HELENA– Handhabung massiver verteilter Systeme mit ELaborierten ENsemble Architekturen [1]

Annabelle Klarl[2]

Abstract: Ensemble-basierte Systeme sind software-intensive Systeme mit einer großen Anzahl an Komponenten, die sich dynamisch zu kleineren, sich möglicherweise überschneidenden zielorientierten Kommunikationsgruppen zusammenschließen. Mit gängigen komponenten-basierten Entwicklungsmethoden können solche Systeme nur durch ein komplexes Modell beschrieben werden, das alle Ensembles und Komponenten gleichzeitig spezifiziert. Die HELENA-Entwicklungsmethodik schlägt vor, ein Ensemble nicht über die teilnehmenden Komponenten zu spezifizieren, sondern über Rollen, die Komponenten im Ensemble annehmen können. Rollen sind die aktiven Teilnehmer eines Ensembles, während Komponenten nur die technischen Ressourcen bereitstellen, um Rollen auszuführen. Die formal fundierte HELENA-Entwicklungsmethodik adressiert die speziellen Herausforderungen von ensemble-basierten Systemen in allen Phasen der Softwareentwicklung.

1 Einführung

Zunehmende Digitalisierung und globale Vernetzung prägen heutige Softwaresysteme. Sie bestehen aus einer großen Anzahl an Softwarekomponenten, die nicht nur auf leistungsstarken Servern, sondern auch auf einfacheren Geräten wie Personal Computern, Laptops oder Smartphones ausgeführt werden. Aufgrund der physischen Verteilung der Komponenten ist verteilte Ausführung eine inhärente Eigenschaft dieser Systeme. Trotzdem ist es oft gewünscht, dass mehrere Komponenten kurzfristig für ein gemeinsames Ziel zusammenarbeiten und sich dynamisch zu Kommunikationsgruppen, *Ensembles*, zusammenschließen. Aus Systemsicht umfasst dabei jedes Ensemble nur einen Teil der Komponenten, die im Gesamtsystem existieren. Im Ensemble selbst übernimmt jede Komponente eine bestimmte Aufgabe; sie spielt eine bestimmte *Rolle* im Ensemble. Sie kann allerdings auch (nacheinander oder gleichzeitig) mehrere Aufgaben bzw. Rollen in einem oder mehreren Ensembles übernehmen.

Die "Science Cloud Platform" (SCP) [KMH14], eine Fallstudie des EU-Projekts ASCENS [Wi15] verdeutlicht die Hauptcharakteristika eines ensemble-basierten Systems (EBS):

(1) Ein EBS baut auf einem großen verteilten, möglicherweise heterogenen komponenten-basierten System auf: Die SCP basiert auf einem Netzwerk aus vielen verteilten, heterogenen und freiwillig zur Verfügung gestellten Rechenknoten (z.B. in München und Berlin, Deutschland, sowie Lucca und Palermo, Italien). Im Gegensatz zu anderen Cloud Computing Plattformen können die Knoten nicht nur leistungsstarke Server, sondern auch Personal Computer, Laptops oder Smartphones sein.

[1] Englischer Titel der Dissertation: "HELENA– Handling massively distributed systems with ELaborate ENsemble Architectures"

[2] Ludwig-Maximilians-Universität-München, klarl@pst.ifi.lmu.de

(2) Komponenten schließen sich dynamisch zu (möglicherweise überlappenden) Ensembles zusammen, in denen sie durch Interaktion ein gemeinsames Ziel erreichen: Als Platform-as-a-Service Lösung ermöglicht die SCP einem Benutzer, Anwendungen bereitzustellen, auszuführen und zu verwenden. Pro Anwendung arbeiten dazu mehrere Knoten des zugrunde liegenden Netzwerks zusammen.

(3) In einem Ensemble müssen verschiedene Aufgaben erledigt werden: In der SCP bedarf die Anwendungsverwaltung vier verschiedener Aufgaben. Die Anwendung muss verfügbar gemacht, gespeichert, ausgeführt und Anfragen verarbeitet werden.

(4) Komponenten übernehmen Aufgaben im Ensemble, indem sie gewisse Rollen spielen. Welche Komponenten welche (möglicherweise mehrere) Rollen spielen, wird dynamisch zur Ausführungszeit bestimmt: Beispielsweise dient ein Laptop in Lucca dazu, die gewünschte Anwendung in der SCP verfügbar zu machen. Ein Münchner Personal Computer mit großen Speicherressourcen legt den Bytecode der Anwendung ab, während ein leistungsstarker Server in Berlin die Rechenressourcen hat, um die Anwendung auszuführen. Ein Wissenschaftler in Palermo stellt mittels seines Smartphones eine Anfrage an die ausgeführte Anwendung. Die Zusammensetzung des Ensembles kann sich pro verwalteter Anwendung ändern. Ein Rechenknoten könnte sogar mehrere Aufgaben gleichzeitig übernehmen, z.B. der Server in Berlin die Datenspeicherung sowie die Ausführung.

