Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Fakultät für Informatik
Lehrstuhl für Programmiersysteme

Automatische Erzeugung von Prüflisten zur spezifikationsbezogenen Beurteilung der Vollständigkeit von UML-Modellen

Studienarbeit am
Institut für Programmstrukturen und Datenorganisation
Lehrstuhl Programmiersysteme
Prof. Dr. Walter F. Tichy
Fakultät für Informatik
Universität Karlsruhe (TH)

von

Mathias Landhäußer

Betreuer:
Prof. Dr. Walter F. Tichy
Dipl.-Inform. Tom Gelhausen

Tag der Anmeldung: 24. April 2008
Tag der Abgabe: 24. Juli 2008

Hiermit erkläre ich, dass ich diese Studienarbeit selbstständig verfasst habe. Alle nicht von mir stammenden Inhalte sind durch Angabe der Quelle kenntlich gemacht.

Karlsruhe, den 21. Juli 2008

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

Diese Studienarbeit ist in ein größeres Projekt eingebettet, dessen Zielsetzung die automatische Erzeugung von domänenspezifischen Modellen ausgehend von natürlichsprachlichen Spezifikationen ist. Ziel der vorliegenden Arbeit ist es, eine Spezifikation mit SENSE zu verarbeiten und eine Prüfliste zu erzeugen, welche die Überprüfung eines Modells auf seine Vollständigkeit bezüglich der Spezifikation ermöglichen soll.

Die Schritte zum Erreichen dieses Zieles sind zugleich die Gliederungspunkte des 4. Kapitels: Zunächst wird die Spezifikation mittels SAL_E für SENSE vorbereitet. Anschließend werden die UML-Elemente und -konstrukte identifiziert, die zur Umsetzung der thematischen Rollen von SAL_E in UML verwendet werden können. Daraufhin wird beschrieben, inwiefern das SAL_E-Graphenmodell für GrGen.NET erweitert werden musste und wie die Graphersetzungsregeln aufgebaut wurden, die zur Erzeugung der Prüfliste verwendet werden.

Zuvor werden im 3. Kapitel die zugrundeliegenden Theorien und Ansätze erläutert. Im 5. Kapitel wird ein erstes Experiment zur Evaluation der Prüfliste beschrieben.

2. Grundlagen

Dieses Kapitel geht kurz auf die Grundlagen ein. Zunächst werden die Model Driven Architecture und das Software Engineers' Natural language Semantics Encoding (SENSE) beschrieben. Anschließend wird die SENSE Annotation Language for English (SAL_E) anhand eines Beispiels eingeführt und die Basisfunktionen von GrGen.NET gezeigt. Den letzten Teil dieses Kapitels bildet eine kurze Motivation für eine spezifikationsbezogene Prüfliste.

2.1. Die modellgetriebene Architektur

Um der stetig steigenden Komplexität der Softwareentwicklung Herr zu werden, wurden immer wieder neue Entwicklungsmethoden benötigt. Dieser Fortschritt begann mit der Maschinensprache, ging über die Assemblersprachen zu prozeduralen und nun hin zu objektorientierten Konzepten. Ein weiterer konsequenter Schritt in dieser Entwicklung ist die Einführung einer weiteren Abstraktionsstufe – von Modellen [GPR06, 9ff][MSUW04, 2ff]. Die modellgetriebene Architektur (Model-Driven Architecture, MDA) [MM03] der Object Management Group beschreibt einen modernen Entwicklungsprozess, der das zu erstellende Softwaresystem ausgehend von Modellen von oben nach unten aufbaut.

Das oberste, abstrakteste Modell ist das berechnungsunabhängige Modell (Computation Independent Model, CIM), welches auch als domänenspezifisches Modell bezeichnet wird. Es stellt den Ausgangspunkt des Prozesses dar und beschreibt, in welcher Umgebung sich das System befindet und wie es sich verhalten soll. Das System als solches wird auf dieser Stufe nicht als Programm verstanden, sondern kann alles sein: Ein Verbund von Rechnern, ein einzelner Computer oder eine Softwarekomponente. Ebenfalls Teil des Systems können Personen oder Unternehmen sein, die notwendig sind um die anstehende Aufgabe zu bewältigen oder die dem System erst die konkrete Aufgabe stellen. Das CIM ist die Schnittstelle zwischen Domänenexperten und Softwareentwicklern [MM03, 2-5].

Während das CIM als deskriptives Modell zu verstehen ist, haben das plattformunabhängige Modell (Platform Independent Model, PIM) und das plattformspezifische Modell (Platform Specific Model, PSM) einen präskriptiven Charakter [Küh06]: Das PIM steht auf der zweiten Stufe und stellt die Funktionalität des Systems plattformunabhängig [1] dar [MM03, 2-6]. Auf der untersten Stufe steht das PSM, welches die Elemente des PIM darstellt und die benutzten plattformspezifischen Funktionen zeigt.

[1] Plattformunabhängigkeit bezieht sich in diesem Kontext nicht nur auf die Hardwareumgebung, sondern auch auf die Software (JAVA, .NET etc.).

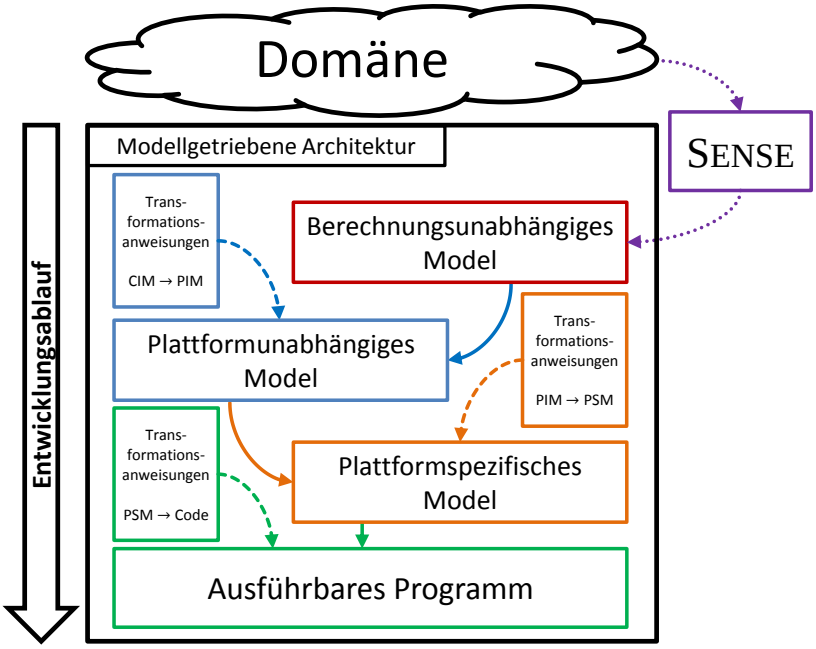

Abbildung 2.1.: Die Modelle der MDA und SENSE nach [MM03] und [GT07].

Die MDA sieht vor, die Modelle in der genannten Reihenfolge zu erstellen und dabei das jeweils obere in das nächst feinere Modell zu überführen. Hierbei kann der Transformationsprozess von Modell zu Modell manuell, computergestützt oder automatisch sein. In jedem dieser Fälle ermöglicht die MDA, die Modelle bei einer Änderung des jeweils übergeordneten Modells kontrolliert zu aktualisieren. Auf unterster Stufe ist beispielsweise eine automatische Transformation eines UML-Modelles nach Java-Code möglich.

2.2. SENSE und SAL$_E$

Das Software Engineers' Natural language Semantics Encoding (SENSE) zielt auf die automatische Erzeugung des initialen domänenspezifischen Modells ab [GT07]. Hierdurch soll der MDA-Prozess vereinfacht und beschleunigt werden. Um dieses Ziel zu erreichen, werden in SENSE natürlichsprachliche Texte – beispielsweise eine technische Spezifikation – in UML-Modelle transformiert. Hierdurch müssen Modelle bei (Anforderungs-) Änderungen nicht manuell angepasst beziehungsweise neu erstellt werden.

SENSE sieht vor, den Text mittels der SENSE Annotation Language for English (SAL$_E$) zu annotieren; die hierdurch hinzugefügten semantischen Informationen können anschließend von einem Rechner ausgewertet und zum Erzeugen des CIM herangezogen werden.

Phrasen und Subphrasen werden in SAL$_E$ mit eckigen Klammern ([und]) gekennzeichnet; Mengen werden durch geschwungene Klammern und den Verknüpfungsoperator

definiert: `{Apfel, Birne AND Banane}` und `{Cola, Fanta OR Limo}`. Die markanteste Annotationsform in SAL$_E$ sind die thematischen Rollen: Sie geben an, welche Rolle ein Element in einem Satz spielt. Gekennzeichnet werden sie durch das Anfügen des Rollennamens an das jeweilige Satzelement, getrennt durch einen senkrechten Strich (|) (beispielsweise `Mathias|AG`). Unnötige Elemente können mit einem # auskommentiert werden; längere Passagen können mit geschwungenen Klammern zusammengefasst und `#{en bloc}` auskommentiert werden. Angaben über Multiplizitäten werden mit einem Stern (*) versehen (beispielsweise `*5 Äpfel`). Attribute von Elementen werden mit einem Dollar-Zeichen gekennzeichnet (beispielsweise ein `$roter Apfel`)[2]. Eine wichtige Sonderrolle nehmen Referenzen ein. Sie werden dazu verwendet, auf ein anderes Element zu verweisen, das vorher bereits im Text erwähnt wurde. Hierbei handelt es sich nicht um gleiche Elemente (dieselbe *Klasse*), sondern um dieselben (dieselbe *Instanz*). Elemente, die eine Referenz darstellen, werden mit einem @ gekennzeichnet:

Listing 2.1: Referenzen in SAL$_E$.

```
[ Tom and Mathias play soccer ].

/* NICHT: [ Tom scores again ]. */
[ @Tom scores again ].

/* Hier sollte auch eine Referenz verwendet werden: */
[ Tom wins ].
```

Der Satz „Tom plays soccer with Sven.“ kann folgendermaßen annotiert werden:

Listing 2.2: Kurzes SAL$_E$-Beispiel mit *dux* und *comes*.

```
[ Tom|{AG,DUX} plays_soccer|ACT #with Sven|COM ].
```

In diesem Beispiel werden die Rollen *actus* (ACT), *agens* (AG), *dux* (DUX) und *comes* (COM) verwendet. Sie machen kenntlich, dass Tom und Sven zwei Objekte sind, die miteinander in einer Begleiter-Beziehung stehen (*dux* und *comes*); der *actus* „plays_soccer“ bezeichnet die Handlung, die vom *agens*, dem Handelden, „Tom“ durchgeführt wird. Eine vollständige Liste er in SAL$_E$ verfügbaren Rollen befindet sich im Anhang in Tabelle A.1.

Neben SAL$_E$ wurde eine neue Art von Graphen namens Omnigraphen[3] entwickelt, die eine kompakte Darstellung von Sätzen natürlicher Sprache ermöglichen [Den07]. Omnigraphen können – wenn auch weniger kompakt – als normale Graphen dargestellt werden [DGG08]. Diese Tatsache ermöglicht es, die Informationen aus dem erzeugten Graphen mit einem normalen Graphersetzungssystem zu verwerten. Auf das verwendete Graphersetzungssystem GrGen.NET wird im folgenden Unterkapitel näher eingegangen.

Wurde der Text vollständig annotiert, kann er vom SAL$_E$-Übersetzer in einen Spezifikationsgraphen transformiert werden, welcher in GrGen.NET geladen werden kann.

[2] Attibute in SAL$_E$ beziehen sich immer auf das direkt folgende Element, sofern nichts anderes angegeben wird. Siehe hierzu auch Abschnitt 4.3.4.

[3] Die ursprünglich eingeführte und in [GT07] verwendete Bezeichnung lautete „Supergraphen“.

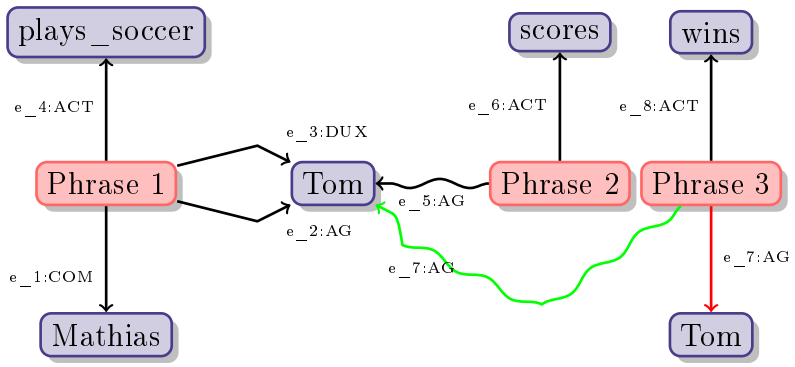

Abbildung 2.2.: Verwendung von Referenzen im Spezifikationsgraphen.

