Entwurf einer Modellbibliothek für die kostengünstige Funktionsentwicklung mechatronischer Systeme

Sven Jacobitz[1*], Jie Zhang[1], Xiaobo Liu-Henke[1]

[1]Institut für Mechatronik, Ostfalia Hochschule für angewandte Wissenschaften, Salzdahlumer Str. 46/48, 38302 Wolfenbüttel; [*]*sve.jacobitz@ostfalia.de*

Kurzfassung. Aufgrund der stetig steigenden Komplexität und des hohen Vernetzungsgrades mechatronischer Systeme ist der Einsatz einer strukturierten, systematischen Entwicklungsmethode wie dem Rapid Control Prototyping (RCP) notwendig. Essenziell hierfür ist eine durchgängige Unterstützung durch eine CAE-Plattform. Solche Plattformen sind jedoch sehr kostenintensiv, weshalb an der Ostfalia die kostengünstige Plattform LoRra entstanden ist. Zentrales Element der Plattform ist die Modellbibliothek, welche über den gesamten Entwicklungsprozess den Zugriff auf einen konsistenten, rückverfolgbaren Datenstand ermöglicht. Durch Versions- und Konfigurationsmanagement wird zudem die Wiederverwendbarkeit der Prozess-Artefakte gesteigert. Der folgende Beitrag stellt den Entwurf der LoRra-Modellbibliothek für die kostengünstige Funktionsentwicklung mechatronischer Systeme vor.

Einleitung

Die Komplexität und Funktionsumfang mechatronischer Systeme nehmen immer weiter zu. Für kleine und mittelständische Unternehmen (KMU) stellt dieser Trend eine große Herausforderung dar. Um konkurrenzfähig zu bleiben, müssen sie immer mehr intelligente Hard- und Software in ihre Produkte integrieren. Neben der Anzahl an Funktionen eines Systems liegt dies auch am stetig steigenden Vernetzungsgrad der komplexen Softwarekomponenten, welche in starker Wechselwirkung miteinander stehen [1]. Zur Handhabung immer kürzerer Entwicklungszeiten bei höheren Qualitätsansprüchen ist eine strukturierte, systematische Entwicklung solcher Systeme unabdingbar [2]. Das durchgängig modellbasierte Rapid Control Prototyping (RCP) ist in diesem Zusammenhang eine häufig eingesetzte Methodik. Essenziell für RCP ist die durchgängige Unterstützung durch eine Computer Aided Engineering (CAE) Werkzeugkette, um einen hohen Automatisierungsgrad zu erreichen. Etablierte CAE-Werkzeugketten sind sehr kostenintensiv, was insbesondere für KMU eine große Hemmschwelle zur Einführung des RCP-Prozesses darstellt [3]. Im Rahmen des durch die EU geförderten Forschungsprojektes *Low-Cost Rapid Control Prototyping-System mit Open-Source-Plattform für die Funktionsentwicklung von eingebetteten mechatronischen Systemen (LoCoRCP)* ist daher an der Ostfalia die kostengünstige Entwicklungsplattform LoRra entstanden [4].

Um die Wiederverwendbarkeit von Modellen und Funktionen zu gewährleisten, wird ein systematisches Datenmanagement mit den dazugehörigen Teilprozessen wie Versions- und Konfigurationsmanagement eingesetzt. Dies ist aufgrund der hohen Vielfalt, Flexibilität und Kurzlebigkeit unabdingbar [5]. Erste Vorschriften hierzu kamen Anfang der 1960ger Jahre bei der NASA auf [6]. Nach Sax et. Al. [7] werden Inkonsistenzen während der Funktionsentwicklung zunehmend durch die hohe Variantenvielfalt hervorgerufen, was durch Einsatz eines entsprechenden Konfigurationsmanagements vermieden werden kann. Im Rahmen des RCP wird hierfür eine CAE-basierte Modellbibliothek, welche sämtliche relevanten Artefakte (Ergebnis eines Teilprozesses wie z. B. Modelle, Programmquelltext oder Dokumentation) verwaltet, benötigt [8].

In der klassischen Softwareentwicklung ist die Versions- und Konfiguraitonsverwaltung weit verbreitet. Eine Übersicht gibt [9]. Ein häufig verwendetes Open-Source-Werkzeug ist GIT [10]. Jedoch liegt der Schwerpunkt dieses Werkzeuges in der änderungsbasierten Verwaltung von Textdateien [11]. Eine Anwendung dieses änderungsbasierten Ansatzes ist jedoch auf im MBD gängige Datenformate nur wenig praktikabel [12], weshalb für die Nutzung in einer Modellbibliothek entsprechende Anpassungen notwendig sind.

