metriken.
[6]Hudson ist eine Software für die kontinuierliche Integration.
[7]Trac ist ein Projektmanagement-Werkzeug mit Bugtracking-System und Wiki.

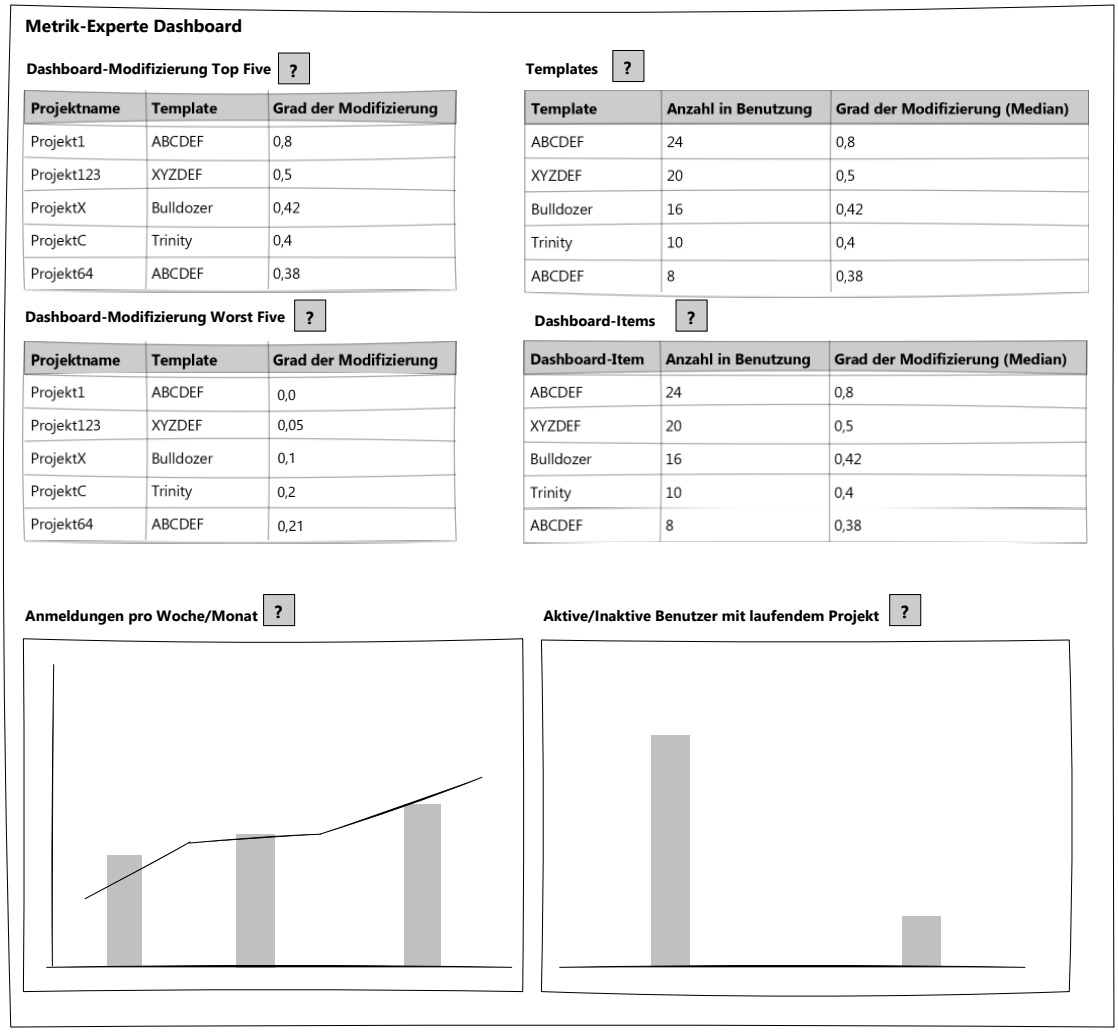

Abbildung 6.4.: Papierprototyp – Erste Iteration des Metrik-Experten-Dashboards

Da es mehrere `BulletGraph`-Elemente pro Dashboard-Container gibt, muss es möglich sein, alle in einem Einstellungsdialog modifizieren zu können. Zur Lösung dieses Problems wird die Navigation über Karteireiter oberhalb der Eingabefelder vorgeschlagen. Die Diagramm-Vorschau auf der rechten Seite bezieht sich jedoch auf alle enthaltenen `BulletGraph`-Elemente. Alle weiteren Funktionen entsprechen denen aus dem ersten Beispieldialog aus Abbildung 6.2.