1.1 Zielsetzung

Es stellt sich die Frage, wie ensemble-basierte Systeme (EBS) systematisch entwickelt werden können. Eine spezielle Spezifikationstechnik soll es erlauben, Ensembles, Rollen und deren zielorientiertes Verhalten unabhängig von, aber gleichzeitig aufbauend auf der zugrunde liegenden komponenten-basierten Plattform zu beschreiben. Ferner soll sie unterstützen, dass Komponenten dynamisch an Ensembles teilnehmen und daher dynamisch eine oder mehrere Rollen annehmen können. Aufbauend auf der formalen Semantik einer Ensemblespezifikation sollen Ensembleziele präzise formuliert und deren Erfüllbarkeit formal verifiziert werden können. Eine systematische Übersetzung der Spezifikation zu lauffähigem Code soll es ermöglichen, das EBS auszuführen, während die Konzepte von Ensembles und Rollen im generierten Code erhalten bleiben sollen. Zusätzlich soll der Entwickler eines EBS in allen Entwicklungsschritten durch eine ganzheitliche Methodik mit systematischen Übergängen zwischen allen Phasen unterstützt werden. Werkzeuge zur Spezifikation, Verifikation und Implementierung sowie Transformatoren zwischen den jeweiligen Artefakten sollen die Entwicklungsarbeit erleichtern.

1.2 Die HELENA-Entwicklungsmethodik

Die Dissertation [Kl16] präsentiert die HELENA-Entwicklungsmethodik zur systematischen Umsetzung von dynamisch gebildeten Ensembles. Sie bedient sich des Konzepts von Rollen, um die Teilnehmer eines Ensembles unabhängig von den zur Verfügung stehenden Komponenten zu definieren und sie den Komponenten dynamisch zuzuweisen.

Im ersten Schritt der HELENA-Entwicklungsmethodik (siehe Abb. 1) wird das Domänenmodell des EBS als Ensemblestruktur modelliert. Es beschreibt die Eigenschaften und Fähigkeiten aller Teilnehmer bzw. Rollen im Ensemble und deren strukturelle Beziehungen. Basierend auf dem Domänenmodell werden die Ziele des Ensembles als Formeln in linearer temporaler Logik (LTL) angegeben. Das dynamische Verhalten des EBS wird mit Bedacht auf die Ensembleziele als eine Menge von Rollenverhalten entworfen. Zusammen mit der Ensemblestruktur bilden diese Rollenverhalten das HELENA-Entwurfsmodell. Um zu zeigen, dass das Verhalten der Rollen im Kontext der Ensemblestruktur die spezifizierten Ziele erreicht, wird das HELENA-Entwurfsmodell gegen seine Ziele verifiziert. Dafür werden die Ziele und das Entwurfsmodell nach PROMELA übersetzt, der Eingabesprache für den Model-Checker Spin [Ho03]. Die Erfüllbarkeit der Ensembleziele wird in dem erzeugten PROMELA-Verifikationsmodell geprüft. Um eine Ausführung des verifizierten Entwurfsmodell zu ermöglichen, wird die Java-Bibliothek jHELENA zur Verfügung gestellt. Sie setzt alle HELENA-Konzepte in Java um und erlaubt daher die direkte Realisierung eines HELENA-Entwurfsmodells. Beide Übersetzungen, von HELENA zu PROMELA und zu jHELENA, werden durch automatisierte Generatoren unterstützt.