Um Art und Umfang der notwendigen Anpassungen zu identifizieren wurden z. B. durch Niedzwiedz und Frei [13] systematische Untersuchungen durchgeführt. Hierbei wird ein Modell standardisiert aus Metadaten, Schnittstelleninformationen und Parametern aufgebaut. Basierend auf solch einen standardisierten Aufbau kann auch ein Versions- und Konfigurationsmanagement für komplexe, zusammengesetzte Modelle unter Einsatz einer System Entity Structure (SES) erfolgen [14].

Im folgenden wird die CAE-basierte LoRra-Modellbibliothek konzipiert, entworfen und exemplarisch realisiert. Der weitere Beitrag ist wie folgt aufgebaut: Abschnitt 1 fasst die RCP-Entwicklungsmethodik zusammen und führt in die LoRra-Plattform ein. In Abschnitt 2 erfolgt anhand einer Analyse der Anforderung die Konzeption des Lösungsansatzes, welcher in Abschnitt 3 zu einem Entwurf konkretisiert wird. Abschließend erfolgt in Abschnitt 4 eine Beschreibung der Realisierung. Abschnitt 5 fasst die Ergebnisse zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

1 Entwicklungsmethodik und -plattform

Aufgrund der hohen Systemkomplexität moderner vernetzter mechatronischer Systeme, wird der strukturierte, modellbasierte, verifikationsorientierte RCP-Prozess zur Entwicklung und Absicherung eingesetzt. Dieser besteht aus den Prozessschritten Modellbildung, Analyse / Synthese, automatisierte Generierung von Quelltext, automatisierte Implementierung auf einer Echtzeithardware sowie Onlineexperiment und wird durch Model-in-the-Loop- (MiL-), Software-in-the-Loop- (SiL-) sowie Hardware-in-the-Loop- (HiL-) Simulationen unterstützt [15].

Kennzeichnend für die vorgestellte Methodik sind die hohe Durchgängigkeit und der Automatisierungsgrad, von der Modellbildung, der modellbasierten Funktionsauslegung über die automatische Codegenerierung bis hin zur Echtzeitrealisierung (vgl. Abbildung 1 links). Sie wird durch eine durchgängige, voll automatisierte CAE-Plattform begleitet. Die modular aufgebaute, kostengünstige Entwicklungsplattform LoRra ist eine solche CAE-Plattform. Abbildung 1 illustriert den RCP-Entwicklungsprozess sowie die durchgängige Unterstützung mittels LoRra [4]. Von besonderer Relevanz ist hierbei eine zentrale Modellbibliothek, welche in sämtlichen Prozessschritten einen konsistenten, rückverfolgbaren Entwicklungsstand zur verfügbar macht.

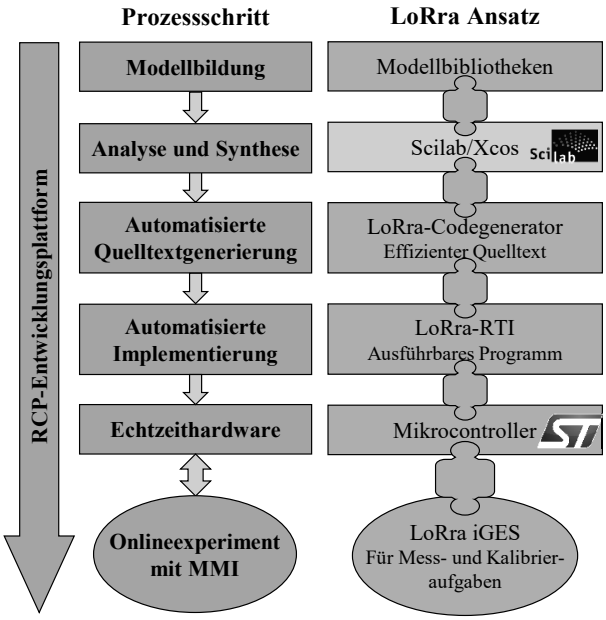

Abbildung 1: RCP-Entwicklungsprozess mit durchgängiger Unterstützung durch die LoRra-Plattform [4].