6.3.2. Erste Iteration des Metrik-Experten-Dashboards

Abbildung 6.4 zeigt die erste Iteration des Dashboards für Metrik-Experten – informell auch „Dashboard für das Dashboard“ genannt. Es soll einen Überblick über die

Nutzeraktivität und das Nutzungsverhalten von Dashboard-Vorlagen (Templates) und Dashboard-Items vermitteln. Eine zentrale Metrik in dieser Ansicht ist der *Grad der Modifizierung*, sowohl für Dashboards im Allgemeinen als auch für einzelne Dashboard-Items.

Der Grad der Modifizierung eines Dashboard-Items ist eine Zahl zwischen 0 und 1, geht also von 0% bis 100%. Betrachtet man alle Attribute eines Dashboard-Items, die als veränderlich gekennzeichnet sind und einen Standardwert besitzen, ergibt sich folgende Formel, um den Grad der Modifizierung zu messen:

$$\frac{\text{Summe der Attribute, die ungleich ihrem Standardwert sind}}{\text{Summe aller Attribute}}$$

Dieses Berechnungsverfahren lässt sich rekursiv fortsetzen, zur Berechnung des Modifizierungsgrads eines Dashboard-Containers und eines gesamten Dashboards.

Die Tabelle oben links zeigt eine Tabelle der Projektdashboards, die am stärksten von der gewählten Dashboard-Vorlage abweichen. Stark modifizierte Dashboards könnten ein Indiz dafür sein, dass für den jeweiligen Projekttyp keine passende Dashboard-Vorlage existiert.

Die Tabelle direkt darunter soll im Gegenteil dazu die Projekte anzeigen, die am geringsten von der gewählten Dashboard-Vorlage abweichen. So könnten Projekte identifiziert werden, für die höchstwahrscheinlich gute Vorlagen existieren. Umgekehrt betrachtet könnte zudem ermittelt werden, ob die Vorlagen allgemein gut angenommen werden oder durchgehend angepasst werden. Ein Problem mit dieser Darstellung ist jedoch, dass Projekte mit geringem Modifizierungsgrad für diesen Wert eine hohe Dichte haben könnten. Das heißt, es könnte sein, dass viele Projektdashboards gleichermaßen gering oder gar nicht angepasst werden. Das würde dazu führen, dass nur Projekte mit einem Modifizierungsgrad von 0 in dieser Tabelle erscheinen. Aufgrund dessen wurde die Tabelle in einer späteren Iteration verworfen.

In der rechten oberen Hälfte ist eine Tabelle zu sehen, die anzeigt, welche Dashboard-Vorlagen im Einsatz sind, wie häufig diese in Benutzung sind und wie hoch der Grad der Modifizierung im Mittel (Median) ist. Die Tabelle darunter zeigt die gleiche Information für Dashboard-Items an.

Zusätzlich dazu sind in der unteren Hälfte zwei Diagramme zur allgemeinen Nutzungsstatistik der Plattform zu sehen. Links sieht man schemenhaft ein Balken- und Liniendiagramm, welches die Anmeldungen von Dashboard-Benutzern über einen Zeitraum von mehreren Wochen oder Monaten anzeigt. So lässt sich ein allgemeiner Trend bestimmen, ob die MeDIC-Plattform gut angenommen wird.

Das Diagramm rechts davon soll Aufschluss darüber geben, ob Projektdashboards für laufende Projekte auch benutzt werden. Dazu werden alle Benutzer mit laufendem Projekt betrachtet. Der linke Balken steht für aktive Benutzer mit laufendem Projekt und der rechte Balken steht für inaktive Benutzer mit laufendem Projekt. Die Einteilung,

wann ein Benutzer aktiv ist und wann inaktiv, muss noch weiter untersucht werden. Eine Möglichkeit wäre es, alle Benutzer, die sich seit sieben Tagen nicht mehr eingeloggt haben als inaktiv zu bewerten.

Über die Fragezeichen-Schaltflächen sollen Erklärungen und Hilfestellungen zur Interpretation der Tabellen und Diagramme abrufbar sein.