Hierbei wird für jede Phrase ein Knoten vom Typ PHRASE und für jedes Wort ein Knoten vom Typ WORD angelegt. Die vom PHRASE-Knoten ausgehenden Kanten zeigen auf die in der Phrase verwendeten Element-Knoten und sind vom Typ der jeweiligen thematischen Rolle. Verwendet man bei der Annotation eine Referenz, wird für das zweite Element kein neuer Knoten angelegt, sondern nur eine neue Rollenkante an den bereits bestehenden Knoten angefügt. Abbildung 2.2 greift das Beispiel aus Listing 2.1 auf und stellt die Arbeitsweise des Übersetzers bei der Verwendung von Referenzen dar. Normale Kanten sind als gerade Pfeile, Kanten, die aufgrund von Referenzen erzeugt wurden mit gewellten Pfeilen eingezeichnet; die fälschlicherweise ausgelassene Referenz ist in grün (richtig: mit Referenz) und rot (falsch: ohne Referenz) in der Abbildung zu sehen.

Im weiteren Verlauf des Prozesses ist vorgesehen, die auftretenden Mehrdeutigkeiten der natürlichen Sprache zu eliminieren – dies kann beispielsweise über einen interaktiven Prozess[Wol06] oder automatisiertes linguistisches Reasoning [KG08] geschehen.

Nach Abschluss dieses Prozesses soll das CIM als UML-Modell ausgegeben werden.

2.3. GrGen.NET

GrGen.NET [KG07] stellt als Backend die Arbeitsgrundlage für die notwendigen Modifikationen am Spezifikationsgraphen dar. Es ist ein Graphersetzungssystem (GRS), welches neben der Regelauswertung auch eine graphische Schnittstelle anbietet. Graphersetzungsregeln bestehen aus einem Suchmuster (dem gesuchten Teil des Arbeitsgraphen), einer oder mehreren (negativen) Anwendungsbedingungen und einem Ersetzungs- oder Modifikationsteil.

Gezeigt wird die Verwendung von GrGen.NET beispielhaft in Abbildung 2.3; hier wird das körperlich anstrengende Fußballspielen durch das Schachspielen ersetzt.

Wie im folgenden Quelltext zu sehen ist, können neue Knoten und Kanten hinzugefügt und Attribute der Graphelemente beeinflusst werden. Ebenso kann im Suchmuster auf Attribute zugegriffen und auf negative Anwendungsbedingungen (NAC) zurückgegriffen werden:

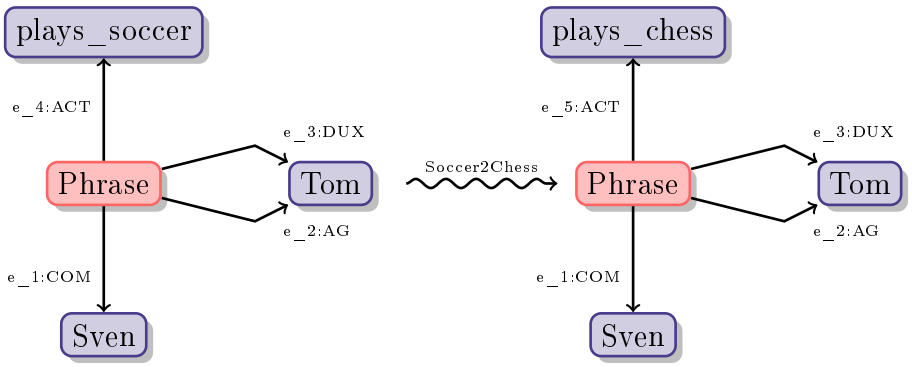

Abbildung 2.3.: Beispiel – Einfacher Satz mit *dux* und *comes*.

Listing 2.3: Beispiel für je eine GrGen.NET-Regel mit *replace* und *modify*.

```
rule playSoccer2playChess_mod () {
  /* Suchmuster */
  a:PHRASE-e_1:ACT->actus_alt:CONSTITUENT;

  /* Anwendungsbedingung */
  if { e_1.VISITED == false; }

  /* Modifikationsteil - Nicht genannte Knoten verbleiben im Graphen */
  modify{
    /* Löschen eines Knotens und einer Kante */
    delete(actus_alt);
    delete(e_1);

    /* Anlegen eines neuen Knotens */
    actus_neu:CONSTITUENT;

    /* Anlegen einer neuen Kante */
    a-e_2:ACT->actus_neu;

    /* Zugriff auf Attribute von Knoten */
    eval { actus_neu.NAME = "plays_chess"; }
  }
}

rule playSoccer2playChess_rep () {
  a:PHRASE-e_1:ACT->actus_alt:CONSTITUENT;

  /* Negative Anwendungsbedingung */
  negative { a-:HAB->:CONSTITUENT; }

  /* Ersetzungsteil - Nicht genannte Knoten werden aus dem Graphen gelöscht */
  replace {
    actus_neu:CONSTITUENT;
    a-:ACT->actus_neu;

    eval { actus_neu.NAME = "plays_chess"; }
  }
}
```

Die erste Regel `playSoccer2playChess_mod` entfernt die beiden Graphelemente `e_1` und `actus_alt` explizit und setzt neue dafür ein; das Ergebnis der Anwendung der zweiten Regel `playSoccer2playChess_rep` ist dasselbe – die Elemente `e_1` und `actus_alt`

werden aus dem Ergebnisgraphen gelöscht, da sie im Ersetzungsteil der Regeln nicht vorkommen. [BG07, 13] Die `if`-Bedingung der ersten Regel hat keine Auswirkung auf die Anwendbarkeit der Regel, da das geprüfte Attribut im Beispiel den Wert false einnimmt. Dasselbe gilt für die NAC der zweiten Regel, da die gesuchte Rolle *habitum* (HAB) im Graphen nicht vorkommt. Der eval-Ausdruck des Ersetzungs- bzw. Modifikationsteils einer Regel erlaubt das Verändern von Attributen der Graphelemente. Die einfachste Regel zum Wechseln der Sportart wäre somit:

Listing 2.4: Beispiel für eine GrGen.NET-Regel zur Veränderung eines Knotens.

```
1 rule playSoccer2playChess () {
2   a:PHRASE-e_1:ACT->actus:CONSTITUENT;
3
4   modify{
5     /* Alle Knoten behalten - nur den Wert ändern */
6     eval { actus.NAME = "plays_chess"; }
7   }
8 }
```

Für eine ausführliche Beschreibung von GrGen.NET, siehe [BG07] und [GBG+06].

2.4. Bewertung von Modellen

Es liegt auf der Hand, dass die domänenspezifischen Modelle, die den Grundstein für den Prozess des Model Driven Development (MDD) legen, höchsten Ansprüchen bezüglich ihrer Qualität und Korrektheit genügen müssen.

Um die Qualität eines Modells beurteilen zu können, muss zunächst ein geeignetes Qualitätsmaß definiert werden. Im 3. Kapitel werden unterschiedliche Bewertungsansätze vorgestellt und ihre Grundlagen und Umsetzungen besprochen.

3. Verwandte Arbeiten und Theorien

Die Qualitätsbwertung für Softwaremodelle ist ein breites Forschungsgebiet. In der Literatur werden unterschiedliche Ansätze verfolgt: Viele Autoren zielen auf modellinhärente Qualitätsmerkmale ab und berücksichtigen nicht, in wie weit sich die Modelle mit den Anforderungsdokumenten decken. Andere Ansätze betrachten die Qualität im Licht der verwendeten Modellierungssprache: Untersucht wird, in wie weit der verwendete Modellierungsansatz für die Problemstellung geeignet ist. Hierbei spielen insbesondere Faktoren wie die Zielgruppe eine große Rolle.

Aranda et al. führen an, dass die Effektivität von Modellen im Allgemeinen von verschiedenen Qualitätsmerkmalen abhängt. So führen sie nicht nur die Verständlichkeit auf, sondern stellen auch auf den „speed of decay“ [1] der Modelle, die Schwierigkeit beim Erlernen der Modellierungssprache sowie die Modellerstellungskosten ab. Sofern Modellierungssprachen bei mehreren dieser Punkte Defizite aufweisen, spielen Ausdrucksmächtigkeit und eine klare, formale Definition der Sprache für den Einsatz in der, sowie der Bedeutung für die Praxis nur noch eine untergeordnete Rolle. [AEHE07]

3.1. Qualitätsbewertung bei der modellgetriebenen Entwicklung

Mohaghegi und Aagedal geben in [MA07] einen Überblick über die Qualitätsbewertung bei der modellgetriebenen Entwicklung. Modelle werden in Zukunft inkrementell entwickelt und zusammen mit anderen Modellen betrachtet werden. Allgemeine Modellierungssprachen wie UML werden mit domänenspezifischen Sprachen kombiniert werden. Die Herausforderungen komplexer Systementwürfe (wie die Wiederverwendbarkeit von Modellteilen) werden in der modellgetriebenen Entwicklung aufgegeriffen werden müssen. Um die dabei entstehenden Probleme lösen zu können, müssen Modelle modular aufgebaut sein und zwischen verschiedenen entwicklungsunterstützenden Programmen ausgetauscht werden können. Die Qualitätsanforderungen sind also bei der modellgetriebenen Entwicklung nicht nur an die Modelle selbst, sondern auch an die Modellierungssprache zu stellen. Da bei der modellgetriebenen Entwicklung Modelle nicht nur

[1] Der „speed of decay“ ist die Geschwindigkeit, mit der sich das modellierte System von den zugehörigen Modellen entfernen.

der Kommunikation dienen, sondern in weitere Modelle und letztendlich in Quelltexte überführt werden, hängt die Qualität des fertigen Produktes direkt von der Qualität der Modelle ab. Diese Qualität wird von verschiedenen Faktoren wie beispielsweise Komplexität, Ausgewogenheit und Vollständigkeit der Modellierung bestimmt. Ihre zukünftige Forschung konzentriert sich auf die Suche nach Antworten auf Fragen wie *Ist die Modellierungssprache X für einen bestimmten Einsatz geeignet?* und *Ist ein Modell vollständig und bereit für die (automatische) Transformation?*.

3.2. Vollständigkeitsanalysen

Stirewalt et al. beschäftigen sich in [SDC05] mit der vollständigen Abbildung von UML-Modellen auf Quelltextsegmente. Hierbei versuchen sie die Modellelemente im Quelltext zu identifizieren. Durch die Verknüpfung von Quelltext und Modellen soll eine automatische Modellanalyse und -verfikiation ermöglicht werden. Probleme bezüglich der Verifikation entstehen bei diesem Ansatz, da es keine feste Umsetzung von UML-Elementen in den Programmiersprachen gibt. In [DSC04] stellen sie eine Methode vor, bei der mittels automatischer Quelltexterzeugung umgesetzte und fehlende Elemente im Quelltext identifiziert werden können. Selbst wenn umgekehrt die Rückverfolgbarkeit der Quelltext-Segmente zu den Modellelementen möglich ist, ist es jedoch zusätzlich notwendig, eine korrekte Abbildung der Anwendungsdomäne auf die Modelle sicherzustellen.

Settimi et al. fanden heraus, dass es mittels Information Retrieval (IR) Systemen einfacher ist, Verbindungen zwischen Anforderungen und UML-Modellen herzustellen als zwischen Anforderungen und Quelltext. [SCHM+04] Im Laufe ihrer Untersuchungen zeigte sich jedoch, dass das verwendete IR-System verschiedene Blickwinkel der selben Aktion nicht verbinden konnte (beispielsweise „der Benutzer sieht“ und „das Display zeigt“). Dieses Problem wird durch die in der vorliegenden Arbeit vorgestellte Methode ebenfalls nicht gelöst, jedoch kann geprüft werden, ob beide Blickwinkel modelliert wurden; insbesondere ist es möglich die Klasse(n) zu benennen, welche die betreffende Funktionalität implementiert bzw. implementieren (sollten).

Lange und Chaudron erkennen in [LC04], dass Metriken im Softwareentwicklungsprozess meist erst auf Ebene des Quelltextes angewendet werden und dass dies oft zu spät ist. Vollständigkeit sehen sie als kritischen Faktor an und unterteilen sie in „Requirements Completeness“ und „Model Completeness“. Erstere beschäftigt sich mit der Kundenperspektive (Sind alle Anforderungen umgesetzt?), letztere mit der Entwicklerperspektive (Ist das UML-Modell vollständig?). Da UML viele Diagrammtypen umfasst und keine strikte Semantik besitzt, können zwischen den einzelnen Diagrammen eines Modells leicht Inkonsistenzen auftreten. Lange und Chaudron richten ihre Aufmerksamkeit auf die Entdeckung und Vermeidung dieser Inkonsistenzen. Die erste Komponente ihrer Modellanalyse ist die Überprüfung der Wohlgeformtheit, welche sich nur auf jeweils ein einzelnes Diagramm bezieht. Die Zweite ist die Konsistenzprüfung, welche verschiedene, sich inhaltlich überlappende Diagramme auf widersprüchliche Informationen untersucht.