Für den Prozess der Modellbildung ist eine domänenübergreifende Modellbibliothek vorhanden. Mittels Versions- und Konfigurationsmanagement können Modellvarianten übersichtlich zusammengestellt und verwaltet werden. Das Open-Source-CAE-Werkzeug Scilab / Xcos wird zur Analyse und Synthese der Funktionen verwendet. Es bietet dabei einen ähnlichen Funktionsumfang wie das kommerziell häufig verwendete Matlab / Simulink. Das entstehende Funktionsmodell kann direkt in die Modellbibliothek integriert werden. Durch die offenen Schnittstellen der LoRra-API lassen sich zudem mit geringem Aufwand vorhandene Programme und Schnittstellentreiber einbinden. Mittels MiL-Simulationen können bereits in frühen Entwicklungsstadien Optimierungen und Tests der entwickelten Funktionen durchgeführt werden.

Durch den LoRra-Codegenerator wird mittels Modell-zu-Text-Transformation automatisiert effizienter, modularer C-Quelltext aus dem Funktionsmodell generiert. Durch offene funktionale Beschreibungen von Grundelementen des Modells, sogenannte Grundblöcke, ist der LoRra-Codegenerator flexibel erweiterbar. Der generierte Quelltext kann ohne manuelle Arbeiten, z.B. zur Optimierung und Test mittels SiL-Simulationen, wieder in das Xcos-Modell eingebunden werden.

Die Verbindung zu Modellen der Regelstrecke oder

weiteren Funktionen werden bei hinreichendem Funktionsstand durch Schnittstellenblöcke des LoRra Real-Time Interface (RTI) ersetzt. Hierdurch erfolgt ohne manuelle Programmierung die Ansteuerung der Echtzeithardware. In Kombination mit einer hardwarespezifischen RTI-Basissoftware, welche unter anderem ein Echtzeitbetriebssystem und standardisierte Schnittstellentreiber beinhaltet, wird somit eine automatisierte Implementierung auf der Echtzeithardware durch das RTI möglich. Als Echtzeithardware werden kostengünstige Mikrocontroller z.B. der Serie STM32H7 eingesetzt. Mittels HiL-Simulationen kann die entwickelte Funktion somit auch unter Echtzeitbedingungen optimiert und getestet werden. Als Mensch-Maschine-Interface (MMI) steht dabei die integrierte Graphikunterstützte Experimentiersoftware (iGES) zur Verfügung. Mit dieser lassen sich Onlineexperimente intuitiv steuern und überwachen sowie Messdaten aufzeichnen.

2 Konzeption der Modellbibliothek

2.1 Ansätze zur Entwicklung graphischer Bedienoberflächen

Für den strukturierten Softwareentwurf werden heutzutage standardisierte Architekturstile eingesetzt [16]. Diese dienen insbesondere zur Steigerung der Wiederverwendbarkeit, zur Strukturierung des Entwurfs sowie zur Schaffung eines einheitlichen Vokabulars. Über 25% der vorhandenen Stile dienen hierbei dem Entwurf einer Benutzeroberfläche [17]. Im Rahmen dieser Arbeit ist besonders das Modell / Präsentation / Steuerung (engl. Model / View / Controller, MVC) Prinzip relevant.

Der Architekturstil MVC in Anlehnung an [18] ist in Abbildung 2 illustriert. Hierbei werden die Visualisierung (Präsentation), die Steuerung und das Datenmodell getrennt voneinander mit definierten Schnittstellen realisiert. Die Steuerungskomponente reagiert auf Benutzereingaben in der graphischen Oberfläche und ändert bei Bedarf das Modell. Des Weiteren kann das Modell auch durch andere Softwarekomponenten geändert werden. Es benachrichtigt die Steuerung, über vorgenommene Änderungen, sodass diese die Präsentation aktualisiert. Aufgrund der geringen Komponentenverkopplung eignet sich dieses Prinzip besonders gut für MMIs, welche auf unterschiedlichen Zielplattformen eingesetzt werden [19]. So kann die Betriebssystem abhängige graphische Darstellung z. B. vollständig von der Steuerung und dem Modell entkoppelt werden.

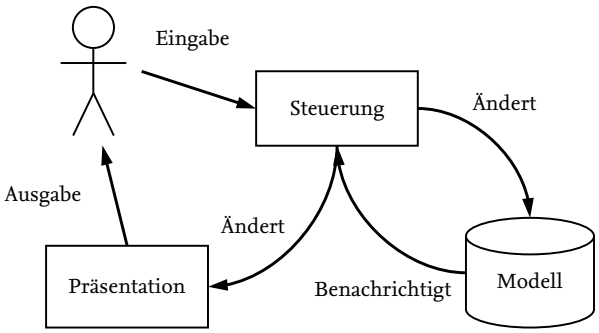

Abbildung 2: Architekturstil Modell / Präsentation / Steuerung.