6.3.3. GDIS Ziele-Workshop und angepasstes Experten-Dashboard

Nachdem die ersten Iterationen der Papierprototypen erarbeitet wurden und eine grundlegende Idee dafür geschaffen wurde, wohin die Entwicklung führen soll, wurde ein Ziele-Workshop mit einem Vertreter der Abteilung für Softwarequalitätsmanagement von *Generali Deutschland Informatik Services GmbH* (GDIS) arrangiert. Dort soll MeDIC in Zukunft das Projektmanagement unterstützen.

Im Mittelpunkt steht der erfolgreiche Metrik-Einsatz bei GDIS. Damit sollen im Endeffekt Projekte erfolgreicher werden. Gleichzeitig soll der Aufwand für den Metrik-Einsatz gering sein.

Die Ermittlung des Modifizierungsgrad spielt dabei eine untergeordnete Rolle. Wichtiger ist eine Analyse des Nutzungsverhaltens von MeDIC. Eine starke Nutzung der Dashboardfunktionen könnte ein Indiz für den erfolreichen Metrik-Einsatz im Unternehmen sein.

Aufgrund der neuen Erkenntnisse und den präzisierten Zielen wurde der Papierprototyp des Metrik-Experten-Dashboards überarbeitet. Abbildung 6.5 zeigt den neuen Entwurf. Im oberen Abschnitt befindet sich ein sogenannter *Activity Feed*, der die Anmeldungen unterschiedlicher Nutzer über die vergangenen Tage oder Wochen in einem zeitlichen Verlauf anzeigt.

Links darunter befindet sich ein Balkendiagramm welches, über einen zeitlichen Verlauf hinweg, die Nutzung der Report-Export-Funktion zeigt. Betrachtet werden alle aktiven Projektdashboards. Falls für jedes aktive Projektdashboard innerhalb des vergangenen Monats ein Report (Bericht) exportiert wurde, zeigt der Balken 100% an. Die Höhe des Balkens wird aus dem folgendem Quotienten berechnet:

$$\frac{\text{Summe der aktiven Dashboards mit genutzter Export-Funktion im vergangenen Monat}}{\text{Summe aller aktiven Projektdashboards}} \tag{6.1}$$

Ein Dashboard ist aktiv, wenn es zu einem laufenden Projekt gehört.

Rechts daneben befindet sich ein Diagramm zur Visualisierung aktiver und inaktiver Benutzer, welches es schon in der ersten Iteration gab.

Unten links zeigt die Abbildung ein Diagramm über Nutzungsverteilung der verschiedenen Datenquellen (externe Systeme) an. Der dunklere obere Balken zeigt