Die Letzte ist die Vollständigkeitsprüfung. Sie soll sicherstellen, dass Informationen überlappender Diagrammbereiche vollständig in beiden Diagrammen enthalten sind. Inkonsistenzen und Unvollständigkeiten können jedoch nicht immer unterschieden werden.[2]

3.3. Empirische Untersuchungen

In [LC06] beschreiben Lange und Chaudron zwei Experimente mit Studenten und professionellen Entwicklern, die durchgeführt wurden, um die Auswirkungen von verschiedenen Modelldefekten auf die resultierende Implementierung zu untersuchen. Untersucht wurden unter anderem „Nachrichten im Sequenzdiagramm ohne zugehörige Methode im Klassendiagramm“ und „Nachrichten, die in die falsche Richtung gesendet werden“. Ihre Ergebnisse zeigen, dass die meisten der untersuchten Defekte nur von weniger als der Hälfte der Teilnehmer entdeckt werden. Viele Fehler führen bei den Teilnehmern zu unterschiedlichen Interpretationen wie beispielsweise das Fehlen einer Klasse im Sequenzdiagramm. Lange und Chaudron fordern Richtlinien für die Erstellung von UML-Modellen um Fehlinterpretationen vorzubeugen.

Lange et al. stellen fest, dass obwohl Modelle oft die ersten Artefakte im Entwicklungsprozess sind, ihre Bewertung mithilfe von Metriken der Industrie immer noch Schwierigkeiten bereitet. In [LCM06] beschreiben sie eine Umfrage und eine industrielle Fallstudie, in denen sie sich mit der Verwendung von UML-Modellen in der Praxis beschäftigen. Bei über 50 Prozent der Befragten ist die Vollständigkeit der Modelle das entscheidende Kriterium zum Beenden der Modellierungstätigkeiten im Entwicklungsprozess. Im Gegensatz dazu steht ein Mangel an objektiven Kriterien zur Bewertung der Vollständigkeit; der Wunsch nach dem Einsatz von Metriken ist der Umfrage zufolge zwei- bis viermal so hoch wie der Tatsächliche. Darüber hinaus entstehen etwa die Hälfte der berichteten Missverständnisse zwischen den am Projekt beteiligten Personen, wenn der Entwicklungsprozess mit unvollständigen Modellen weitergeführt wird. Dies kann dazu führen, dass Kundenanforderungen nicht umgesetzt werden und der Aufwand zum Testen der Software stark ansteigt. Lange et al. fordern die Entwicklung von Methoden und Hilfsmitteln, welche die angesprochenen Probleme in Angriff nehmen.

Laitenberger et al. stellen in [LASE00] einen experimentellen Vergleich von verschiedenen Lesetechniken für die Inspektion von UML-Modellen vor. Hierbei vergleichen sie insbesondere Prüflisten und perspektivenbasiertes Lesen. Die verwendeten Prüflisten und Perspektiven beziehen sich auf allgemeine Fehler[3] und Inkonsistenzen[4]. Hierbei zeigte sich, dass Perspektiven eine ca. 40% höhere durchschnittliche Effektivität und dabei

[2] Beispielsweise: Fehlt eine Methode `methodX` im Klassendiagramm, die in einem Aktivitätsdiagramm verwendet, im Klassendiagramm jedoch nicht aufgeführt wurde, oder handelt es sich um einen Widerspruch?

[3] Beispielsweise: „Sind alle (Klassen-) Namen eindeutig?“.

[4] Beispielsweise: „Sind alle Klasseninstanzen in den Kollaborationsdiagrammen [...] mit ihren Definitionen in den Klassendiagrammen konsistent?“.

einen Kostenvorteil von über 50% gegenüber Prüflisten aufweisen. Im Gegensatz zu den hierbei verwendeten allgemeinen Prüflisten, wird in der vorliegenden Arbeit eine Prüfliste entwickelt, welche Fragen zum konkret vorliegenden Modell enthält.

3.4. Qualitätsmodelle und Qualitätssicherungsmaßnahmen

Basierend auf Ihren Untersuchungen stellen Lange et al. in [LC05] ein Qualitätsmodell vor, das speziell auf Modelle zugeschnitten ist. Wie in Abbildung 3.1 zu sehen ist, handelt es sich um ein dreistufiges Qualitätsmodell das einem Factor-Criteria-Metrics-Modell ähnelt. Auf der obersten Ebene zielt es auf die Verwendung von Modellen ab: Die (System-) Entwicklung und die Wartung. Unter diesen beiden Zielen stehen deren Elemente wie Analyse, Implementierung, Verstehen und so weiter. Auf der untersten Ebene stehen Regeln und Metriken, die helfen sollen, eine möglichst hohe Qualität zu erreichen bzw. die aktuell erreichte Qualität zu bestimmen. Vollständigkeit als Metrik spielt in diesem Modell eine wichtige Rolle für viele Elemente der Entwicklung.

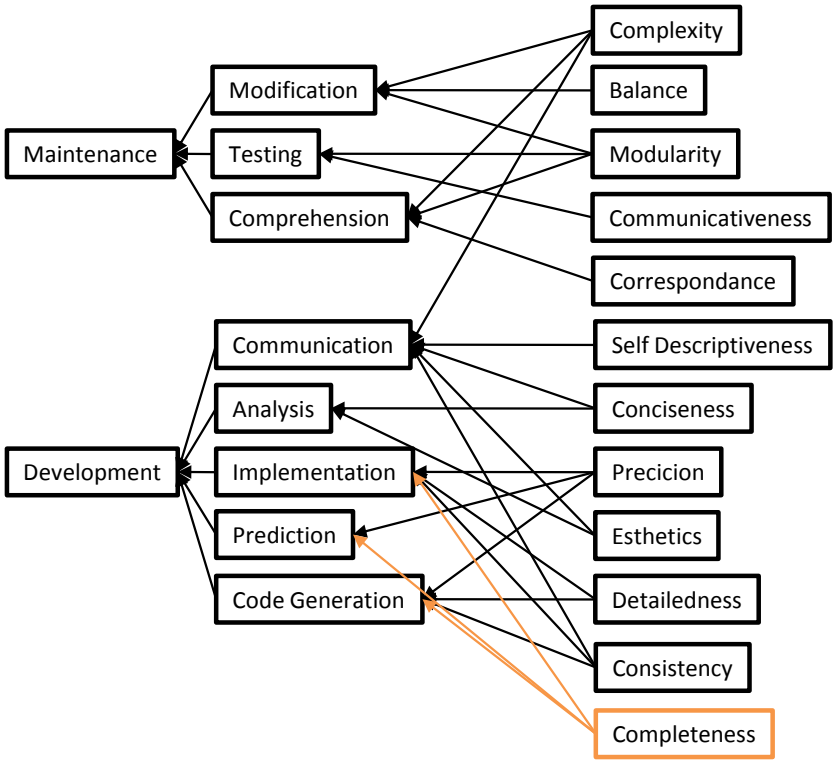

Abbildung 3.1.: Factor-Criteria-Metrics-Modell nach [Lan06].

Lindland et al. beschreiben in [LSS94] ein System zur Bewertung von UML-Modellen, welches auf semiotischen Konzepten basiert. Ausgehend davon, dass für die Situation, die Domäne und die Zielgruppe eine geeignete Modellierungssprache gewählt wurde, bewerten sie Modelle anhand der semiotischen Konzepte der Syntax, der Semantik und der Pragmatik: Einfach zu beurteilen ist die syntaktische Qualität – wenn das Modell den Regeln der Modellierungssprache entspricht, ist es qualitativ hochwertig. Eine hohe pragmatische Qualität ist erreicht, wenn das Modell von der Zielgruppe verstanden werden kann beziehungsweise verstanden wird. Die semantische Qualität erinnert an die Definitionen von Recall und Precision: Je ähnlicher das Modell der Domäne ist, desto besser – oder anders: Je weniger der Domäne im Modell fehlt und je weniger überflüssige Dinge modelliert wurden, desto besser. Da eine perfekte Modellierung nicht erreicht werden kann, sieht das Bewertungssystem vor, nur so weit zu modellieren, wie es sinnvoll ist: Ein Systemelement muss ins Modell aufgenommen werden, solange der Aufwand zur Aufnahme kleiner ist, als der Nachteil, es nicht im Modell zu haben. Ebenso ist es möglich überflüssige Informationen im Modell zu belassen: Ein ungültiges (nicht in der Domäne existierendes) Systemelement darf im Modell verbleiben, solange das Entfernen aufwändiger ist, als der Schaden der durch sein Verbleiben im Modell entsteht. Ein derart entstandenes, „optimales“ Modell bezüglich der Semantik ist folglich nur ein Kompromiss. Ziele wie Eindeutigkeit, Konsistenz und ein möglichst kompaktes Modell werden ebenfalls betrachtet, jedoch auf Vollständigkeit und Gültigkeit zurückgeführt.

4. Erstellen der Prüfliste

Die in der Einführung und dem 3. Kapitel beschriebene Aufgabe der Vollständigkeitsbeurteilung von UML-Modellen soll mit der in dieser Arbeit entwickelten Prüfliste methodisch durchführbar gemacht und vereinfacht werden. Entwickelt wurde ein System von Graphersetzungsregeln, mit dem eine Prüfliste erzeugt werden kann. Abbildung 4.1 zeigt den Arbeitsablauf mit dem fertigen Softwaresystem: Zunächst muss die Domänenbeschreibung annotiert werden (Schritt 1). Anschließend wird vom SAL_E-Übersetzer ein Spezifikationsgraph erzeugt (Schritt 2). Dieser kann in GrGen.NET geladen und mit den entwickelten Regeln verarbeitet werden (Schritt 4). Am Ende des Prozesses entsteht eine nach den Sätzen der Spezifikation sortierte Prüfliste.

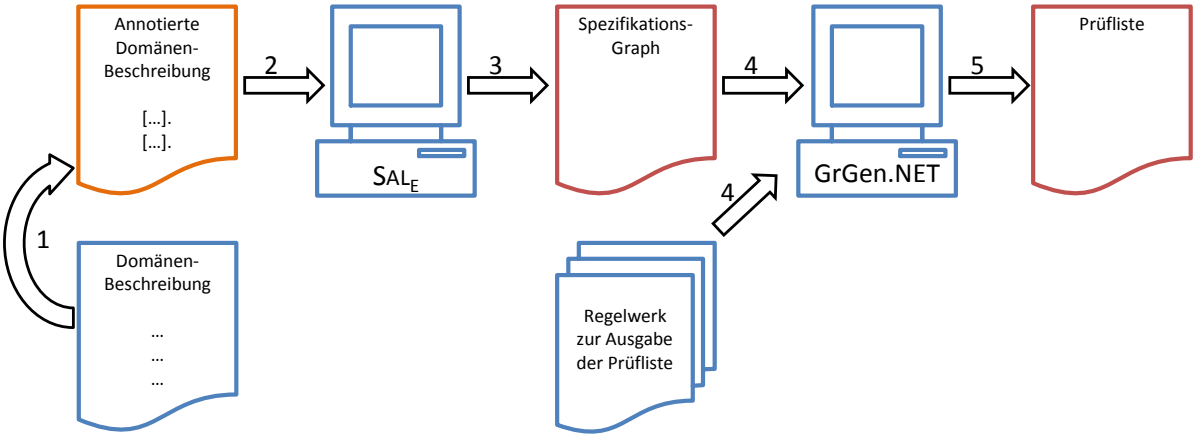

Abbildung 4.1.: Ablauf der Prüflistenerzeugung.

Die resultierende Prüfliste soll für alle Elemente der Spezifikation, sowie für die genannten Beziehungen und Aktionen je eine Frage enthalten. Die möglichen Antworten sollen für den Benutzer vorbereitet sein, sodass er das Element nur noch im vorliegenden Modell finden muss. In Tabelle 4.1 ist ein Beispiel für eine derartige Prüfliste zu sehen. Zur Veranschaulichung wurde die Tabelle bereits ausgefüllt. Die Prüfliste enthält für jeden Satz der Spezifikation einen Abschnitt, der mit „Phrase:“ beginnt. In jedem Abschnitt befinden sich die Fragen zu den jeweiligen Elementen des Satzes; für den ersten Satz sind die Elemente *opponents* und die Multiplizität 2 stellvertretend aufgeführt. Das gesuchte Element *opponents* wurde vom Korrektor weder als Klasse, noch als Rolle, noch als Instanz gefunden; stattdessen wurde vom Modellierenden notiert, dass die

Klassen Opponent und Player zusammengefasst wurden und deswegen keine Klasse für Opponent angelegt wurde.

Tabelle 4.1.: Beispiel für einen mit SAL$_E$ erzeugten Fragebogen.