2.2 Anforderungen an die Modellbibliothek

Wie aus Abschnitt 1 hervorgeht, ist die Modellbibliothek ein zentrales Werkzeug zum Datenmanagement in sämtlichen Prozessschritten. Um das Versions- und Konfigurationsmanagement in Gänze optimal zu unterstützen, werden übergeordnet folgende Anforderungen an die Modellbibliothek gestellt:

1. Versionierung sämtlicher enthaltener Daten sowie Unterstützung von Abläufen des Versionsmanagements (z. B. Prüfung, Freigabe).

2. Unterstützung der notwendigen Datenstrukturen zur Konfigurationsverwaltung in sämtlichen Prozessschritten sowie der Abläufe des Konfigurationsmanagements (z. B. Prüfung, Freigabe) zur Sicherstellung eines konsistenten Datenstands in sämtlichen Prozessschritten.

3. Hierarchische Strukturierung der Modelle in konfigurierbaren Kategorien und Hierarchieebenen.

4. Suchfunktion zum schnellen Finden spezifischer Modelle.

5. Unterstützung der Arbeit in verteilten Teams mit einer gemeinsamen Modellbasis.

6. Darstellung sämtlicher relevanter Modellinformationen in einer Übersicht.

2.3 Lösungsansatz

Um die Anforderungen an die Modellbibliothek umfassend zu erfüllen, wird zunächst der Aufbau eines Modells genauer betrachtet. Hierbei erfolgt eine Unterscheidung zwischen generischen und aggregierten Modellen.

Ein generisches Modell ist die kleinste in sich geschlossene Modelleinheit auf unterster Hierarchieebene. Sie untergliedert sich nicht in weitere hierarchisch geordnete Teilmodelle. Ein Beispiel für ein generisches Modell ist der elektrische Teil eines Gleichstrommotors, welcher sich durch Gl. (1) beschreiben lässt. Der Strukturierte Aufbau eines generischen Modells wird durch Abbildung 3 illustriert. Es besteht aus vier Komponenten:

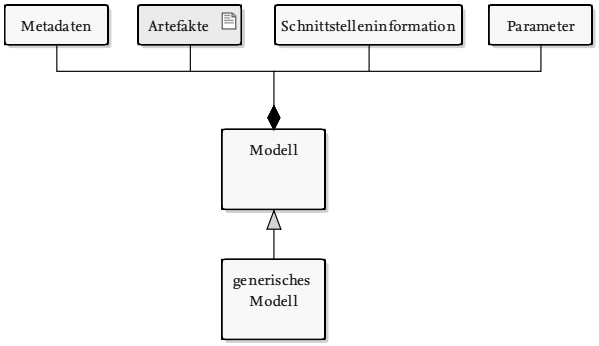

Abbildung 3: Strukturierter Aufbau eines generischen Modells.

- **Metadaten** beschreiben die übergeordneten Merkmale (z. B. Namen, Autor, allgemeine Beschreibung) des Modells.
- **Schnittstelleninformationen:** Datenstruktur, Einheiten und weitere relevante Informationen der Ein- und Ausgänge des Modells. Am Beispiel von Gl. (1) die Klemmspannung u in V als Eingang und der Motorstrom i in A als Ausgang.
- **Parameter:** Informationen und Werte zu den Parametern des Modells. Am Beispiel von Gl. (1) der Widerstand R in Ω und die Induktivität L in H.
- **Artefakte** des Modells wie z. B. die (Xcos-)Modelldatei, der generierte C-Code oder die Modelldokumentation.

$$u = Ri + L\frac{\mathrm{d}i}{\mathrm{d}t} - u_i \tag{1}$$

Ein aggregiertes Modell setzt sich aus weiteren Teilmodellen zusammen und bilden somit höhere Hierarchieebenen ab. Aggregierte Modelle werden als sogenannte Konfiguration in der Modellbibliothek abgebildet. Eine Konfiguration entsteht durch Integration definierter Versionsstände der Teilmodelle. Abbildung 4 illustriert das Prinzip.