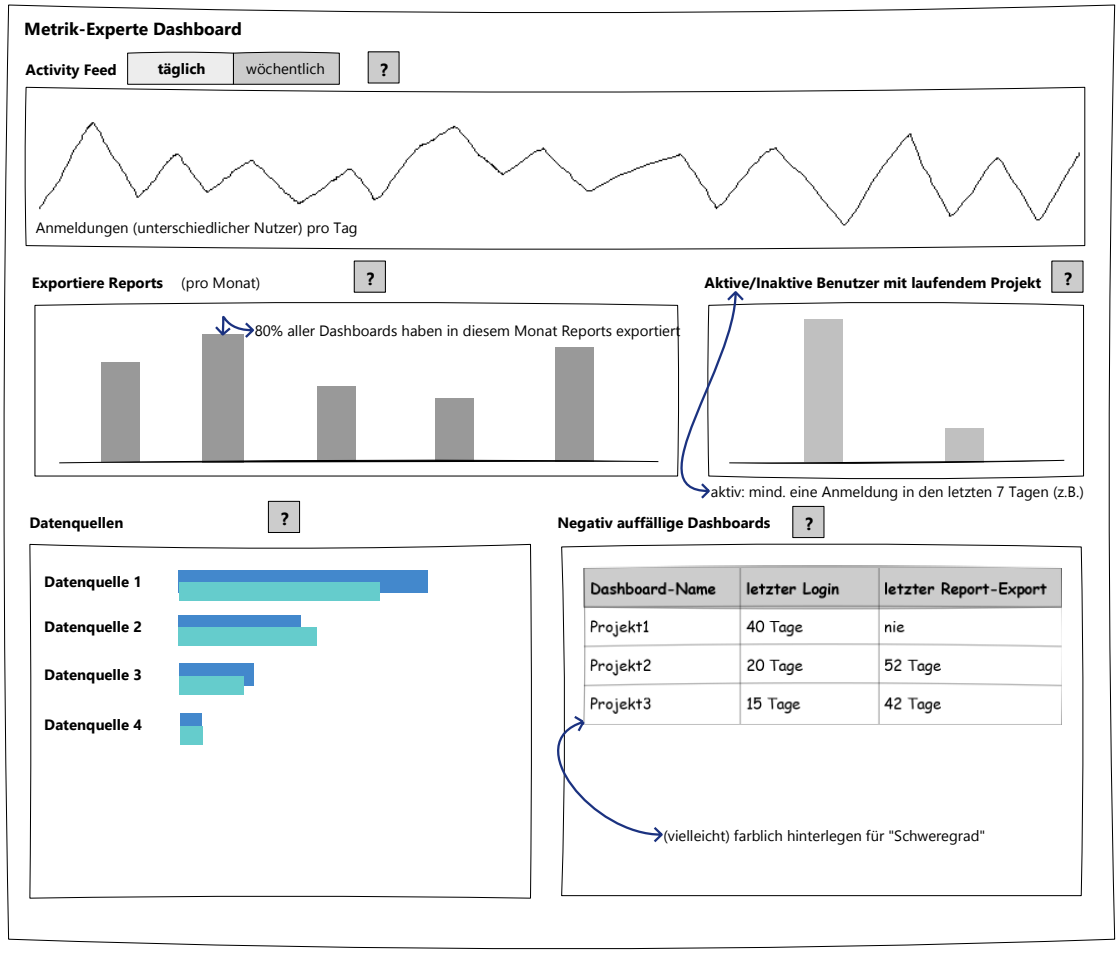

Abbildung 6.5.: Papierprototyp – Letzte Iteration des Metrik-Experten-Dashboards

jeweils den aktuellen Messwert an. Wohingegen der untere Balken den Vergleichswert des Vormonats anzeigt. Eine sinnvolle Berechnungsmethode der Balkenlängen muss noch entwickelt werden.

Die Tabelle unten rechts zeigt negativ auffällige Dashboards. Das sind Dashboards mit laufenden Projekten und wenig Nutzeraktivität, das heißt, wenige Anmeldungen und eine schlechte bzw. seltene Nutzung der Report-Export-Funktion.

6.4. Anforderungen an das Modell

Die Rechenkerne bzw. Metrikmodelle der Enterprise Measurement Infrastructure (EMI) spielen eine zentrale Rolle in der Messarchitektur von MeDIC. Die fachliche Aufgabe des Rechenkerns ist Repräsentation einer Metrik. An dieser Stelle ist der Typ der

Eingabedaten definiert und die Berechnungsvorschrift implementiert. Die Eingabedaten werden aus dem Strom der Metriknachrichten des EMDB-Buses anhand des Attributs `metricRefId` der EMDB-Nachricht herausgefiltert. Der Rechenkern ist jedoch keinesfalls als vollwertige Spezifikation einer Metrik[8] zu sehen. Diese umfasst neben den genannten Eigenschaften nämlich auch Interpretationshilfen und gegebenenfalls eine Spezifikation für die Visualisierung. Beide Aufgaben werden in EMI von der Visualisierungsschicht übernommen, also dem MeDIC-Dashboard.

Wie bereits im Abschnitt 3.4 beschrieben, ist es erforderlich, das Konzept der Rechenkerne um die Möglichkeit der variablen Konfiguration zu erweitern.
Im Folgenden werden die wichtigsten funktionalen und nicht-funktionalen Anforderungen an das Modell der Rechenkerne aufgelistet.

- Für das Modell der Rechenkerne muss eine Webservice-Schnittstelle definiert werden über die die Konsumenten der Visualisierungsschicht (vgl. Abschnitt 3.3 Messarchitektur) projektspezifische Messdaten abrufen können.
- Im engeren Sinne muss die Schnittstelle es ermöglichen, einzelne Messwerte abzurufen.
- Ein Messwert soll vom Typ `DataValueDB` sein, da dieser bereits in MeDIC-Dashboard etabliert ist.

  Kommentar: Ein Objekt vom Typ `DataValueDB` enthält zwei Attribute. Das erste Attribut steht für den eigentlichen Messwert, welches entweder vom Typ `Double` oder vom Typ `String` ist. Ein weiteres Attribut vom Typ `String` ordnet dem Messwert optional eine Kategorie zu.
- Beim Abruf eines einzelnen Messwertes soll standardmäßig die neueste bzw. aktuellste Messung zurückgegeben werden.
- Es soll jedoch auch möglich sein, Messwerte zu beliebigen Zeitpunkten der Vergangenheit abzurufen.