Element of Phrase	**Modeled...**	**yes**	**no**	**wrong**
Phrase: The game of chess is played between two opponents.				
"opponents"	as class		✗	
	as role		✗	
	as instance		✗	
Differently (please specify):				
Explicitly not modeled (please specify): Opponent- and Player-classes are equivalent...		✗		
Multiplicity "2" of "opponents"	as multiplicity		✗	
	in an OCL constraint			✗
Differently (please specify):				
Explicitly not modeled (please specify):				
~~	*etc.*			~~~~~~~~~~
Phrase: The player with the white pieces commences the game.				
"player"	as class	✗		
	as role		✗	
	as instance	✗		
Differently (please specify):				
Explicitly not modeled (please specify):				
Attribute "white" of "pieces"	as Boolean function (with a parameter)	✗		
	as scalar function		✗	
	as state of "pieces"		✗	
	as attribute of "pieces"	✗		
Differently (please specify):				
Explicitly not modeled (please specify):				
~~	*etc.*			~~~~~~~~~~

Als Vollständigkeitsmaß kann nun die Anzahl der modellierten Elemente zur Anzahl der zu modellierenden Elemente gesetzt werden. Hieraus ergibt sich eine prozentual angegebene Vollständigkeitsbeurteilung des Modells bezüglich der Spezifikation:

$$\text{Recall} = \frac{|\{\text{Elemente des Modells}\} \cap \{\text{Elemente der Spezifikation}\}|}{|\{\text{Elemente der Spezifikation}\}|}$$

Das erste Unterkapitel führt zunächst in den im Folgenden verwendeten Spezifikationstext ein. Im darauf folgenden Unterkapitel wird die Abbildung von SAL$_E$-Strukturen auf UML-Elemente und somit auf Elemente der Prüfliste durchgeführt. Anschließend wird besprochen, welche Rollen zum bestehenden Rollenumfang hinzugefügt werden mussten und welche Probleme beim Annotieren eines Textes auftreten können. Daraufhin werden die Erweiterungen für das GrGen.NET-Graphenmodell erläutert und die Vorbereitung des Spezifikationsgraphen für die Weiterverarbeitung erklärt. Das abschließende Unterkapitel behandelt die Erstellung der GrGen.NET-Regeln für die Ausgabe der Prüfliste im XML-Format.

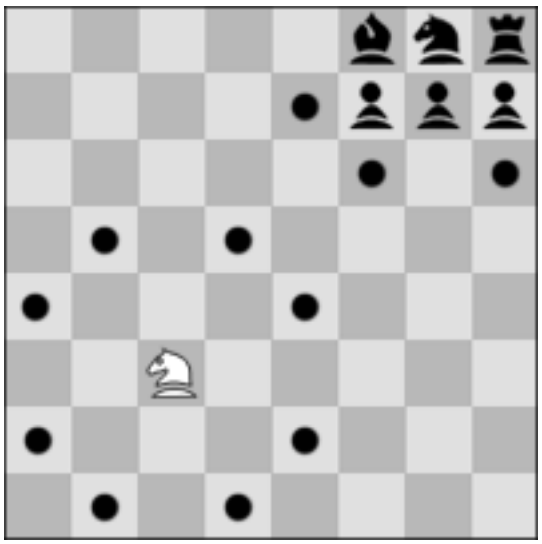

Abbildung 4.2.: Mögliche Züge des Springers aus [WCF04].

4.1. Einführung in das laufende Beispiel

Im Verlauf dieses Kapitels und der Evaluation in Kapitel 5 wird immer wieder auf denselben Spezifikationstext zurückgegriffen: Die offiziellen Schachregeln der FIDE (FIDE Laws of Chess [WCF04, Abschnitt E.I.01A.]).

Die Spezifikation beschreibt in den ersten 5 Artikeln die gängigen Schachregeln. Diese behandeln die Anfangsaufstellung, das Ziel, und die möglichen Ausgänge des Spieles sowie die erlaubten Züge der Figuren. Bei den darauf folgenden Artikeln handelt es sich um die Turnierregeln, welche den Umgang mit der Schachuhr, das Verhalten der Spieler am Austragungsort und die Strafen für Regelverstöße vorschreiben. Den Abschluss bildet ein Abschnitt über die Organisation der FIDE und ihrer Mitgliedsorganisationen.

Um eine Evaluation der Methode im Rahmen der Softwaretechnikvorlesung zu ermöglichen, sollte eine nicht zu kurze, aber auch nicht zu umfangreiche Spezifikation gewählt werden. Die Wahl fiel auf die Schachregeln, da diese über einen großen Bekanntheitsgrad verfügen und ein möglichst anschauliches Beispiel gewählt werden sollte. Darüber hinaus enthalten die Schachregeln nur einen geringen Anteil von Informationen, die nicht im domänenspezifischen Modell enthalten sein sollen. Im Verlauf der Arbeit zeigte sich, dass aus nicht-technischen Beschreibungen und den daraus resultierenden Grammatikstrukturen Probleme bei der Annotation entstehen können; diese und die möglichen Lösungen werden in Abschnitt 4.3 beschrieben. Der große Bekanntheitsgrad der Spezifikation stellte sich insbesondere bei der Bearbeitung durch die Studenten auch als Nachteil heraus (siehe dazu Abschnitt 5.3.1).

Phrase: The game of chess is played between two opponents|AG.	
Element of Phrase	**Modeled...**
„opponents“ (*agens*)	as class as role as instance

Tabelle 4.2.: Mögliche Umsetzung von Objektrollen (*agens*).

Phrase: The game of chess is played|ACT between two opponents.	
Element of Phrase	**Modeled...**
„played“ (*actus*)	as method in a state chart (as state or as transition) as relation between two classes

Tabelle 4.3.: Mögliche Umsetzung von Methodenrollen (*actus*).

4.2. Identifikation zugehöriger UML-Entitäten zu SAL$_E$-Strukturen

Der Satz *The game_ of_ chess/PAT is_ played/ACT between two opponents/AG who move their pieces alternately on a square board called a 'chessboard'.* macht deutlich, dass einige SAL$_E$-Elemente nicht alleine auf eine Entität in einem Diagramm abgebildet werden können. Der *agens* opponents sollte einmal im Klassendiagramm zu finden sein (dort als Rolle von Player). Darüber hinaus sollte es auch zwei Instanzen der Klasse Player/Opponent geben, die in einem Sequenzdiagramm verwendet werden um das abwechselnde Ziehen zu beschreiben. Ein anderer Modellierungsansatz beschreibt das abwechselnde Ziehen nicht, sondern schränkt die Methode zum Ziehen ein – sei es über einen OCL-Ausdruck oder einen Kommentar.

Das Problem, dass die Abbildung von UML nach Quelltext nicht eindeutig ist, setzt sich bei der Abbildung von Spezifikationstext nach UML fort. Folglich müssen verschiedene Modellierungen zugelassen werden. Diese sollten nach Möglichkeit nicht als Freitext in die Korrekturliste eingetragen, sondern direkt als mögliche Antwort vorgesehen werden.

Im ersten Schritt wurden die thematischen Rollen als solche auf entsprechende UML-Elemente abgebildet. In die Gruppe der Rollen, die im fertigen Modell als Objekt, Rolle oder Instanz zu finden sein sollten, fallen die Rollen *agens*, *beneficiens*, *causa*, *comes*, *comparius*, *compariens*, *contratius*, *contrariens*, *creator*, *donor*, *dux*, *experior*, *fautor*, *favor*, *fictum*, *fingens*, *habitum*, *instrumentum*, *omnium*, *opus*, *pars*, *patiens*, *possessor*, *recipiens*, *substitutus*, *substituens*, *thema* sowie *thematiens*. Tabelle 4.2 enthält ein Beispiel für eine derartige Rolle.

Die Elemente mit der Rolle *actus* sollten im fertigen Modell als Methode zu finden sein; in der Prüfliste erwarten wir eine Frage nach der Existenz einer Methode. Ebenso könnte das Element in einem Zustandsautomaten als Zustand oder Zustandsübergang oder als Beziehung zwischen zwei Klassen zu finden sein. Tabelle 4.3 enthält ein Beispiel für ein derartiges Element.

Phrase: Mathias has *2 $red cars.	
Element of Phrase	**Modeled...**
Multiplicity „2“ of „cars“ (*)	as multiplicity in an OCL contraint
Attribute „red“ of „cars“ ($)	as Boolean function (with parameter) as scalar function as state of „cars“ as attribute of „cars“

Tabelle 4.4.: Mögliche Umsetzung von Multiplitzitäten und Attributen.

Phrase: Tom|DON gives|ACT Mathias|RECP an apple|HAB.	
Element of Phrase	**Modeled...**
„gives“ (*actus*)	as method of „Tom“ as method with according parameters for „Tom“, „Mathias“ and „apple“ as classes with associations

Tabelle 4.5.: Mögliche Umsetzung von Kombinationen (*donor* + *recipiens* + *habitum*).

Mulitpizitäten und Attribute von Elementen werden ebenfalls berücksichtigt. Hierzu wird unterschieden, ob es sich um ein Attribut zu einem Objekt oder einer Methode handelt, um entsprechende Formulierungen zu verwenden. Ein Beispiel findet sich in Tabelle 4.4.

Anschließend wurden Rollenkombinationen betrachtet, die anhand der thematischen Liste aus [Geb06] ersichtlich sind. An dieser Stelle werden die Rollen *donor*, *recipiens*, *habitum* und *actus* stellvertretend betrachtet. Sie kennzeichnen jeweils ein Objekt, die Rolle des *actus* eine Handlung/Methode. In Kombination führen Sie zu einer Beziehung: Das *habitum* ist ein Objekt, welches vom *donor* zum *recipiens* gegeben wird; infolgedessen, erwarten wir im Modell eine Methode, die diesen Vorgang widerspiegelt: Die Methode, welche aufgrund des *actus* angelegt wurde, sollte mit ausreichend vielen Parametern ausgestattet sein, so dass die anderen Objekte übergeben werden können. Außerdem kann die Beziehung mit einer dreistellige Assoziation (mit Assoziationsklasse) dargestellt werden. Die möglichen Umsetzungen sind in Tabelle 4.5 zusammengefasst, (für weitere Kombinationen von thematischen Rollen siehe Tabelle A.2).

Unter Umständen enthält die Spezifikation Elemente, die bewusst nicht modelliert werden (sollen). Entdeckt der Modellierende ein solches Element, kann er von der Modellierung absehen und dies notieren. Bei der Überprüfung des Modells kann dieser Hinweis im Fragebogen vermerkt werden („explicitly not“ in Tabelle 4.1). Zuletzt sollen andere Modellierungsvarianten nicht ausgeschlossen werden. Hierzu wurden zu allen Fragen die Möglichkeit hinzugefügt, eine abweichende Modellierung zu vermerken („differently“ in Tabelle 4.1).

4.3. Allgemeine Schwierigkeiten beim Annotieren

Im Laufe der Annotation der Schachregeln ergaben sich neue Anforderungen an den Text, die Annotationssprache SAL$_E$ und den SAL$_E$-Übersetzer. Die folgenden Abschnitte fassen die aufgetretenen Probleme und die entwickelten Lösungen zusammen. Eine aktuelle Liste mit allen in SAL$_E$ verfügbaren thematischen Rollen befindet sich in Tabelle A.1.

4.3.1. Auflösen von unechten Referenzen

Wie in der Einführung beschrieben, können aufeinander folgende wiederholte Elemente in der Spezifikation mit einem `@` gekennzeichnet werden. Im entstehenden Graphen wird anstelle zweier „gleicher“ Knoten nur der erste Knoten erzeugt und die hinzugekommenen Kanten zu diesem hinzugefügt. Der Graph in Abbildung 2.2 zeigt den Unterschied anschaulich. Da der SAL$_E$-Übersetzer die Referenz aufgrund der textuellen Repräsentation erzeugt, kann diese Funktion nicht verwendet werden, wenn die beiden Nennungen unterschiedlich sind. Dies entsteht im (im Englischen seltener, im Deutschen oftmals) durch die Biegung eines Substantives oder die Verwendung von Synonymen. Häufiger – auch bei englischen Texten – werden Personalpronomen in Sätzen verwendet, wenn der Name eines Subjektes oder Objektes nicht wiederholt werden soll. Im Satz *The opponent whose king has been checkmated has lost the game.* tritt dieses Problem in Form des *whose* auf; eine Annotation mit `@whose` würde im ersten Fall zu einem Fehler führen, da *whose* noch nicht verwendet und der Zielknoten für die Referenz noch nicht angelegt wurde. Unterstellt man, dass im vorangegangenen Spezifikationstext bereits ein *whose* aufgetreten ist, führt eine Annotation mit `@whose` zu einer großen Fehlinterpretation: Die Referenz zeigt dann auf das falsche Element. Unterlässt man eine derartige Annotation, würde man im Ergebnis unerwünschte Fragen erhalten, beispielsweise ob das Objekt *whose* als Klasse modelliert wurde.