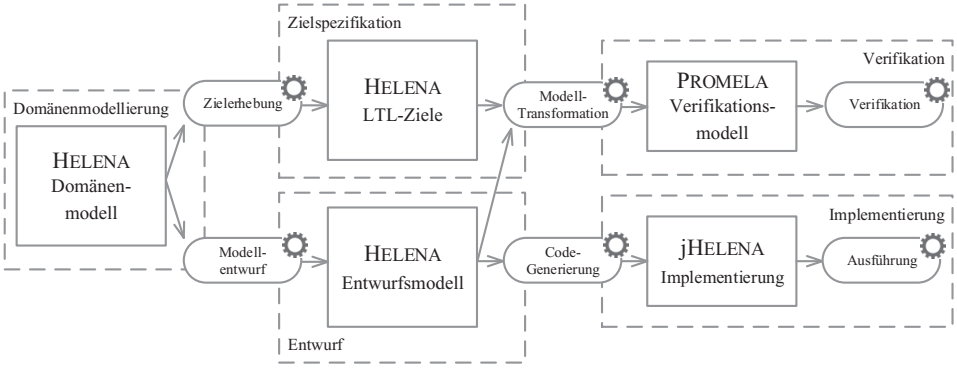

Abb. 1: Die HELENA-Entwicklungsmethodik für ensemble-basierte Systeme

1.3 Verwandte Arbeiten

Die HELENA-Entwicklungsmethodik ergänzt und führt Methoden und Techniken aus drei Wissenschaftsbereichen zusammen. Klassische komponenten-basierte Entwicklungsansätze wie [AG97] oder [MK96] beschreiben Komponenten sowie deren Komposition und Interaktion. Sie sind allerdings nicht ausreichend, um die Dynamik von EBS vollständig auszudrücken. Sie bieten kein explizites Modellierungskonzept für Ensembles und Rollen. Design und Analyse können daher nicht auf eine Gruppe von interagierenden Komponenten fokussiert und die unterschiedlichen Aufgaben in einem Ensemble nicht getrennt von den ausführenden Komponenten beschrieben und dynamisch zugewiesen werden. Ensemble-basierte Ansätze wie SCEL [De14] oder DEECo [Bu13] erweitern komponenten-basierte Ansätze um Zugehörigkeitsprädikate. Ein solches Prädikat bestimmt dynamisch anhand der Eigenschaften von Komponenten, welche Komponenten ein Ensemble bilden. Die Beschreibung der Architektur eines Ensembles ist jedoch nicht mög-

lich. Die HELENA-Entwicklungsmethodik erweitert komponenten-basierte und ensemble-basierte Ansätze um das Konzept von Rollen. Rollen erlauben nicht nur die dynamische Bildung von Ensembles über eine dynamische Zuweisung von Rollen an Komponenten, sondern ermöglichen auch die Beschreibung von Architektur und Teilnehmern eines Ensembles unabhängig von den zugrunde liegenden Komponenten. Im Gegensatz zu anderen Rollenmodellen [St00, Kü14] ist eine Rolle in HELENA eine aktive Entität, die ein zielorientiertes Verhalten hat und nur in einem bestimmten Kontext, dem Ensemble, existiert.

2 HELENA-Ensemblespezifikationen

Zur Modellierung der Domäne eines EBS und dem anschließenden Entwurf einer Lösung verwenden wir HELENA-Ensemblespezifikationen. Sie stellen Konzepte zur Verfügung, um zu beschreiben, welche Rollen von welchen Komponenten übernommen werden können und wie Rollen in einem Ensemble zusammenarbeiten.

Ensemblestrukturen Wir betrachten zunächst den strukturellen Aufbau eines Ensembles, die Ensemblestruktur. Eine Ensemblestruktur baut auf einer Menge von *Komponententypen* auf, z.B. dem Komponententyp `Node` in Abb. 2, der einen Rechenknoten im SCP-Netzwerk repräsentiert. Ein Komponententyp kann Assoziationen zu Komponententypen haben, z.B. die Assoziation `neighbor` des Komponententyps `Node`, die den Nachbar eines SCP-Knotens speichert. Komponentenattribute repräsentieren Kerninformationen, die für alle Rollen, die eine Komponente annehmen kann, von Bedeutung sind, z.B. das Attribut `memSize`, das die zur Verfügung stehende Speicherkapazität des SCP-Knotens angibt. Außerdem stellt ein Komponententyp Operationen zur Verfügung, die von Rollen aufgerufen werden können, z.B. die Operation `printResponse` (nicht gezeigt), die die Antwort auf eine Anfrage an einen SCP-Knoten ausgibt.