  Kommentar: Das ist nützlich für Vergleichswerte (siehe `BulletGraph`).
- Die Schnittstelle muss es ermöglichen, eine Messreihe abzurufen (Liste von Messwerten).
- Eine Messreihe ist eine einfache Liste von Messwerten vom Typ `DataValueDB`.
- Die Messwerte einer Messreihe sind chronologisch in aufsteigender Reihenfolge sortiert.
- Der Abruf einer Messreihe erfordert die Angabe eines Zeitintervalls (z.B. wöchentlich, monatlich etc.).
- Beim Abruf einer Messreihe soll es möglich sein, die Anzahl der zurückgegebenen

[8] Für eine vollständige Definition und Beschreibung des Begriffs siehe Ludewig und Lichter [31].

Messwerte festzulegen.

- Falls beim Abruf einer Messreihe die Anzahl der zurückgegebenen Messwerte nicht festgelegt wurde, sollen standardmäßig vier Messwerte zurückgegeben werden.
- Darüber hinaus muss die Schnittstelle ein Konzept für Variabilität beinhalten, die es ermöglicht, eine variablen Anzahl an Einstellungsparametern an den Rechenkern zu übergeben.
- Da jede Rechenkern-Implementierung andere Einstellungen zur Verfügung stellt, muss es eine einfache Möglichkeit geben, der Visualisierungsschicht die Art und den Umfang der unterstützten Variabilität mitzuteilen.

  Welche veränderlichen Attribute gibt es? Welchen Datentyp haben diese? Gibt es Vorbelegungen für die Attribute?
- Schließlich muss es dem Entwickler auf einfache und intuitive Art möglich sein, einen Rechenkern um Variabilität zu erweitern oder Varianten hinzuzufügen.

6.5. Variabilitätsmodell

Zur Beschreibung der Variabilität eines Rechenkerns wurde ein Variabilitätsmodell entwickelt, das sich stark an dem orthogonalen Variabilitätsmodell aus Abschnitt 4.3 orientiert.

Das Klassendiagramm des implementierten Modells ist in Abbildung 6.6 zu sehen. Es enthält bereits einige implementierungsspezifische Details, die während der Realisierungsphase hinzugekommen sind und die Deserialisierung betreffen, welche für die Übertragung wichtig ist (siehe Abschnitt 7.1).

Die Hauptbestandteile des Modells bilden die Variationspunkte und die dazugehörigen Varianten. Konzeptionell entsprechen die Variationspunkte den veränderlichen Attributen eines Rechenkerns und die damit verknüpften Varianten repräsentieren die möglichen Ausprägungen.

Die Oberklasse `VariabilityModel` besteht aus zwei Attributen, einer Menge von `VariationPoint`-Objekten und einer Menge von `VariabilityConstraint`-Objekten, zur Modellierung von Beschränkungen zwischen zwei verschieden Varianten.

Anders als im orthogonalen Variabilitätsmodell ist ein `VariationPoint` entweder vom Typ `ClosedVariationPoint` (geschlossener Variationspunkt) oder vom Typ `OpenVariationPoint` (offener Variationspunkt). Ein `ClosedVariationPoint` beinhaltet eine Liste von Varianten *variants* und das Attribut *selectedVariantIndices*. Letzteres ist eine Liste von Indizes, mit der bestimmte Varianten in der Liste *variants* assoziiert werden. Auf diese Weise werden ausgewählte bzw. voreingestellte Varianten kodiert.