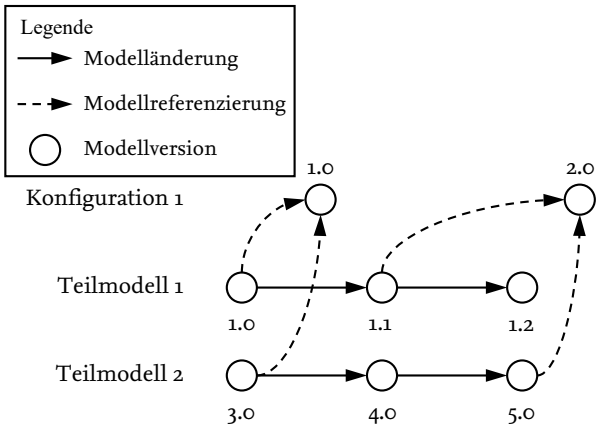

Abbildung 4: Prinzipielle Zusammensetzung einer Konfiguration.

Die Modelle sollen hierarchisch in einer Baumstruktur angeordnet werden. Diese Baumstruktur enthält Ordner (gruppierendes Hierarchieelement) und Modellelemente (sowohl generische als auch aggregierte Modelle). Sowohl für Gruppierungen als auch für einzelne Elemente können Benutzerrechte vergeben werden.

Um einen Modellzugriff für mehrere Benutzer zu ermöglichen, wird das Prinzip eines zentralen Speichers angewendet. Abbildung 5 verdeutlicht das Konzept. Über eine lokale Arbeitskopie erfolgt der Zugriff auf Modellartefakte. Änderungen werden an den zentralen Speicher, welcher als Datenbank fungiert, übermittelt und gespeichert. Nutzer können Sich die geänderten Daten anschließend in ihre lokale Arbeitskopie kopieren.

3 Entwurf der Modellbibliothek

Das Konzept aus Abschnitt 2 wird im Folgenden konkretisiert und in einen Entwurf umgesetzt. Hierzu werden zunächst die Datenstrukturen und Schnittstellen sowie das Datenmanagement betrachtet und anschließend die graphische Oberfläche entworfen.

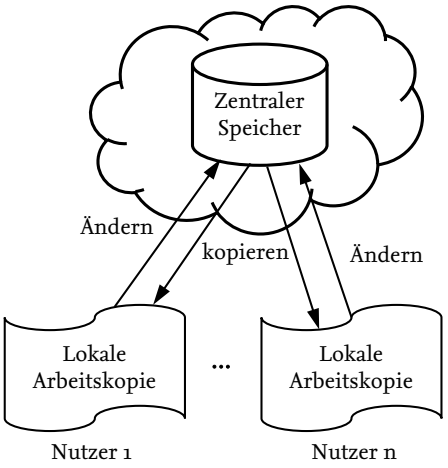

Abbildung 5: Konzept zur zentralen Speicherung von Modellen.

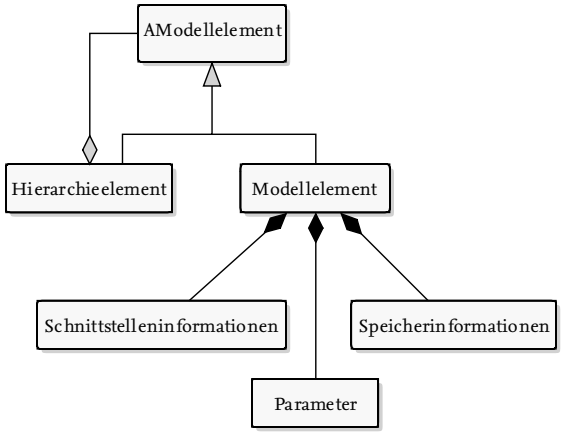

Abbildung 6: Klassenstruktur des hierarchischen Modellbaums.

3.1 Datenstrukturen

Für die Modellbibliothek werden verschiedene Datenstrukturen und Schnittstellen benötigt. Kernelement ist der hierarchische Modellbaum, welcher gleichzeitig als Datenbasis für die nach dem MVC-Prinzip aufgebaute graphische Oberfläche fungiert. Im folgenden wird exemplarisch die Datenstruktur und die Schnittstellen des Modellbaums entworfen.

Der Aufbau der Datenstrukturen erfolgt objektorientiert. Die abstrakte Klasse *AModellelement* repräsentiert die Grundstruktur für jedes Element des Baumes. Sie beinhaltet zentrale Daten wir Titel, Pfad im Baum oder Elternelement. Daraus abgeleitet werden die Klassen *Hierarchieelement* und *Modellelement*. *Hierarchieelement* beinhaltet eine Liste mit unterlagerten Elementen. *Modellelement* fasst neben weiteren Daten Schnittstelleninformationen, Parameter und Speicherinformationen des Modells zusammen. Abbildung 6 illustriert exemplarisch den Zusammenhang als UML-Klassenstruktur.