Um in derartigen Fällen eine sinnvolle Frage erzeugen zu können (oder die Frage zu unterlassen), wurden zwei neue Rollen eingeführt: EQK und EQD. EQ steht hierbei für „equals“. Die Rolle EQK wird verwendet um das Ziel der Referenz zu kennzeichnen. Das Ziel der Referenz soll im Graphen verbleiben, daher steht das K im Rollennamen für „keep“. Die Rolle EQD wird verwendet um den Ausgangspunkt der Referenz zu kennzeichen. Der Ausgangspunkt der Referenz soll aus dem Graphen entfernt werden, daher steht das D im Rollennamen für „drop“. Die Rollen werden nicht in den Spezifikationssatz eingefügt, sondern eine Hilfsphrase zum Text hinzugefügt. Diese benennt die beteiligten Elemente und gibt die Richtung der Referenz an:

Listing 4.1: Die Verwendung von EQD und EQK in SAL$_E$.

```
1 /* The opponent whose king has been checkmated has lost the game. */
2 [ #The @opponent|AG
3   [ whose|POSS king|{HAB, STATII} #has #been checkmated|STAT ]|SUM
4   #has lost|ACT #the @game|PAT ].
5
6 [ @whose|EQD @opponent|EQK ].
```

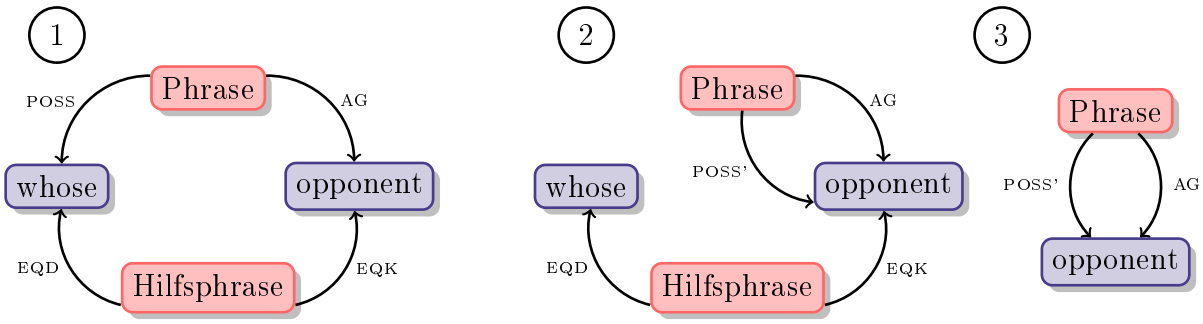

Abbildung 4.3.: Verschmelzen von gleichen Knoten.

Da es sich bei den eingefügten Rollen nicht um Rollen mit spezieller Funktion sondern nur mit spezieller Bedeutung handelt, erstellt der SAL$_E$-Übersetzer zunächst zwei Knoten für die genannten Elemente und fügt anschließend die Hilfsphrase ein. Der resultierende Teilgraph ist unter Punkt 1 in Abbildung 4.3 dargestellt.

Der resultierende Graph unterscheidet sich zunächst strukturell von dem, der mit einer Referenz aufgebaut wurde. Um diesen Unterschied zu auszugleichen, sind 4 Grapherersetzungsregeln notwendig. Diese können ohne weitere Vorbereitung auf dem Graphen ausgeführt werden und überführen ihn in die Form, die bei der Verwendung von Referenzen erreicht wird. Abbildung 4.3 zeigt die Arbeitsweise der Regeln. Zunächst muss eine Hilfsphrase gefunden werden. Das Suchmuster in Listing 4.2 leistet dies (1):

Listing 4.2: die Verwendung von EQK und EQD in GrGen.NET.

```
1 hilfsPhrase:PHRASE-eqKeep:EQK->nodeA:CONSTITUENT;
2 hilfsPhrase-eqDrop:EQD->nodeB:CONSTITUENT;
```

Somit können eingehende Kanten zu **nodeB** (*whose*) identifiziert und zu **nodeA** (*opponent*) neu aufgebaut werden (2). Gibt es (abgesehen von der EQD-Kante) keine eingehenden Kanten mehr zu **nodeB**, können sowohl die Hilfsphrase, als auch **nodeB** entfernt werden (3). Sobald das obenstehende Suchmuster nicht mehr im Graph vorhanden ist, hat der Graph die gewünschte Form und die weitere Verarbeitung kann erfolgen. Der vollständige Regelsatz namens **prepare_mergeEqualNodes** findet sich im Anhang im Listing E.2.

4.3.2. Referenzen und Teilmengenbeziehungen

Wie bei einfachen unechten Referenzen entsteht ein ähnliches Problem, wenn unechte Referenzen von mehreren Objekten auf ein Mengenobjekt zeigen sollen. Einfach darzustellen ist dieser Sachverhalt mit zwei Sätzen, wobei im ersten die Mengenelemente eingeführt werden. Im zweiten soll dann der Mengenbegriff verwendet werden.

Möchte man derartige Mengenbegriffe abbilden, eignen sich die Rollen EQD und EQK nicht, da dann die ursprünglichen Elemente entfernt werden. Aus diesem Grund wurden zwei weitere Gleichheitsrollen eingeführt: EQS und EQB. Erstere wird verwendet um

eine Menge zu kennzeichnen, letztere kennzeichnet ihre Mengenelemente. Die Vorverarbeitung des Graphen ist ähnlich zur Auflösung von unechten Referenzen, weswegen an dieser Stelle nicht weiter darauf eingegangen werden soll. Der nachstehende Quelltext verdeutlicht einen solchen Sachverhalt:

Listing 4.3: Die Verwendung von EQS und EQB in SALE.

```
1 /* Tom and Mathias play soccer */
2 [ { Tom AND Mathias}|AG play_soccer|ACT ].
3
4 /* They run fast */
5 [ They|AG run|ACT $fast<<1 ].
6 [ @They|EQS @Tom|EQB ].
7 [ @They|EQS @Mathias|EQB ].
```

Der vollständige Regelsatz namens `mergeEqualSets` findet sich im Anhang im Listing E.2.

4.3.3. Modalkonstruktionen

SALE war bei Beginn der Arbeit nicht in der Lage, Modalkonstruktionen zu verarbeiten. Um diese dreiwertige Beziehung (wer kann/darf/soll/muss was?) abbilden zu können, wurden die Rollen MODAL_WHO, MODAL_MOD und MODAL_WHAT in das Graphenmodell eingefügt. MODAL_WHO kennzeichnet das Subjekt, MODAL_MOD kennzeichnet das Modalverb, und MODAL_WHAT kennzeichnet die Satzaussage.

4.3.4. Verschieben von Attributen und Multiplizitäten

Bei Beginn dieser Arbeit bezogen sich Attribute und Multiplizitäten in SALE immer auf das darauf folgende Element, wie beispielswiese ein `$roter Schuh`. In vielen Texten stehen jedoch Attribute hinter dem Element oder gar an einer weiter entfernten Stelle. Um eine Annotation ohne die Umstellung des Satzes zu ermöglichen, wurde der SALE-Übersetzer erweitert um Attribute verschieben zu können. Gekennzeichnet durch `<<Zahl` oder `>>Zahl` können Attribute und Multiplizitäten nach vorne oder hinten verschoben werden: `#Der Schuh|ROLLE #ist $rot<<2`. [1]

4.3.5. Komplizierte Sprachkonstrukte

Komplizierte Sprachkonstrukte wie beispielsweise erklärende Relativsätze mussten entfernt werden um eine Annotation mittels SALE zu ermöglichen. So wurde beispielsweise der erste Satz des ersten Artikels folgendermaßen vereinfacht:

Listing 4.4: Vereinfachung von Relativsätzen.

```
1 /* 1.1. The game of chess is played between two opponents who move
2   their pieces alternately on a square board called a 'chessboard'. */
```

[1] Bei der Zählung werden alle Elemente (auch Kommentare) berücksichtigt.

```
3  /* The game of chess is played between two opponents. */
4  [ #The game_of_chess|PAT #is played|ACT #between *two opponents|AG ].
5
6  /* They alternatey move their pieces on a square board called a
7  'chessboard'.*/
8
9  [ They|AG $alternately move|ACT their|POSS ^pieces|{HAB,PAT} #on
10   [ #a $square ^board|{PAT,FIN} called|ACT #a chessboard|FIC ]|LOC_POS
11 ].
```

Wie im voranstehenden Beispiel zu sehen ist, wurde nicht der Informationsgehalt des Satzes verändert, sondern dieser nur in zwei gleichwertige Sätze aufgespalten.

Als Ergebnis dieses Teils der Studienarbeit entstanden die Annotierungen, die im Angang C und D zu finden sind.

4.4. Annotieren mit SAL_E

Da SALE für natürliche Sprache entwickelt wurde, mussten die graphischen Elemente aus dem Spezifikationstext entfernt werden. Bis auf die Anfangsaufstellung sind jedoch alle Informationen ebenfalls in Textform verfügbar, da die Abbildungen lediglich der Veranschaulichung dienen (wie beispielsweise Abbildung 4.2).

Die Anfangsaufstellung wurde gänzlich aus der Spezifikation entfernt; von einer textuellen Beschreibung der Abbildung wurde abgesehen, um die Spezifikation nicht zu verfälschen.

4.5. Überführung des annotierten Textes in GrGen.NET

Durch die Übersetzung durch den SALE-Übersetzer entstand zunächst ein unzusammenhängender Graph, in dem jeder Satz der Spezifikation einen isolierten Teilgraph darstellte. Ausgenommen hiervon waren lediglich die Elemente, die durch (echte oder unechte) Referenzen verbunden wurden. In einem unzusammenhängenden Graphen ist es unmöglich, die ursprüngliche Reihenfolge des Textes zu „raten“. Um den Graphen entsprechend der Reihenfolge des Spezifiaktionstextes durchlaufen zu können, wurden NEXT-Kanten eingeführt. Diese werden zur Übersetzungszeit vom Übersetzer zwischen die einzelnen Satz-Knoten eingefügt. Mittels dieser Kanten kann der (jetzt vollständig zusammenhängende) Graph einfach traversiert werden um die Fragen zu den jeweiligen Sätzen in der korrekten Reihenfolge zu erzeugen.

Bevor jedoch die Erzeugungsregeln auf den Graphen angewendet werden, sollen zunächst einige Prosaelemente entfernt und durch standardisierte Angaben ersetzt werden. Da im Ergebnis ein UML-Modell geprüft werden soll, sollen Elemente im Fragebogen möglichst so benannt werden, wie sie im Modell anzutreffen sind. Daher wurde ein kleiner Regelsatz geschrieben, der Multiplizitäten vereinheitlicht. Auf diese Weise können

Multiplizitäten im Fragebogen mit Ziffern angegeben werden anstatt als Wort: „1“ statt „one“ oder „One“ bzw. „0..*“ statt „any“.

4.6. Erzeugen von Regeln zur Ausgabe der Prüfliste

Ausgehend von den entstehenden Mustern im Graphen können nun Suchmuster erstellt werden, welche die identifizierten SAL$_E$-Strukturen im Graphen erkennen. Auf die gefundenen Knoten kann dann zugegriffen und insbesondere der Wert (das entsprechende Element aus der Spezifikation) ausgelesen werden.

Um eine möglichst flexible Ausgabe zu erhalten, wurden die Ausgaben der erstellten Regeln im XML-Format erzeugt. So ergibt sich eine einfache aber klare Struktur, welche unabhängig vom Ausgabedokument ist. Ein übersichtliches Korrekturschema kann einfach im HTML-Format erzeugt werden. Die Transformation in ein HTML-Dokument kann mit dem im Listing E.33 angegeben XML-Stylesheet explizit, oder implizit in einem Web-Browser beim Aufruf erfolgen.

Die Struktur des XML-Dokuments ist folgendermaßen:

Listing 4.5: Die XML-Struktur des Fragebogens.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Questionnaire.xsl"?>
<phrases>
  <phrase value="Originalfassung des Satzes ohne Annotationen">
    <element>
      <question>Nennung des Satzelements</question>
      <option>Modellierung</option>
      <option>Variante 1</option>
      <option>Variante ...</option>
      <error>ggf. Fehlermeldung</error>
      <note>ggf. Hinweise</note>
    </element>
  </phrase>
</phrases>
```

Wie in Tabelle A.2 zu sehen ist, erwartet man beim Auftreten eines Konstrukts mit einem *fingens* und einem *fictum* die folgenden Modellierungen:

- Eine Rolle „*fingens*“ der Klasse des *fictum*,
- eine Beziehung zwischen den Klassen von *fingens* und *fictum* oder
- eine Vererbungsbeziehung (die in der Regel falsch ist).

Die Regel mit der folgenden Suche identifiziert das Suchmuster im Graph

Listing 4.6: Ein Suchmuster für *fingens* und *fictum*.

```
rule finFic() {
  a:PHRASE-e_1:FIN->fingens:CONSTITUENT;
  a-e_2:FIC->fictum:CONSTITUENT;
  ...
}
```

und kann dementsprechend die folgende Ausgabe erzeugen:

Listing 4.7: XML-Ausgabe für *fingens* und *fictum*.

```
<!-- emitFINFIC(a) -->
<element>
  <question>
    Relationship between
    &quot;" + fingens.VALUE + "&quot; and
    &quot;" + fictum.VALUE + "&quot; modeled...
  </question>

  <option>using inheritance (usually wrong)</option>
  <option>as association</option>
  <option>as role</option>
</element>
<!-- end of emitFINFIC(a) -->
```

Die Eingabemöglichkeit für eine abweichende Modellierung („differently“ in Tabelle 4.1) oder eine Auslassung im Modell („explicitly not“ in Tabelle 4.1) ist für jedes Element auszugeben und kann dementsprechend bei der Anwendung des XML-Stylesheets hinzugefügt werden.

Das Identifizieren von Suchmustern, wie sie im Abschnitt 4.2 erläutert wurden, führt einfach zu den Ausgaben, die für die Erstellung des Fragebogens notwendig sind. Ein naives Suchen und Ausgeben ist jedoch nicht ausreichend, um die Korrekturliste in der richtigen Reihenfolge zu erzeugen.

Um dies zu erreichen, können die in Abschnitt 4.5 eingeführten NEXT-Kanten verwendet werden um den Graph in der korrekten Reihenfolge zu traversieren. GrGen.NET erlaubt das Parametrisieren von Regeln. Diese Funktionalität kann dazu verwendet werden um die gefundenen PHRASE-Knoten an die weiteren Ausgaberegeln zu übergeben:

Listing 4.8: Ermitteln des nächsten Satzes im Graphen.