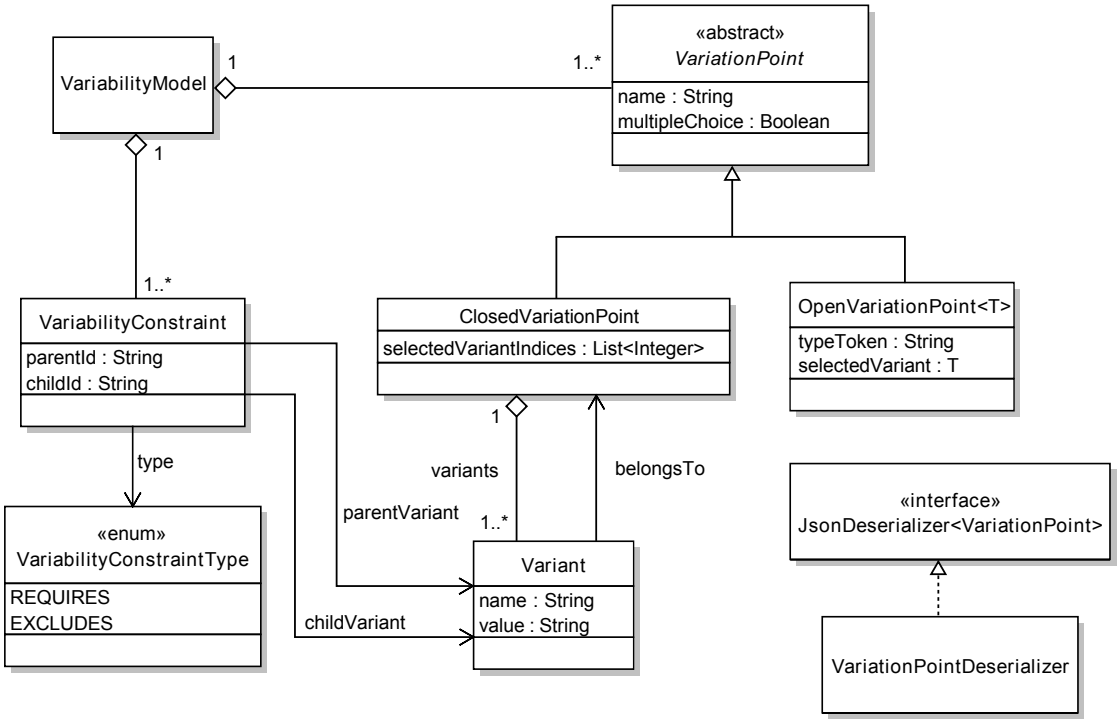

Abbildung 6.6.: Variabilitätsmodell von MeDIC und EMI

Ein OpenVariationPoint unterstützt einen parametrisierten Typ. So ist es möglich, einen Variationspunkt mit beliebigen Varianten aus dem Geltungsbereich des übergebenen Typs zu definieren. Das Attribut *selectedVariant* steht für einen ausgewählten bzw. voreingestellten Wert.

Das Modell verzichtet im Gegensatz zu dem orthogonalen Variabilitätsmodell auf eine Fragmentierung der Varianten in *Optional*, *Mandatory* und *Alternative Choice* mit Multiplizitäten. Stattdessen hat die Klasse VariationPoint ein Attribut namens *multipleChoice*, das festlegt, ob für die dazugehörigen Varianten eine Mehrfachausfall möglich sein soll oder lediglich eine Einfachauswahl. Diese Lösung ist weniger komplex und scheint für die Anwendung in MeDIC dennoch ausdrucksstark genug zu sein (vergleiche Abbildung 4.3). Ein weiterer Unterschied zum orthogonalen Variabilitätsmodell ist die Eigenschaft, dass eine Variante zu genau einem Variationspunkt gehört. Pro Variationspunkt sind hingegen mehrere Varianten erlaubt.

Des Weiteren bietet das Modell die Möglichkeit, Abhängigkeiten zwischen Varianten mit Hilfe der Klasse VariabilityConstraint festzulegen. Bei diesen Abhängigkeiten handelt es sich um Beschränkungen, die entweder vom Typ *REQUIRES* (erfordert) oder *EXCLUDES* (schließt aus) sein können (vgl. Abschnitt 4.3). Pro Variabilitätsmodell kann es eine beliebige Anzahl an Beschränkungen geben. Um das Modell möglichst einfach zu halten, besteht fürs Erste nur die Möglichkeit, Abhängigkeiten zwischen

zwei Varianten festzulegen. Bei Bedarf kann das Modell später erweitert werden, um zusätzlich Abhängigkeiten zwischen Varianten und Variationspunkten zuzulassen.

Details zu den Attributen *parentId*, *childId*, *typeToken* und der Klasse `VariationPointDeserializer` folgen im Abschnitt 7.1.

6.6. Webschnittstelle

Abbildung 6.7 zeigt einen vergrößerten Ausschnitt der obersten beiden Schichten von EMI, *Visualisierung* und *Berechnung und Persistierung*.