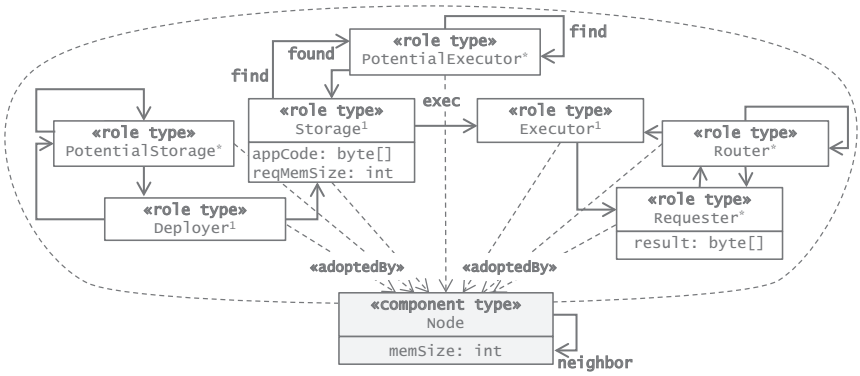

Abb. 2: Ensemblestruktur für SCP Fallstudie

Rollentypen repräsentieren die verschiedenen Arten von Teilnehmern eines Ensembles. Ein Rollentyp ist ein Tupel (*rtnm*, *rtcomptypes*, *rtattrs*, $rtmsgs_{out}$, $rtmsgs_{in}$): *rtnm* ist der Name des Rollentyps; die Menge *rtcomptypes* gibt an, von welchen Komponententypen dieser Rollentyp angenommen werden kann; die Menge *rtattrs* von Rollenattributen er-

möglicht die Speicherung von Informationen, die allein für den Rollentyp relevant sind; die Mengen $rtmsgs_{in}$ und $rtmsgs_{out}$ beschreiben die Nachrichten, die der Rollentyp senden und empfangen kann. Eine *Ensemblestruktur* definiert, welche Rollentypen und wie viele Instanzen jedes Typs zusammenarbeiten müssen. Wir nehmen an, dass zwischen zwei Rollentypen die Nachrichten ausgetauscht werden können, die von einem Rollentyp gesendet und vom anderen empfangen werden können. In der SCP-Fallstudie arbeiten sieben Rollentypen zusammen, die alle von Komponenten vom Typ `Node` gespielt werden können (siehe Abb. 2). Beispielsweise kann eine Rolle vom Typ `Storage` den Byte-Code der auszuführenden Anwendung im Attribut `appCode` und die Speicheranforderungen zur Ausführung der Anwendung im Attribut `reqMemSize` speichern. Sie kann die Nachricht `find` an eine Rolle vom Typ `PotentialExecutor` senden, um einen Knoten zu finden, der die Speicheranforderungen zur Ausführung der Anwendung erfüllt. Von dieser Rolle kann die Rolle vom Typ `Storage` die Nachricht `found` erhalten, sobald ein passender Knoten gefunden wurde. An diesen passenden Knoten in der Rolle `Executor` kann die Rolle vom Typ `Storage` schließlich die Aufforderung `exec` zur Ausführung der Anwendung senden.

Rollenverhalten Das Verhalten für einen Rollentyp geben wir als Prozessausdruck an. Ein Prozessausdruck wird aus Termination `quit`, Aktionspräfix $a.P$, nicht-deterministischer Auswahl $P_1 + P_2$, `if-then-else`-Ausdruck und Prozessaufruf `P` gebildet. Aktionen sind Erzeugung (`create`) und Wiederauffinden (`get`) einer Rolleninstanz, Senden (`!`) und Empfangen (`?`) einer Nachricht, Setzen von Rollen- und Komponentenattributen und Aufrufen einer Operation einer Komponente. Zu Verifikationszwecken können Zustandsmarkierungen eingeführt werden, um einen gewissen Fortschritt im Rollenverhalten zu markieren.

Ensemblespezifikationen Ein komplettes Ensemble mit seiner Struktur und seinem dynamischen Verhalten wird durch eine *Ensemblespezifikation* beschrieben. Sie besteht aus einer Ensemblestruktur und einem Rollenverhalten pro Rollentyp in der Ensemblestruktur.

Semantik Um die Zweiteilung von EBS in Komponenten und Rollen auszudrücken, wird die Semantik von Ensemblespezifikationen durch spezielle markierte Transitionssysteme (LTS) repräsentiert. Ein solches LTS beschreibt den Übergang von einem Ensemblezustand zu einem anderen durch Ausführung einer Aktion. Ein Ensemblezustand gibt einerseits an, welche Komponenten gerade welche Daten speichern, und andererseits, welche Rollen gerade von welchen Komponenten gespielt werden und sich in welchem Kontrollzustand gemäß ihres Verhaltens befinden. Übergänge zwischen Ensemblezuständen werden durch Aktionen von Rollen initiiert. Strukturell-operationelle Semantikregeln legen fest, welche Übergänge zwischen zwei Ensemblezuständen erlaubt sind, z.B. eine Nachricht kann nur zwischen zwei Rollen ausgetauscht werden, wenn die eine Rolle sie senden und die andere sie empfangen kann. Besonderheit der Semantikregeln ist eine erneute Zweiteilung: Die erste Ebene beschreibt, welche Übergänge für eine Rolle in Isolation erlaubt sind, während die zweite Ebene das Zusammenspiel zwischen Rollen festlegt.