3.2 Datenmanagement

Das Datenmanagement der Modellbibliothek besteht im wesentlichen aus dem Versions- und Konfigurationsmanagement.

Um eine Versionierung und somit die konsistente Wiederverwendung in Form von Konfigurationen zu gewährleisten, muss die Modellbibliothek einen Versionsmanagementprozess unterstützen. Dies betrifft in erster Linie die Freigabe neuer Versionen. Wurden Änderungen an einem Modell vorgenommen, darf die Nutzung in Form einer neuen Modellversion erst nach firmenspezifischen Freigabeprozessen erfolgen. Für die LoRra-Modellbibliothek bedeutet dies, dass von Benutzern vorgeschlagene Versionen erst für die Allgemeinheit freigegeben werden, nachdem die nach dem firmenspezifisch konfigurierten Prozess notwendigen Personengruppen dem zugestimmt haben.

Unterschiedliche Versionsstände eines Modells werden durch Versionsnummern der Form x.y gekennzeichnet. Hierbei wird x als Major-Version und y als Minor-Version bezeichnet. Wurden bei einer Änderung keine kompatibilitätsrelevanten Anpassungen am Modell vorgenommen (z. B. Fehlerbehebungen, Verhalten und Schnittstellen bleiben jedoch gleich), erfolgt lediglich die Inkrementiereung der Minor-Version. Wurden Anpassungen vorgenommen, welche die Kompatibilität des Modells mit anderen Modellen beeinflussen (z. B. Änderungen an den Schnittstellen, Erweiterung der Funktionalität) wird die Major-Version inkrementiert und die Minor-Version zu 0 gesetzt. Die Zusammenstellung einer Konfiguration erfolgt durch Verknüpfung von Teilmodellen. Hierzu werden die entsprechenden Versionsnummern der Modelle referenziert. Abbildung 4 verdeutlicht das Prinzip.

3.3 Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche (engl. Graphical User Interface, GUI) wird nach dem in Abschnitt 2.1 eingeführten MVC-Prinzip aufgebaut. Die Datenbasis hierfür (Modell) stellt der in Abschnitt 3.1 entworfene Modell-

baum dar. Hinzu kommen für verschiedene Aufgaben konzipierte Steuerungsklassen.

Abbildung 7 illustriert den Gesamtentwurf der graphischen Bedienoberfläche (Präsentation) der LoRra-Modellbibliothek. Das Hauptfenster ist in drei Bereiche unterteilt. Der Navigationsbereich (1) enthält die hierarchische Modellbaum der Bibliothek. Hierüber können Nutzende sowohl Aktionen für einzelne Modelle ausführen (z. B. öffnen oder bearbeiten) als auch einen ersten Überblick über den aktuellen Modellstatus erhalten. Zudem kann der Modellbaum durchsucht werden. Der Anzeigebereich (2) enthält verschiedene Ansichten zur Informationsdarstellung und -bearbeitung. Hier können beispielsweise die Metadaten und Modellartefakte angezeigt oder die Versionshistorie eingesehen werden. Zusätzlich ist im Werkzeugbereich (3) eine kontextabhängige Werkzeugleiste sowie die Menüstruktur zur Bedienung der Bibliothek angeordnet.

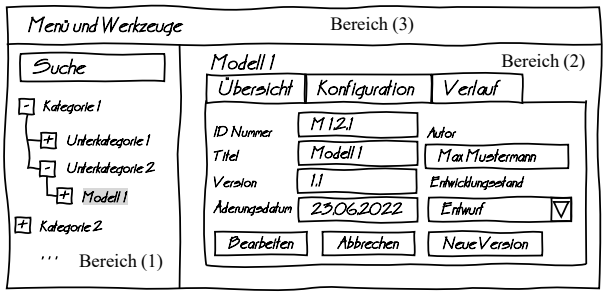

Abbildung 7: Entwurf der graphischen Oberfläche der Modellbibliothek.

4 Realisierung

Die Realisierung der Modellbibliothek erfolgt zunächst mit grundlegendem Funktionsumfang objektorientiert in Java als Eclipse Rich Client Platform (vgl. [20]). Das Eclipse-Framework bietet hierbei bereits viele zur Realisierung notwendige Mechanismen wie das *Standard Widget Toolkit* oder eine Ereignisbasierte, Kopplungsarme Kommunikation zwischen verschiedenen graphischen Elementen. Zudem ist eine Vielzahl an Erweiterung mit offenen Schnittstellen nutzbar.