```
rule emitNextSatz {
  act:PHRASE-e_next:NEXT->next:PHRASE;
  if {
    act.VISITED == true;
    e_next.VISITED == false;
    next.VISITED == false;
  }
  modify {
    eval{ e_next.VISITED = true; }
    exec(emitQuestions(next));
  }
}
```

Folglich kann der PHRASE-Knoten des aktuellen Satzes im Suchmuster verwendet werden und es werden nur noch Muster gefunden, die den aktuellen Satz enthalten:

Listing 4.9: Das Parametrisieren einer GrGen.NET-Regel.

```
rule finFic(aktuell:PHRASE) {
  aktuell-e_1:FIN->fingens:CONSTITUENT;
  aktuell-e_2:FIC->fictum:CONSTITUENT;
  ...
}
```

Um die oben beschriebene Struktur der Ausgabe zu erreichen, wird die folgende Regel verwendet:

Listing 4.10: Vollständige Regel für *fingens* und *fictum*.

```
1  /* Create questions for a single phrase */
2  rule emitQuestions(a:PHRASE) {
3    if { a.VISITED==false; }
4    modify {
5      eval {a.VISITED=true;}
6      emit ("<phrase value=\"" + a.VALUE + "\">\n");
7      exec ( finFic(a));
8      emit ("</phrase>\n");
9    }
10 }
```

Zusammen mit den öffnenden und schließenden Elementen in Zeile 1 und 15 ergibt sich die gewünschte Ausgabestruktur:

Listing 4.11: Vollständige Ausgabe für *fingens* und *fictum*.

```
1  <phrase value=" + a.VALUE + ">
2  <!-- emitFINFIC(a) -->
3  <element>
4    <question>
5      Relationship between
6      &quot;" + fingens.VALUE + "&quot; and
7      &quot;" + fictum.VALUE + "&quot; modeled...
8    </question>
9
10   <option>using inheritance (usually wrong)</option>
11   <option>as association</option>
12   <option>as role</option>
13 </element>
14 <!-- end of emitFINFIC(a) -->
15 </phrase>
```

Da SALE Subphrasen unterstützt, müssen diese ebenso behandelt werden, wie „normale“ Phrasen – das Suchmuster entspricht dem für Phrasen.

Wenn *gleiche* Elemente (also Instanzen der gleichen Klasse) nicht mit einer Referenz gekennzeichnet wurden, da es sich nicht um *dieselbe* Entität (Instanz) handelt, werden bei dieser Herangehensweise Fragen mehrfach ausgegeben. Um dies zu vermeiden, wird nach jedem Durchlauf die transitive Hülle der besuchten Knoten berechnet und die anderen gleichen Elemente ebenfalls als besucht markiert:

Listing 4.12: Berechnen der transitiven Hülle eines besuchten Knotens in GrGen.NET.

```
1  rule createTransitiveClosureForEqualNodes {
2    /* Suche ein Objekt in einer Phrase */
3    a:PHRASE-e:OBJECTROLE->x:CONSTITUENT;
4
5    /* und ein weiteres irgendwo im Graph */
6    y:CONSTITUENT;
7
8    /* Wenn X schon besucht wurde und Y den selben Wert hat, sind die Objekte gleich
9     * und die Fragen für die Objektrollen wurden schon gestellt */
10   if {
11     x.VISITED == true;
```