3 Spezifikation von Zielen für HELENA-Ensemblespezifikationen

Die Rollen eines Ensembles arbeiten zusammen, um globale Ziele zu erfüllen. Ein Ziel ist eine temporale Eigenschaft, die entweder irgendwann während der Ausführung des

Ensembles erreicht (“achieve goal”) oder während der gesamten Ausführung des Ensembles aufrecht erhalten werden soll (“maintain goal”). Wir drücken Ensembleziele daher durch Formeln in linearer temporaler Logik (LTL) aus. Als atomare Propositionen verwenden wir Kontrollzustand-Propositionen und Attribut-Propositionen. Eine *Kontrollzustand-Proposition* $rt[i]@label$ ist erfüllt, wenn die Rolleninstanz i vom Typ rt einen durch $label$ markierten Zustand in ihrem Rollenverhalten erreicht hat. Eine *Attribut-Proposition* ist ein boolescher Ausdruck und wird aus Ausdrücken der Form $rt[i]{:}attr$ oder $ct[i]{:}attr$, Konstanten und den üblichen arithmetischen und relationalen Operatoren gebildet. Ein Ausdruck $rt[i]{:}attr$ (oder $ct[i]{:}attr$) beschreibt den Wert des Attributs $attr$ für die Rolleninstanz i vom Typ rt (oder für die Komponenteninstanz i vom Typ ct). LTL-Formeln zur Spezifikation von Ensemblezielen werden induktiv aus obigen Propositionen sowie den üblichen propositionalen und temporalen Operatoren gebildet und ihre Semantik wie üblich definiert.

In der SCP-Fallstudie soll ein Ensemble u.A. das Ziel erreichen, dass die Anwendung auf einem Knoten ausgeführt wird, der die nötigen Speicherressourcen hat (falls existent). In LTL formulieren wir das Ziel, wie folgt (da LTL keine Quantoren bietet, müssen alle Knoten im SCP-Netzwerk explizit aufgezählt werden):

$$(Node[1]{:}memSize \geq Storage{:}reqMemSize \vee Node[2]\ldots) \Rightarrow \Diamond(Node[1]{:}isExecuting \vee Node[2]\ldots)$$

4 Verifikation von Zielen für HELENA-Ensemblespezifikationen

Um die Erfüllbarkeit von Ensemblezielen in einer Ensemblespezifikation automatisiert zu verifizieren, verwenden wir den Model-Checker Spin [Ho03]. Wir übersetzen die HELENA-Ensemblespezifikation und deren Ziele nach PROMELA, der Eingabesprache für Spin, und verifizieren die übersetzten Artefakte mit Spin. PROMELA ist insbesondere gut als Zielsprache für die Übersetzung geeignet, da sie dynamische Prozesserzeugung (analog zu dynamischer Rollenerzeugung in HELENA) und asynchronen Nachrichtenaustausch (analog zu asynchronem Nachrichtenaustausch zwischen Rollen in HELENA) unterstützt. Um zu zeigen, dass die Model-Checking-Ergebnisse auf die ursprüngliche HELENA-Ensemblespezifikation übertragen werden können, beweisen wir, dass HELENA und PROMELA die gleiche Menge an LTL-Formeln erfüllen.

Übersetzung In HELENA sind Komponenten lediglich passive Entitäten, die ihre Speicher- und Rechenressourcen zur Verfügung stellen, während Rollen als aktive Entitäten ein zielorientiertes Verhalten ausführen. Um diese zweischichtige Modellierungsidee nach PROMELA zu übersetzen, werden Komponenten und Rollen zwar beide durch Prozesse in PROMELA ausgedrückt, aber haben unterschiedliche Kommunikationsfähigkeiten und Verhalten. Der Prozess für eine Komponente ist ein langlebiger Prozess, der nicht aktiv mit anderen Prozessen kommunizieren kann. Er wartet nur auf Anfragen von seinen angenommen Rollen (z.B. zur Rollenerzeugung oder Operationsaufruf), führt interne Berechnungen aus und antwortet entsprechend. Im Gegensatz dazu ist der Prozess für eine Rolle kurzlebig und repräsentiert das aktive zielorientierte Verhalten einer Rolle. Dabei wird Termination zu `false` übersetzt, Aktionspräfix zu sequentieller Komposition, nicht-deterministische Auswahl zu einer nicht-deterministischen `if`-Anweisung, der `if-then-else`-Ausdruck zu einer deterministischen `if`-Anweisung und Prozessaufruf zu einer `goto`-Anweisung. Senden und Empfangen von Nachrichten zwischen Rollen wird durch Nachrichtenaustausch

auf dedizierten Kanälen der entsprechenden Prozesse in PROMELA ausgedrückt; die Erzeugung und das Wiederauffinden von Rollen sowie Operationsaufruf werden vom entsprechenden Komponentenprozess verarbeitet.