Die Versionsverwaltung wird gemeinsam mit der Anbindung an die zentrale Speicherinfrastruktur über das vorhandene Open-Source-Werkzeug GIT (vgl. [10]) realisiert. Hier sind bereits bewährte Mechanismen zur Versionierung vorhanden. Durch den Einsatz strukturierter, textbasierter, Modellbeschreibungen lassen sich die einleitend erwähnten Einschränkungen umgehen. Zunächst erfolgt die Implementierung der Nutzerauthentifikation für den Atlassian Dienst Bitbucket. Eine Spätere Erweiterung ist möglich.

Die strukturierte Modellbeschreibung erfolgt im JSON-Format (vgl. [21]). Listing 1 beinhaltet einen exemplarisch gespeicherten Modellbaum. Hierarchieelemente werden durch die Felder *title* (Anzeigetitel des Elements), *relPath* (relativer Dateipfad zum übergeordneten Hierarchieelement) und *children* beschrieben. Modellelemente enthalten die Felder *relPath*, *metaFileName* und *repoUrl* (URL zum Online-GIT-Repository). Sämtliche relevanten Metadaten werden in der unter *metaFileName* angegebenen Datei gespeichert.

Listing 1: Exemplarischer Modellbaum im JSON-Format.

```
{
  "title" : "root",
  "relPath" : "",
  "children" : [ {
    "title" : "Fahrzeugmodelle",
    "relPath" : "Fahrzeugmodelle/",
    "children" : [ ... ]
  }, {
    "title" : "Funktionsmodelle",
    "relPath" : "Funktionsmodelle/",
    "children" : [ {
      "title" : "VMS",
      "relPath" : "VMS/",
      "children" : [ ... ]
    },
    {
      "title" : "AMS",
      "relPath" : "AMS/",
      "children" : [ {
        "relPath" : "efm/",
        "metaFileName" : "efm.json",
        "repoUrl" : "https://tinyurl
            .com/repo_efm/"
      }, ... ]
    } ]
  }, ... ]
}
```

Die Integration von Teilmodellen zu Konfigurationen erfolgt XML-basiert in Form einer SES. Hierdurch kann zunächst abstrakt die Struktur einer neuen Konfiguration erstellt werden. Durch Referenzierung der Teilmodelle unter Angabe von Version und Variante entsteht anschließend eine konkrete Konfiguration.

5 Zusammenfassung und Ausblick

Der vorliegende Beitrag stellt den Entwurf einer Modellbibliothek für die kostengünstige Funktionsentwicklung mechatronischer Systeme vor. Als Teil der auf Open-Source-Software basierenden durchgängigen RCP-Entwicklungsplattform LoRra bietet die Modellbibliothek in jedem Entwicklungsschritt eine konsistente und rückverfolgbare Datenbasis. Anhand der grundsätzlichen Anforderungen wurde ein Lösungsansatz für das Versions- und Konfigurationsmanagement hierarchischer Modelle sowie die zentrale Speicherung von Modellen erarbeitet. Anschließend wurden die Datenstruktur eines Modellbaums für hierarchische Modellkonfigurationen sowie die graphische Benutzeroberfläche entworfen. Abschließend erfolgte eine Zusammenfassung der Realisierung eines grundlegenden Funktionsumfangs.

Zukünftige Arbeiten befassen sich mit der weitergehenden Optimierung der Benutzerfreundlichkeit. Hierzu können weitere Funktionen zur Nutzerunterstützung (z.B. undo / redo) realisiert werden. Auch ist die Integration von graphischen Editoren, welche die Erstellung und Verwaltung von Konfigurationen vereinfachen, möglich. Für eine optimierte Übersicht in der Versionshistorie ist zudem eine Erweiterung des GIT-Werkzeugs *diff* vorgesehen, welche Änderungen in Xcos-Modellen visualisiert. Zuletzt ist eine Generalisierung der Nutzerauthentifikation möglich, sodass beliebige zentrale Speichersysteme genutzt werden können.

Danksagung

Gefördert vom Niedersächsischen Ministerium für Wissenschaft und Kultur unter Fördernummer ZN3495 im Niedersächsischen Vorab der VolkswagenStiftung und betreut vom Zentrum für digitale Innovationen Niedersachsen (ZDIN).