```
12      x.VALUE == y.VALUE;
13    }
14
15    /* Wenn es schon eine Kante gibt, erstelle keine weitere */
16    negative { x-alreadyCreated:EQUALNODES->y; }
17
18    /* Anlegen der Kante für die transitive Hülle */
19    modify {
20      x-:EQUALNODES->y;
21      emit("<!--Inserting edge for transitive closure between "
22          + x.NAME + " and " + y.NAME + "-->\n");
23    }
24  }
```

Im Anhang E sind alle Ausgabe- und Vorbereitungsregeln abgedruckt.

Das hier entwickelte System von Regeln kann auch dazu verwendet werden, den SENSE-Prozess weiterzuentwickeln. Die XML-Datei kann mit einem anderen Stylesheet oder einem interaktiven Prozess dazu verwendet werden, ein Modell zu erzeugen: Anstatt in einem fertigen Modell die Elemente zu suchen, könnte das Programm den Benutzer fragen, wie die einzelnen Elemente modelliert werden sollen und dann die entsprechenden Elemente in einem Modell anlegen. Hierzu könnte auch die von Bugra Derre entwickelte MOF-Suite [Der08] für GrGen.NET eingebunden werden, mit welcher in GrGen.NET angelegte UML-Modelle als standardkonforme XMI-Datei ausgegeben werden können. [GDG08]

5. Empirische Evaluation

Um den im 4. Kapitel erläuterten Ansatz zu evaluieren, wurde im Rahmen der Softwaretechnikvorlesung der Universität Karlsruhe eine Übungsaufgabe gestellt, welche mit dem entwickelten Korrekturschema korrigiert wurde. Um die Benutzbarkeit der Prüfliste und die Akzeptanz durch die Korrektoren nicht nur informell zu evaluieren, wurden zufällig ausgewählte Modelle von insgesamt 4 Korrektoren unabhängig bewertet und deren Ergebnisse verglichen.

5.1. Experimentaufbau

Im Rahmen der Softwaretechnikvorlesung im Hauptstudium des Informatiklehrplans wurde eine Modellierungsaufgabe gestellt, die von Studentenpaaren als bewertete Hausaufgabe bearbeitet werden sollte. Die Studenten wurden im Vorfeld in der Verwendung von UML geschult; eine der Experimentaufgabe vorangehenden Hausaufgaben war ebenfalls eine Modellierungsaufgabe. Die Korrektoren bewerteten die Modelle anhand der Prüfliste, nachdem sie in der Verwendung der Liste geschult wurden. Die Korrektoren wurden angewiesen, keine Punkte zugunsten der Studenten zu verschenken. Um eine konsequente Bewertung durch die Tutoren zu erreichen, wurde gegenüber dem Übungsbetrieb versichert, dass die erreichte Punktzahl der Studenten wohlwollend in Notenpunkte umgerechnet werden würden.

Als Übungsaufgabe wurden die ersten 5 Artikel der Schachregeln gewählt. Die Teilnehmer erhielten die im vorangegangenen Kapitel verwendete Fassung ohne graphische Elemente und ohne SAL$_{\mathsf{E}}$-Annotationen. Um die Studenten zu ermutigen, nicht nur ein Klassendiagramm zu erstellen, wurden Sie darauf hingewiesen, auch andere Diagrammtypen und OCL-Zusicherungen zu verwenden. Die verwendete Aufgabenstellung befindet sich im Anhang B.

Nach der Korrektur durch die Tutoren wurde ein Tutor zufällig ausgewählt und vier der ihm zugeteilten Übungsblätter zufällig ausgewählt. Diese Übungsblätter wurden von drei weiteren Korrektoren korrigiert. Hierdurch konnten pro Modell vier Bewertungen erhalten und miteinander verglichen werden.

5.2. Auswertung der Ergebnisse

Abbildung 5.1 zeigt die Korrekturergebnisse für die vier Modelle. Die jeweils eingeblendete horizontale Linie markiert den Durchschnittswert eines Modells über alle Korrektoren.

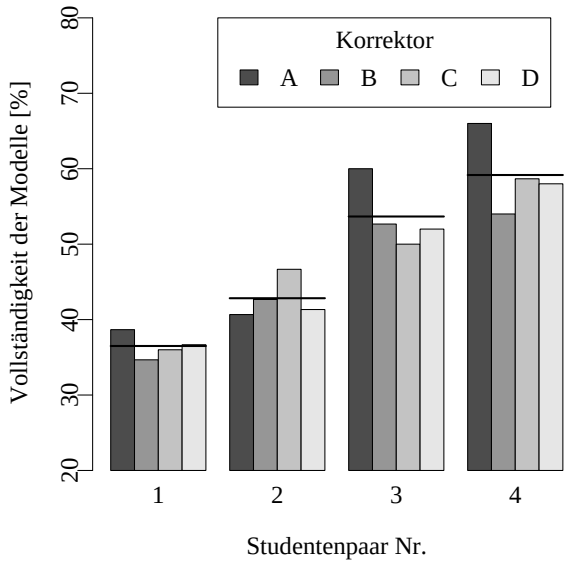

Abbildung 5.1.: Erreichte Punktzahlen der betrachteten Modelle.

$(r$; $p)$	A	B	C
B	(0,904 ; 0,048)	–	–
C	(0,826 ; 0,087)	(0,935 ; 0,032)	–
D	(0,964 ; 0,018)	(0,968 ; 0,016)	(0,945 ; 0,003)

Tabelle 5.1.: Die paarweise Korrelation der Korrekturergebnisse.

Um die Ergebnisse der Korrektoren zueinander in Beziehung zu setzen, wurde für alle möglichen Korrektorenpaare der Korrelationskoeffizient r nach Pearson berechnet. Eine positive Korrelation zwischen zwei Variablen A und B bedeutet, dass sich B erhöht, wann immer sich A erhöht und dass B kleiner wird, wann immer A kleiner wird. Eine negative Korrelation zeigt den umgekehrten Fall: Sind zwei Variablen C und D negativ korreliert, sinkt C wann immer D steigt und umgekehrt. Pearsons Korrelationskoeffizient zeigt die Stärke der Korrelation der gegebenen Datenpunkte an: Ein Wert zwischen 0 und 1 zeigt eine positive Korrelation, ein Wert zwischen 0 und -1 zeigt eine negative Korrelation. Ein Wert von 0 zeigt an, dass die Variablen unkorreliert sind.

Da eine positive Korrelation erwartet wird, wurde versucht, die Nullhypothese $H_0 : r \leq 0$ zu verwerfen. Die Nullhypothese kann verworfen werden, wenn die zu den Korrelationskoeffizienten berechneten p-Werte kleiner sind als das Signifikanzniveau $\alpha = 0,05$. Ein p-Wert kleiner als α sagt aus, dass die Wahrscheinlichkeit, die berechneten (oder höhere) Korrelationskoeffizienten zufällig zu erhalten kleiner ist als 5 %. Die p-Werte wurden mit einem einseitigen T-Test berechnet.

	A	B	C	D
r	0,961	0,976	0,945	0,999
p	0,019	0,012	0,027	0,000

Tabelle 5.2.: Korrelation der Korrektoren mit dem Gruppendurchschnitt.

Tabelle 5.1 fasst die Ergebnisse dieser Berechnungen zusammen. Wie dort zu sehen ist, konnte mit dem vorgestellten Ansatz eine hohe Korrelation der Korrekturergebnisse erreicht werden. Bis auf den Korrelationskoeffizienten zwischen Korrektor A und C sind alle Ergebnisse statistisch signifikant. Obwohl damit die Nullhypothese nicht für alle Korrektoren verworfen werden kann, zeigt der p-Wert dieser Korrelation, dass die Wahrscheinlichkeit für eine zufällig entstandene Korrelation kleiner ist als 9 %.

Um die Korrelation der Korrektoren mit dem durchschnittlich erreichten Ergebnis zu vergleichen, wurde ebenfalls ein Korrelationskoeffizient berechnet. Tabelle 5.2 zeigt die Berechnungsergebnisse. Auch hier zeigt sich ein deutlicher Zusammenhang der Prüfgrößen: Alle Korrelationen sind stark ausgeprägt und statistisch signifikant.

Obwohl die Korrelationskoeffizienten ein gutes Ergebnis zeigen, muss nochmals auf Abbildung 5.1 eingegangen werden. Die Säulendiagramme für Modell 3 und 4 zeigen, dass Korrektor A das dritte Modell besser bewertet hat, als die übrigen Korrektoren das vierte, obwohl die gesamte Gruppe der Ansicht ist, Modell vier sei das vollständigere der beiden. Für die Studenten, die das dritte Modell erstellten, wäre es am besten gewesen, von Korrektor A bewertet zu werden. Dieses Ergebnis zeigt, dass trotz der Prüfliste immer noch ein Spielraum für die Korrektoren bleibt. Beispiele für einen derartigen Interpretationsspielraum werden in Abschnitt 5.3.5 diskutiert.

5.3. Lessons Learned

Im Laufe der Arbeit mit den Korrekturlisten und den Modellen wurden einige Schwierigkeiten entdeckt, die an dieser Stelle aufgeführt werden sollen. Neben diesen Schwierigkeiten sind an dieser Stelle andere Überlegungen bezüglich vorlesunsbegleitenden oder prüfungsrelevanten Modellierungsaufgaben gesammelt, die für zukünftige Entwicklungen basierend auf dieser Arbeit von Bedeutung sind.

5.3.1. Verwendete Spezifikation

Das Schachspiel und seine Regeln sind weithin bekannt. Im Rahmen der Auswertung konnte festgestellt werden, dass einige Studenten ihr Domänenwissen modellierten, anstatt sich exakt an die vorgegebene Spezifikation zu halten. So enthielten einige Modelle Elemente, die nicht in der Spezifikation enthalten waren, andere Teile (insbesondere die Regeln bezüglich der Rochade) wurden oft nicht modelliert. Diese Ergebnisse legen nahe,

dass einige Studenten die Spezifikation nur überflogen und sie dann beiseite gelegt haben. Infolgedessen wurden manche Modelle mit schlechten Bewertungen versehen, was jedoch keine Schwäche der Korrekturliste darstellt und die Ergebnisse der Auswertung nicht beeinflusste. Verwendet man anstelle einer leicht verständlichen und „bekannten" Spezifikation eine technisch ausgefeilte aber gänzlich unbekannte, kann es zu Missverständnissen und Fehlinterpretationen kommen. Um Modellierungsfehler auszuschließen, die auf diesen Missverständnissen beruhen, müsste man die Studenten im Vorfeld eingehend in der Domäne schulen oder eine sehr ausführliche, alles umfassende Spezifikation verwenden; beide Alternativen sind im Rahmen einer Übung oder einer Klausuraufgabe jedoch nicht durchführbar.

5.3.2. Geforderte Diagrammtypen, Erwartungshorizont

Im Rahmen einer Vorlesung oder Übung (gegebenenfalls einer Klausur) sollte vorher festgelegt werden, welche UML-Diagrammtypen im Erwartungshorizont liegen. Auf diese Weise kann sowohl der Spezifikationstext als auch die Prüfliste gekürzt werden. Eine automatisch erzeugte Korrekturliste kann für die Korrektur von Klausuren speziell angepasst, beispielsweise gewichtet werden um dem Erwartungshorizont gerecht zu werden.

5.3.3. Abgabe von Quelltext anstatt Modellierungen

Obwohl die Abgabe von Quelltexten[1] nicht erwünscht war, wurden – zumindest in manchen Modellierungen – viele Elemente mit (Pseudo-) Code ausgedrückt. Das Verwenden von Quelltext kann in vielen Fällen Modellierungsentscheidungen vorgreifen, da so Programmiersprachen oder gar konkrete Implementierungen festgelegt werden. In den meisten Fällen hätte der Sachverhalt auch mit einem Diagramm ausgedrückt werden können. Da die Prüfliste ausschließlich auf UML- oder OCL-Elemente ausgelegt ist, versagt hier das Korrekturschema. In den genannten Modellierungen wurden die Quelltextangaben als Kommentare zu Methoden usw. gewertet.

Um die Modelle untereinander besser vergleichen zu können und den oben angesprochenen Problemen vorzubeugen, sollte die Abgabe von Quelltext bei derartigen Modellierungsaufgaben ausgeschlossen oder stark eingeschränkt werden.

5.3.4. Verwendung von digitalen Modellen

Der größte Aufwand beim Ausfüllen der Prüfliste ist das Suchen und Finden der Elemente in den mehrseitigen Modellen. Die Abgabe der Modelle in digitaler Form könnte den Korrekturprozess beschleunigen, da man dann im Dokument nach Texten suchen könnte.

[1] Weder Pseudo-Code noch Java- oder C#-Code wurden gefordert.

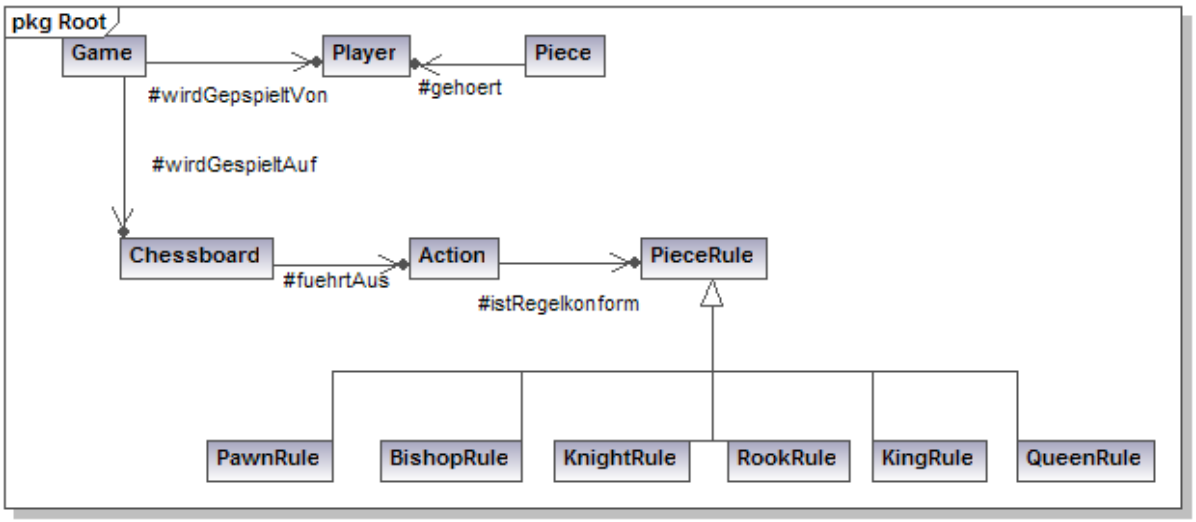

Abbildung 5.2.: Modell ohne spezielle Figurenklassen jedoch mit Regelcontroller.

Wenn erreicht werden kann, dass alle Studenten das gleiche Programm verwenden, könnte das einheitliche Erscheinungsbild zusätzlich dazu beitragen, dass die Korrektur angenehmer und schneller durchgeführt werden kann. Neben diesen Vorteilen können manche Modellierungswerkzeuge die UML- und OCL-Syntax prüfen und so zu syntaktisch besseren Modellen führen.

5.3.5. Einarbeitung der Korrektoren

Um zu gewährleisten, dass die Korrektoren die Korrekturliste konsistent anwenden, sollten sie darauf hingewiesen werden, möglichst wenig Interpretationsarbeit zu leisten.

Beispiel: In einem Modell wurden keine Klassen oder Rollen für die verschiedenen Figuren angelegt, sondern ein Regelsystem erstellt, welches für jede Figur eine Implementierung anbot. Die Modellierung ist in Abbildung 5.2 wiedergegeben. In diesem Fall sind die Fragen nach den Klassen, Rollen oder Instanzen zu verneinen.

Eine zu enge Auslegung könnte jedoch ebenfalls zu Unstimmigkeiten führen. Betrachtet man beispielsweise die Multiplizitäten für den Springer, so entsteht mit dem vorgestellten Ansatz die Frage, ob es *zwei* Springer gibt. Die Multiplizitäten können jedoch auch 0..2 oder 0..10 sein; ersteres entsteht durch das Schlagen der Springer, letzteres durch den Tausch aller Bauern gegen Springer. In diesem Beispiel ist das Modellierungsergebnis abhängig davon, ob der Modellierer die Auswirkungen der möglichen Züge interpretiert oder ob er am Wortlaut der Beschreibung stehen bleibt.

Bei der Einarbeitung der Korrektoren sollte nachdrücklich darauf hingewiesen werden, dass die Prüfliste vollständig auszufüllen ist. Es ist beispielsweise nicht ausreichend, nach einer gefundenen Klasse diesen Abschnitt der Prüfliste zu überspringen, sondern man sollte auch die anderen möglichen Modellierungen suchen.

6. Zusammenfassung

In dieser Arbeit wurde vorgestellt, wie eine zuverlässige Prüfliste für UML-Modelle bezüglich ihrer Vollständigkeit erstellt werden kann. Ausgehend von einer annotierten Spezifikation kann die Prüfliste automatisch erstellt werden. Die Ausgaben der entwickelten Graphersetzungsregeln ist im XML-Format. Dies erlaubt eine flexible Anpassung an das gewünschte Ausgabeformat; zur Umwandlung ein HTML-Dokument wurde ein XSL-T Stylesheet erstellt. Die Geradlinigkeit des Ansatzes führt zu einer Prüfliste, die keine Ausnahmen oder Sonderfälle vorsieht. Ist bekannt, welche Modellierungen (nicht) vorgenommen werden sollen, kann die Prüfliste nach der Erstellung manuell angepasst, gekürzt oder verfeinert werden.

Mithilfe der Prüfliste kann der Recall eines Modells bestimmt werden. Dieser ergibt sich aus dem Quotienten der Anzahl der gefundenen Elemente durch die Gesamtzahl der Elemente der Spezifikation. Da nicht geprüft wird, welche Elemente das Modell zusätzlich enthält, kann keine Aussage über die Precision getroffen werden.

Ebenfalls vernachlässigt werden die Modellkomplexität und die Modellierungweise: Zwei gänzlich unterschiedliche Modelle, von denen eins sehr ausgefeilt [1] und eines einfach gehalten ist, können denselben Recall haben. Entwurfsfehler, sogenannte *Bad Smells*, werden ebenfalls nicht berücksichtigt.

Die Eignung der Prüfliste als Korrekturvorlage für Übungsaufgaben wurde empirisch untersucht. Das Ergebnis der Korrelationsanalyse legt nahe, dass die Prüfliste als Korrekturvorlage geeignet ist, auch wenn nicht alle Ergebnisse statistisch signifikant sind. Trotz der klaren Vorgaben der Prüfliste verbleibt ein Interpretationsspielraum für den Korrektor.

[1] Zum Beispiel durch die Verwendung von Entwurfsmustern.

A. Tabellen

Tabelle A.1.: Die thematischen Rollen von SALE.

ROLLE	KURZFORM	BEISPIEL	SIEHE AUCH
actus	ACT	Die Handlung, die von einer Sache oder einer Person ausgeführt wird.	*agens*, *patiens*
agens	AG	Der Handelnde, der Aktive. Person oder Sache, die eine Handlung durchführt.	*actus*, *patiens*
*beneficiens**	BEN	Der von einer Handlung Profitierende (+). Person oder Sache, zu deren Vorteil (+) oder Nachteil (−) eine Handlung ausgeführt wird.	*fautor*, *favor*