Äquivalenzbeweis Bei der Übersetzung müssen einige HELENA-Prozessausdrücke und -Aktionen durch mehrere Anweisungen in PROMELA ausgedrückt werden. Beispielsweise wird in HELENA in einem einzigen Schritt eine neue Rolle erzeugt und eine Referenz auf die neu erzeugte Rolle in einer lokalen Variable gespeichert. In PROMELA wird hingegen zunächst ein neuer Kanal zum Nachrichtenempfang für die Rolle erzeugt und in einer lokalen Variable gespeichert. Anschließend wird ein neuer Prozess für die Rolle gestartet und ihm die Referenz auf den Kanal übergeben. Daher stellt sich die Frage, ob eine HELENA-Ensemblespezifikation und ihre PROMELA-Übersetzung semantisch äquivalent sind. In [Kl16] beweisen wir Stotterpfad-Äquivalenz für vereinfachte Varianten von HELENA und PROMELA. Diese Stotterpfad-Äquivalenz zeigt, dass zusätzliche Aktionen in PROMELA zwar die möglichen Verzweigungen eines Verhaltens einschränken, aber nicht die Belegung von atomaren Propositionen. Sie garantiert deshalb, dass beide Spezifikationen die gleichen LTL-Formeln erfüllen (falls der *next*-Operator nicht verwendet wird). Somit kann für solche Formeln das Model-Checking-Ergebnis von PROMELA nach HELENA übertragen werden.

Anwendung Für die SCP-Fallstudie können wir u.A. die Erfüllbarkeit des Ziels aus Abs. 3 mit Spin zeigen. Allerdings müssen wir bereits bei drei zugrunde liegenden Knoten im SCP-Netzwerk auf approximative Model-Checking-Methoden zurückgreifen. Die Größe des Suchraums übertrifft schon mit 18 parallelen Prozessen eine Speicherkapazität von 32 GB. Abhilfe könnte eine effizientere Übersetzung oder eine Abstraktion von der zugrunde liegenden komponenten-basierten Plattform schaffen.

5 Implementierung von HELENA-Ensemblespezifikationen

Um eine HELENA-Ensemblespezifikation implementieren und ausführen zu können, wird die prototypische Java-Bibliothek jHELENA zur Verfügung gestellt. jHELENA repräsentiert alle HELENA-Konzepte, wie Komponenten, Rollen und Ensembles, in Java. Dabei werden Rollen als Java Threads umgesetzt. Sie sind an ein bestimmtes Ensemble gebunden, während die darunter liegenden Komponenten mehrere Rollen in verschiedenen, parallel laufenden Ensembles annehmen können.

Die prototypische Implementierung dient zwei Zielen: Zum Einen zeigt jHELENA, wie die strukturellen und dynamischen Regeln der formalen HELENA-Modellierungskonzepte in einer objekt-orientierten Sprache umgesetzt und ausführbar gemacht werden können. Zum Anderen bietet jHELENA eine Programmierschnittstelle für Entwickler, um ensemblebasierte Anwendungen zu implementieren. Diese beiden Ziele werden durch eine zweischichtige Architektur mit einem orthogonalen Systemmanager adressiert (siehe Abb. 3):

Die `metadata`-Schicht erlaubt es dem Entwickler, das Metamodell einer Ensemblespezifikation zu definieren. Instanzen der `metadata`-Klassen repräsentieren die verschieden Arten von Typen (Komponenten- und Rollentypen, Ensemblestrukturen, ...),

die in einer Ensemblespezifikation auftreten. Eine Ensemblestruktur wird somit als Netzwerk von Objekten dargestellt, das gemäß der Regeln für Ensemblestrukturen gebildet wird.

Die `developer interface`-Programmierschnittstelle enthält abstrakte Basisklassen zur Implementierung von Komponenten-, Rollen- und Ensembleinstanzen. Während der Umsetzung einer konkreten ensemble-basierten Anwendung erweitert der Entwickler diese Basisklassen (z.B. mit `C1`, `C2`, `C3`, `R1`, `R2`, `E` in Abb. 3) und implementiert insbesondere das Verhalten jeder teilnehmenden Rolle. Die jHELENA-Bibliothek sorgt dafür, dass das Rollenverhalten gemäß der strukturellen Regeln für HELENA-Prozessterme implementiert und gemäß der HELENA-Semantik ausgeführt wird.

Der Systemmanger `SysManager` ist dafür verantwortlich, eine konkrete ensemble-basierte Anwendung zu konfigurieren und zu starten. Er erzeugt Komponenten, baut Ensemblestrukturen auf und startet konkrete Ensembles gemäß ihrer Struktur auf der zugrunde liegenden komponenten-basierte Plattform.