Das MeDIC-Dashboard befindet sich kontextuell auf der Visualisierungsschicht. Zur Nutzung der Webschnittstelle muss ein bestimmtes Paket eingebunden werden, welches die Funktionalität des Variabilitätsmodells kapselt. Dazu gehört das Variabilitätsmodell selbst und Klassen, die für die Serialisierung und Deserialisierung des Objekts zuständig sind. Der Begriff Serialisierung beschreibt den Vorgang strukturierte Daten (Objekte) in eine sequentielle Darstellungsform zu bringen. Der umgekehrte Vorgang wird als Deserialisierung bezeichnet. Bezogen auf den Webservice findet bei der Serialisierung eine Umwandlung von einem Objekt in ein Datenaustauschformat statt, welches anschließend übertragen wird (vergleiche Abschnitt 2.2.3). Der Empfänger erzeugt aus dieser Form anschließend wieder ein Objekt.

Jeder Rechenkern besitzt ebenfalls alle Funktionalitäten des Variabilitätsmodells. Auf diese Weise ist es möglich die Daten eines Objekts vom Typ `VariabilityModel` über den Webservice auszutauschen.

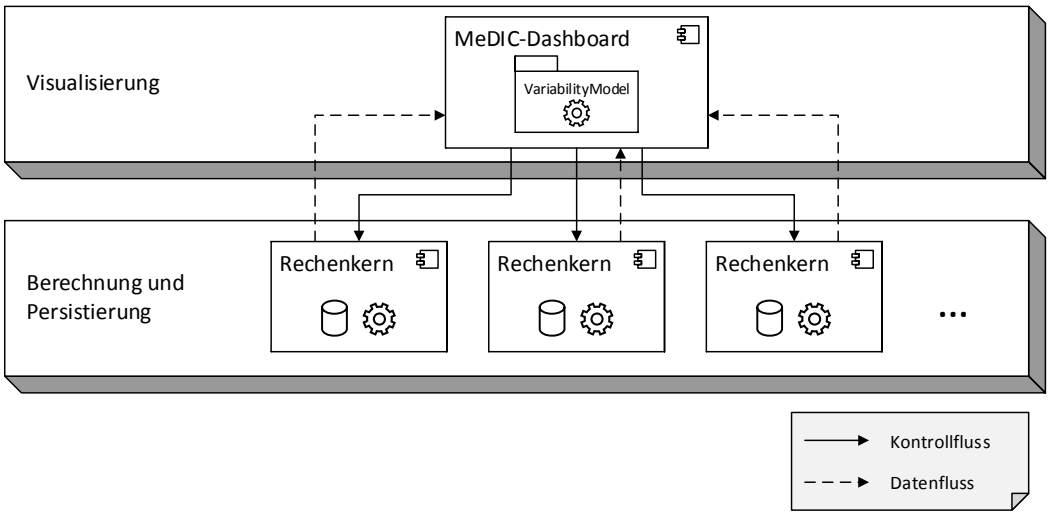

Abbildung 6.7.: EMI-Schichten: Visualisierung und Berechnung und Persistierung

Gemäß der definierten Anforderungen benötigt der Webservice drei Funktionen:

- `getVariability` zum Abrufen des Variabilitätsmodells.
- `getDataSeries` zum Abrufen von Messreihen.
- `getDataValue` zum Abrufen von Messwerten.

Beim Abrufen von Messreihen bzw. Messwerten soll das Variabilitätsmodell mit den ausgewählten Varianten als Parameter zurück an den Rechenkern geschickt werden. Dieser deserialisiert das Modell und identifiziert die ausgewählten Varianten auf dessen Basis anschließend die Berechnung der Metrik erfolgt. Auf diese Weise ist es möglich dem Rechenkern eine variable Anzahl an Einstellungsparametern zu übergeben.

Zusammengefasst ergibt sich folgender Ablauf:

- Das Variabilitätsmodell eines Rechenkerns wird zuerst an das MeDIC-Dashboard übertragen. Hier wird auf Basis der Variationspunkte ein Einstellungsdialog mit entsprechenden Eingabefeldern erstellt, die dem Benutzer die Konfiguration der Varianten erlauben.
- Wenn das Dashboard Messwerte eines Rechenkerns abruft, werden die ausgewählten Varianten – gekapselt in einem Variabilitätsmodell – als Parameter mitgeschickt. Bei diesem Variabilitätsmodell handelt es sich um das gleiche Modell wie vorher, modifiziert durch eine Menge von ausgewählten Varianten.