Literatur

[1] Quantmeyer F, Liu-Henke X. Entwicklung eines hochflexiblen HiL-Systems zur Echtzeiterprobung des elektronischen Fahrzeugmanagements. In: *Effizienz, Präzision, Qualität*. Univ. 2013; pp. 1–10.

[2] Liu-Henke X, Duym S. Modellgestützte Funktionsabsicherung des vernetzten mechatronischen Kraftfahrzeugs. In: *Mechatronik 2005*, VDI-Berichte. VDI-Verl. 2005; pp. 1073–1090.

[3] Liu-Henke X, Feind R, Roch M, Quantmeyer F. Investigation of low-cost open-source platforms for developing of mechatronic functions with rapid control prototyping. In: *10th International Conference on Mechatronic Systems and Materials*. Opole, Polen. 2014; pp. 1–9.

[4] Jacobitz S, Liu-Henke X. The Seamless Low-cost Development Platform LoRra for Model based Systems Engineering. In: *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS - Science and Technology Publications. 2020; pp. 57–64.

[5] Broy M, Kuhrmann M. *Projektorganisation und Management im Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg. 2013.

[6] Versteegen G, Weischedel G. *Konfigurationsmanagement*. Berlin, Heidelberg: Springer Berlin Heidelberg. 2003.

[7] Sax E, Reussner R, Guissouma H, Klare H. A Survey on the State and Future of Automotive Software Release and Configuration Management. 2017.

[8] Zehetner J, Lu W, Watzenig D, Bernasch J. Co-Simulation und Modellbibliothek - Eckpfeiler für einen modernen Systementwicklungsprozess. In: *Berechnung, Simulation und Erprobung im Fahrzeugbau 2012*, VDI-Berichte. VDI-Verl. 2012; pp. 597–607.

[9] Ratti N, Kaur P. Case Study: Version Control in Component-Based Systems. In: *Designing, Engineering, and Analyzing Reliable and Efficient Software*. IGI Global. 2013; pp. 283–297.

[10] Preißel R, Stachmann B. *Git: Dezentrale Versionsverwaltung im Team : Grundlagen und Workflows*. Heidelberg: dpunkt.verlag, 5th ed. 2019.

[11] Nugroho YS, Hata H, Matsumoto K. How different are different diff algorithms in Git? *Empirical Software Engineering*. 2020;25(1):790–823.

[12] Schmitz D, Deng W, Rose T, Jarke M, Nonn H, Sanguanpiyapan K. Configuration Management for Realtime Simulation Software. In: *35th Euromicro*

Conference on Software Engineering and Advanced Applications. IEEE. 2009; pp. 229–236.

[13] Niedzwiedz S, Frei S. A structured model library for the analysis of electric-vehicle drivetrains. In: *AmE 2012 - automotive meets electronics*, GMM-Fachbericht. VDE-Verl. 2012; pp. 21–26.

[14] Durak U, Pawletta T, Oguztuzun H, Zeigler BP. System entity structure and model base framework in model based engineering of simulations for technical systems. In: *Proceedings of the Symposium on Model-driven Approaches for Simulation Engineering*. Society for Computer Simulation International. 2017; pp. 1–10.

[15] Liu-Henke X, Jacobitz S, Scherler S, Göllner M, Yarom O, Zhang J. A Holistic Methodology for Model-based Design of Mechatronic Systems in Digitized and Connected System Environments. In: *Proceedings of the 16th International Conference on Software Technologies*. SCITEPRESS - Science and Technology Publications. 2021; pp. 215–223.

[16] Starke G. *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. München: Hanser, 9th ed. 2020.

[17] Henninger S, Corrêa V. Software pattern communities: current practices and challenges. In: *Proceedings of the 14th Conference on Pattern Languages of Programs - PLOP '07*. ACM Press. 2007; pp. 1–19.

[18] Adams S. MetaMethods: The MVC paradigm. *HOOPLA!* 1988;1(4).

[19] Liu Z, Li F, Liu H, Wu C, Zhang J. A Study of Cockpit HMI Simulation Design Based on the Concept of MVC Design Pattern. In: *Proceedings of the 2018 3rd International Conference on Modelling, Simulation and Applied Mathematics (MSAM 2018)*. Atlantis Press. 2018; pp. 82–84.

[20] Steppan B. *Eclipse Rich Clients und Plug-ins*. München: Carl Hanser Verlag GmbH & Co. KG. 2015.

[21] ISO/IEC 21778:2017: Information technology – The JSON data interchange syntax. *Standard*, International Organization for Standardization (ISO). 2017.