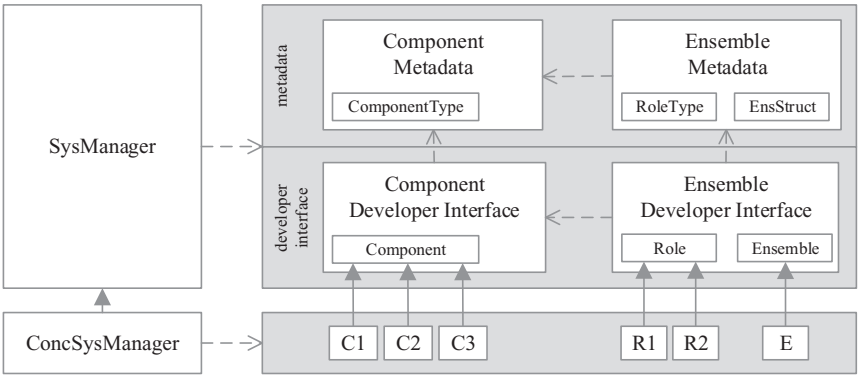

Abb. 3: Architektur der Java-Bibliothek jHELENA

Dank der jHELENA-Bibliothek kann die SCP-Fallstudie folgend ihrer Spezifikation in HELENA modular implementiert werden. Die sieben Rollen der HELENA-Ensemblespezifikation aus Abb. 2 werden durch sieben Klassen mit nur 1200 LOC sowie weiteren 400 LOC in Klassen für Nachrichten umgesetzt. Zusätzlich wird jHELENA um einige SCP Technologien für robusten Nachrichtenaustausch und Datenspeicherung erweitert. Mit dieser modularen Implementierung ist es möglich die SCP-Fallstudie mit 100 verteilten Knoten auszuführen und gleichzeitig mehrere Ensembles auf der SCP zu starten.

6 HELENA-Workbench

Um den Entwickler eines EBS bei der Spezifikation, Verifikation und Implementierung sowie insbesondere bei den Übergängen zwischen den einzelnen Phasen zu unterstützen, bieten wir ein passendes Entwicklungswerkzeug an: die HELENA-Workbench. Sie erweitert die Eclipse-Entwicklungsumgebung um einen HELENA-spezifischen Editor, an den

automatische Code-Generatoren zu PROMELA und Java angeschlossen sind. Der Editor bietet eine umfangreiche Entwicklungsunterstützung mit Syntaxhervorhebung, Autovervollständigung und Validierung gemäß der domänenspezifischen Sprache HELENATEXT, die als konkrete Syntax für HELENA-Ensemblespezifikationen dient. Die systematischen Übersetzungsregeln von HELENA zu PROMELA und Java bilden die Grundlage für zwei regelbasierte Code-Generatoren. Sie garantieren eine korrekte (teilweise bewiesene) Übersetzung einer HELENA-Ensemblespezifikation zu einem PROMELA-Verifikationsmodell und einer ausführbaren Implementierung gemäß der Java-Bibliothek jHELENA.

7 Zusammenfassung und Ausblick

HELENA [Kl16] ist eine Methodik mit Techniken und Werkzeugen zur systematischen Entwicklung von ensemble-basierten Systemen. Ein Ensemble bildet die strukturelle Einheit, in der mehrere Rollen kollaborieren, um ein gemeinsames Ziel zu erreichen. Jede Rolle führt dazu ein zielorientiertes Verhalten aus. Komponenten können dynamisch an einem Ensemble teilnehmen, indem sie eine oder mehrere Rollen im Ensemble annehmen. Alle Spezifikationselemente wurden formal definiert und mit einer rigorosen Semantik versehen. Dies erlaubt die formale Definition von Ensemblezielen und deren Erfüllbarkeit. Zur automatisierten Verifikation von Zielen in einer Ensemblespezifikation wurde eine Übersetzung nach PROMELA und die Verwendung des Model-Checkers Spin vorgeschlagen. Die Korrektheit dieses Ansatzes wurde mit Hilfe von Stotterpfad-Äquivalenz zwischen HELENA und PROMELA bewiesen. Zur Implementierung und Ausführung einer Ensemblespezifikation wurde die prototypische Java-Bibliothek jHELENA entwickelt. Sie stellt alle HELENA-Konzepte als Programmierschnittstelle für einen Entwickler bereit und setzt die HELENA-Semantik der Ausführung eines ensemble-basierten Systems um. Der Entwickler erhält mit der HELENA-Methodik eine Richtlinie zur systematischen Entwicklung von ensemble-basierten Systemen und wird in allen Phasen durch ein einheitliches Entwicklungswerkzeug, der HELENA-Workbench, unterstützt.

Für die Weiterentwicklung von HELENA bieten sich ihre Anwendung auf andere Arten von Systemen sowie die Erweiterung ihrer Ausdrucksmächtigkeit an. Beispielsweise wurde in [Kl16] bereits gezeigt, dass das Konzept von Rollen auf Komponenten angewendet werden kann, die sich an eine dynamische Umgebung anpassen. Eine wünschenswerte Erweiterung wäre die Integration von adaptiven Komponenten in Ensembles, die sich gemeinsam an die Umgebung anpassen. Weiterhin wäre es interessant, HELENA auf Domänen mit unsicheren Effekten von Aktionen auszuweiten. Im Zusammenspiel mit künstlicher Intelligenz könnte Verhalten unter Zuhilfenahme von Rollen generiert werden.

Literaturverzeichnis

[AG97] Allen, Robert; Garlan, David: A Formal Basis for Architectural Connection. ACM Transactions on Software Engingeering and Methodology, 6(3):213–249, 1997.

[Bu13] Bures, Tomas; Gerostathopoulos, Ilias; Hnetynka, Petr; Keznikl, Jaroslav; Kit, Michal; Plasil, Frantisek: DEECO: An Ensemble-based Component System. In: International

ACM SIGSOFT Symposium on Component-based Software Engineering. ACM, S. 81–90, 2013.

[De14] De Nicola, Rocco; Loreti, Michele; Pugliese, Rosario; Tiezzi, Francesco: A Formal Approach to Autonomic Systems Programming: The SCEL Language. ACM Transactions on Autonomous and Adaptive Systems, 9(2):1–7, 2014.

[Ho03] Holzmann, Gerard: The Spin Model Checker: Primer and Reference Manual. Addison-Wesley, 2003.

[Kl16] Klarl, Annabelle: HELENA– Handling massively distributed systems with ELaborate ENsemble Architectures. Dissertation, Ludwig-Maximilians-Universität München, November 2016.

[KMH14] Klarl, Annabelle; Mayer, Philip; Hennicker, Rolf: Helena@Work: Modeling the Science Cloud Platform. In: International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Jgg. 8373 in Lecture Notes in Computer Science. Springer, S. 99–116, 2014.

[Kü14] Kühn, Thomas; Leuthäuser, Max; Götz, Sebastian; Seidl, Christoph; Aßmann, Uwe: A Metamodel Family for Role-Based Modeling and Programming Languages. In: International Conference on Software Language Engineering. Jgg. 8706 in Lecture Notes in Computer Science. Springer, S. 141–160, 2014.

[MK96] Magee, Jeff; Kramer, Jeff: Dynamic Structure in Software Architectures. In: ACM SIGSOFT Symposium on Foundations of Software Engineering. Jgg. 21. ACM, S. 3–14, 1996.

[St00] Steimann, Friedrich: On the Representation of Roles in Object-Oriented and Conceptual Modelling. Elsevier Data and Knowledge Engineering, 35(1):83–106, 2000.

[Wi15] Wirsing, Martin; Hölzl, Matthias; Koch, Nora; Mayer, Philip: Software Engineering for Collective Autonomic Systems, Jgg. 8998 in Lecture Notes in Computer Science. Springer, 2015.

Annabelle Klarl wurde 1986 in München geboren. Während ihres Bachelorstudiums der Bioinformatik beschäftigte sie sich mit der computergestützten Analyse und Verwaltung von Genomdaten. Für das Masterstudium und ihre anschließende Promotion wechselte sie zum Fachgebiet Software Engineering. In ihrer Dissertation an der Ludwig-Maximilians-Universität München entwickelte sie eine systematische Methodik zur Entwicklung von großen verteilten Systemen, in denen sich autonome Komponenten dynamisch zu zielorientierten Gruppen, so genannten Ensembles, zusammenschließen. Als wissenschaftliche Mitarbeiterin war sie an dem EU-Projekt ASCENS beteiligt, konnte ihre wissenschaftliche Expertise als Keynote-Sprecherin auf der VAO 2015 (Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling) in L'Aquila, Italien vermitteln und evaluierte als Mitglied des Programm-Komitees für DAS@ICAC 2015, DAS@ICAC 2016 und LASSY 2017 Einreichungen zum Thema Adaptive Systeme. Darüber hinaus stellte sie ihre Lehrbefähigung durch die Betreuung und teilweise Neukonzeption von sechs verschiedenen Lehrveranstaltungen unter Beweis.