*causa**	CAU	Sachverhalt, der die Ursache einer Handlung darstellt (+) oder trotz dessen (−) eine Handlung ausgeführt wird.	*actus*, *intentio*
comes	COM	Der Begleiter eines Elements.	*dux*
comparius	COMP	Das Element, mit dem verglichen wird.	*compariens*
compariens	COMPII	Das Element, das verglichen wird.	*comparius*
contratius	CONT	Der Gegner eines Elements.	*contrariens*
contrariens	CONTII	Das Element, das einen Gegner hat.	*contratius*
*creator**	CREA	Person oder Sache, die etwas erzeugt (+) oder zerstört (−).	*opus*
*destinatio***	DEST	Ein Ziel oder Endpunkt.	*origo*, *positio*
*dimensio***	DIM	Das Ausmaß von etwas.	*locus*, *tempus*
donor	DON	Person oder Sache, die eine Sache abgibt.	*habitum*, *possessor*, *recipiens*
dux	DUX	Person oder Sache, die von einer anderen Person oder Sache begleitet wird oder zusammen mit dieser auftritt.	*comes*
experior	EXP	Jemand, der etwas erfährt (z.B. durch Sinneswahrnehmung)	*notio*, *stimulus*
*fautor**	FAU	Person oder Sache, die eine Handlung zum Vorteil (+) oder Nachteil (+) einer Person oder Sache ausführt.	*beneficiens*, *favor*
*favor**	FAV	Der Vorteil (+) oder Nachteil (−) einer Person oder Sache.	*fautor*, *beneficiens*
fictum	FIC	Eine Rolle, die jemand oder etwas einnimmt.	*fingens*
fingens	FIN	Person oder Sache, die eine Rolle einnimmt.	*fictum*
frequens	FREQ	Häufigkeit oder Zeitpunkte einer Handlung.	*tempus*
habitum	HAB	Eine Habe, der Besitz. Person oder Sache, die von einer Person oder Sache besessen (auch kurzzeitig), erhalten oder weitergegeben wird.	possessor, donor, recipiens
*instrumentum**	INST	Das Hilfsmittel, mit dem (+) oder ohne das (−) eine Handlung durchgeführt wird.	*actus*, *modus*
intentio	INT	Der Zweck einer Handlung.	*actus*, *causa*
limes	LIM	Der Pfad, den etwas nimmt.	*locus*
*locus***	LOC	Ein Ort, ein Platz, eine Gegend.	*dimensio*, *destinatio*, *frequens*, *origo*, *positio*
modus	MOD	Die Art und Weise, in der eine Handlung durchgeführt wird.	*actus*, *instrumentum*
notio	NOT	Der Eindruck, die Vorstellung, der Begriff, die Idee oder die Erfahrung, die jemandem übermittelt wird. Sachverhalt, den eine Person oder Sache erfährt (z.B. durch Sinneswahrnehmung).	*experior*, *stimulus*
omnium	OMN	Person oder Sache, die ein Ganzes, das aus Teilen besteht, darstellt.	*pars*
*opus**	OPUS	Das durch eine Handlung erzeugte (+) oder zerstörte (−) Element.	*creator*
*origo***	ORIG	Eine Quelle, eine Herkunft, ein Beginn.	*destinatio*, *positio*
pars	PAR	Ein Teil eines Ganzen.	*omnium*

Tabelle A.1.: Die thematischen Rollen von SALE (Fortsetzung).

ROLLE	KURZFORM	BEISPIEL	SIEHE AUCH
patiens	PAT	Person oder Sache, die von einer Handlung betroffen ist oder an der eine Handlung ausgeführt wird.	*actus*, *agens*
*positio***	POS	Die aktuelle Position (im Sinne eines festen Bezugspunktes) eines Elements (nicht nur lokal!)	*destinatio*, *origo*
possessor	POSS	Der (gegenwärtige) Besitzer eines Elements, der „Haber“.	*donor*, *habitum*, *recipiens*
qualitas	QUAL	Die Beschaffenheit eines Elements.	*qualificiens*
qualificiens	QUALII	Das durch eine Beschaffenheit beschriebene Objekt.	*qualitas*
recipiens	RECP	Der Empfänger eines Elements.	*donor*, *habitum*, *possessor*
stimulus	STIM	Person oder Sache, die etwas (einen Eintruck/Bild/Idee/...) erzeugt, das eine Person von der Sache erfährt (z.B. durch Sinneswahrnehmung).	*experior*, *notio*
substitutus	SUBS	Person oder Sache, die durch eine andere Person oder Sache ersetzt oder vertreten wird.	*substituens*
substituens	SUBSII	Person oeder Sache, die eine andere Person oder Sache ersetzt oder vertritt.	*substitutus*
sumtio	SUM	Die Vorbedingung einer Handlung, Beziehung oder Zustand.	*actus*
	SUMII	?!? Braucht man das?	*sumtio*
*tempus***	TEMP	Eine Zeitangabe.	*dimensio*, *destinatio*, *frequens*, *origo*, *positio*
thema	THE	Der Inhalt, der Gegenstand einer Betrachtung.	*thematiens*
thematiens	THEII	Das Element, dessen Inhalt oder Gegenstand beschrieben wird.	*thema*
Modalkonstr.	MODAL_MOD	Modalverb	
Modalkonstr.	MODAL_WHAT	Was	MODAL_MOD, MODAL_WHO
Modalkonstr.	MODAL_WHO	Wer	MODAL_MOD, MODAL_WHAT
predecessor	PRE	Vorgänger. Handlung, die vor einer anderen ausgeführt wird.	successor
successor	SUCC	Nachfolger. Handlung, die nach einer anderen ausgeführt wird.	predecessor
Equal Drop	EQD	Gleiches Element (wird aus dem Graph entfernt); unechte Referenz	immer mit EQK
Equals Keep	EQK	Gleiches Element (verbleibt im Graph) unechte Referenz	immer mit EQD
Equals SetElem	EQB	Mengenelement (immer mit EQS)	EQS
Equals Set	EQS	Mengenelement (immer mit einem oder mehreren EQBs)	EQB

* kennzeichnet Rollen, die vorzeichenbehaftet sind; Kennzeichnung durch P bzw. M (INSTP oder CREAM)
** kennzeichnet Rollen, die nicht alleine auftreten können (TEMP_POS oder LOC_DIM)

Tabelle A.2.: Abbildung von SALE-Strukturen auf UML-Elemente.

Thematische Rolle(n)	Mögliche UML-Artefakte
AG, BEN, COM, COMP, COMPII, CONT, CONTII, CREA, DON, DUX, EXP, FAU, FAV, FIC, FIN, HAB, OMN, OPUS, PAR, PAT, POSS, RECP, SUB, SUBII	Klasse Rolle Instanz
ACT	Methode Zustand (-sübergang) Beziehung
CAU	Nicht darstellbar?
LOC_*	Instanz einer (unbekannten) Klasse
TEMP_*	Instanz einer (unbekannten) Klasse
QUAL, QUALII	Klassen für [QUAL] und [QUALII] Beziehung zwischen Klassen Enum für [QUAL] Attribut vom [QUAL]-Enum bei [QUALII]-Klasse (welche Qualität?) Bool'sche Funktion bei [QUALII]-Klasse (hat es *diese* Qualität?) Skalare Funktion bei [QUALII]-Klasse (in welchem Maß hat es die Qualität?) Funktionen bei [QUALII]-Klasse, die [QUAL]-Enum setzen und/oder zurückgeben
EXP, NOT, STIM	Methode mit entspr. Parametern (Kommunikation) Zustandsdiagramm mit Signalen Klassen mit Beziehungen
MOD	Zustandsübergang mit Wächter
INST	Zustandsübergang mit Wächter
SUM	OCL Vorbedingung oder Invariante Modelliert in einem Automaten (Zustand!) if-Anweisung im Kommentar Insbesondere: In [SUM] verwendete Attribute prüfbar Aktivitätsdiagrammteil Sequenzdiagrammteil
PRE, SUCC	Sequenzdiagrammteil Aktivitätsdiagrammteil Aufeinander folgende Zustände eines Automaten
STAT, STATII	Sequenzdiagrammteil Aktivitätsdiagrammteil Zustand Methode (implizit, evtl. bei anderer Klasse)
THE, THEII	Beziehung zwischen den Klassen von [THE] und [THEII]
MODAL_*	OCL Bedingung Zustandsübergang (ggf. mit Wächter, auch mit Instrument) if-Anweisung im Kommentar Konstruktion und/oder Multiplizitäten
* [Multiplizität]	Multiplizität OCL Bedingung

Tabelle A.2.: Abbildung von SalE-Konstrukten auf UML-Elemente (Fortsetzung).

Thematische Rolle(n)	Mögliche UML-Artefakte
SUB, SUBII	Zwei Unterklassen einer (unbekannten) Oberklasse
OMN, PARS	Bei Klassen/Instanzen: Aggregation Bei Klassen: Vererbungsbeziehung mit [OMN] als Oberklasse Bei ACT: Sequenzdiagramm: Unteraufrufe Bei ACT: Kommentar an Methode: Unteraufrufe
POSS, HAB	Beziehung zwischen [POSS] und [HABB]
$ [Attribute]	Bei Klassen/Instanzen: Bool'sche Funktion mit Parameter (weißer König) Bei Klassen/Instanzen: Zustand (grüne Ampel, rote Ampel) Bei Klassen/Instanzen: Als Attribut modelliert Bei Methoden: Parameter Bei Methoden: Im Namen enthalten Bei Beziehungen: Attribut Bei Beziehungen: Eigene Beziehung
FIN, FIC	Vererbungsbeziehung (idR falsch) Beziehung Rolle
CREA, OPUS	Methode mit Rückgabe Entwurfsmuster Zustandsübergang
FAV, FAU, BEN	Methode (vorzugsweise bei [AG], sonst bei [PAT]) mit entspr. Parametern (Kommunikation) Klassen mit Beziehungen
DON, RECP, HAB	Methode (vorzugsweise bei [AG]) mit [HAB]-Parameter Klassen mit Beziehungen
COMP, COMPII	(Bool'sche) Methode zum Vergleichen von [COMP] und [COMPII]
CONT, CONTII	Zwei Instanzen einer Klasse Zwei Unterklassen einer (unbekannten) Oberklasse
DUX, COM	Mit ACT: Methode (void [DUX].[ACT]([COM])
INST und MOD	Klasse Parameter in Methode Attribut einer Assoziation Enum INST eher Klasse, MOD eher Enum
FREQ	Zustandsautomat Kommentar in einer Methode (Zähler...)
PRE, SUCC	Sequenzdiagrammteil Aktivitätsdiagrammteil Kommentar in einer Methode Automat
LIM	Nicht darstellbar?

Tabelle A.2.: Abbildung von SALE-Konstrukten auf UML-Elemente (Fortsetzung).

Thematische Rolle(n)	Mögliche UML-Artefakte
SETs und DUX, COM	Wenn Elemente keine Instanzen sind: Vererbungsbeziehung zu (idR unbekannter) Oberklasse Wenn Set|AG: Methode bei allen Elementen anlegen (Klassendiagramm) und aufrufen (Sequenzdiagramm) Wenn Set|AG: Bei Methode (=ACT) min.1 Parameter(Dux/Com); Signatur: void dux.ACT(com) Wenn Set|PAT: wie AG Wenn Set|[bel. Rolle]: Dann Regel von [bel. Rolle] für alle Elemente des Sets Wenn Set|SUM: Wenn Set|ACT: Unteraufrufe bei übergeordneter Methode?
SUM, MODAL_*	OCL Bedingung if-Anweisung im Kommentar Konstruktion und Multiplizitäten Modelliert in einem Automaten (Zustand!) (ggf. Übergang mit Guard/Instrument) Aktivitätsdiagrammteil Sequenzdiagrammteil
SUM mit OMN, PARS	Bedingung für alle [PARS] geprüft?

B. Die Experimentaufgabe

Aufgabe 1: Schach (30P)

Modellieren Sie den angegebenen Ausschnitt der FIDE-Schach-Spezifikation in UML. Es handelt sich um die ersten fünf Artikel der internationalen Turnierschach-Regeln. Zweck des Modells sei die Entwicklung eines Schach-Programms, das gegen andere Schach-Programme oder menschliche Spieler antreten können soll.

Der Text entspricht weitgehend dem Original, es wurden lediglich einzelne Sätze in einfacheres Englisch umformuliert. Um den Umfang der Aufgabe zu beschränken, haben wir die Artikel 6-14 der Spezifikation ausgenommen. Modellieren Sie das, was in der Spezifikation steht, und nicht das, was Sie über Schach wissen (oder zu wissen glauben): Es geht bei dieser Aufgabe um eine möglichst spezifikationstreue Umsetzung in UML. Denken Sie also auch an Multiplizitäten, Rollennamen, Attribute, Methoden, Zustände, Übergänge, Wächterausdrücke und so weiter. Beachten Sie dabei den Zweck des Modells: Es kann sein, dass bestimmte Regelungen nicht modelliert werden müssen. **Dokumentieren Sie Ihre Entscheidung, Dinge nicht zu modellieren, sorgfältig und begründen Sie Ihre Entscheidung ausreichend – ohne Begründung bekommen Sie keine Punkte für Weggelassenes!**

Verwenden Sie für Ihre Modellierung Klassen-, Aktivitäts-, Sequenz- und Zustandsübergangsdiagramme und OCL-Ausdrücke. Graphische Notationen sind in der Regel wenig platzsparend, rechnen Sie daher mit etwa **8–10 Seiten** für Ihre Lösung. Sie können nicht davon ausgehen, dass Sie für 10 Klassen mit jeweils 3 Methoden die volle Punktzahl bekommen.

Bitte beschreiben Sie Ihre Blätter nur einseitig!

Article 1: The nature and objectives of the game of chess

1.1.

The game of chess is played between two opponents. They alternately move their pieces on a square board called a chessboard. The player with the white pieces commences the game. A player has the move, after his opponent made his move.

1.2.

The objective of each player is to place the opponent's king under attack in such a way that the opponent has no legal move. The player who achieves this goal has checkmated the opponent's king and wins the game. Leaving one's own king under attack, exposing one's own king to attack and also capturing the opponent's king are not allowed. The opponent whose king has been checkmated has lost the game.

1.3.

If the position is such that neither player can possibly checkmate, the game is drawn.

Article 2: The initial position of the pieces on the chessboard

2.1.

The chessboard is composed of an 8x8 grid which is composed of 64 equal squares which are alter-nately light and dark. The chessboard is placed between the players in such a way that the near corner square to the right of the player is white.

2.2.

At the beginning of the game one player has 16 white pieces; the other has 16 black pieces. These pieces are as follows: A white king, a white queen, two white rooks, two white bishops, two white knights, eight white pawns, a black king, a black queen, two black rooks, two black bishops, two black knights and eight black pawns.

2.3.

The initial position of the pieces on the chessboard is as follows: - GESTRICHEN -

2.4.

The eight vertical columns of squares are called files. The eight horizontal rows of squares are called ranks. A straight line of squares of the same color, touching corner to corner, is called a diagonal.

Article 3: The moves of the pieces

3.1

A piece may not move to a square which is occupied by a piece of the same color. If a piece moves to a square occupied by an opponent's piece, the opponent's piece is captured and removed from the chessboard as part of the same move. A piece attacks an opponent's piece if the piece could make a capture on that square. A piece attacks a square, even if this piece cannot move to that square because it would then leave or place the king of its own color under attack.

3.2.

The bishop may move to any square along a diagonal on which it stands.

3.3.

The rook may move to any square along the file or the rank on which it stands.

3.4.

The queen may move to any square along the file, the rank or a diagonal on which it stands.

3.5.

When making these moves the bishop, rook or queen may not move over any intervening pieces.

3.6.

The knight may move to one of the squares nearest to that on which it stands but not on the same rank, file or diagonal.

3.7.

a) The pawn may move forward to the unoccupied square immediately in front of it on the same file, or

b) on its first move the pawn may advance one or two squares along the same file provided both squares are unoccupied, or

c) the pawn may move to a square occupied by an opponent's piece, which is diagonally in front of it on an adjacent file, capturing that piece.

d) A pawn attacking a square crossed by an opponent's pawn which has advanced two squares in one move from its original square may capture this opponent's pawn in the following move as though the latter had been moved only one square. This capture is called an „en passant" capture.

When a pawn reaches the rank furthest from its starting position it must be exchanged as part of the same move for a new queen, rook, bishop or knight of the same color. The player's choice is not restricted to pieces that have been captured previously. This exchange of a pawn for another piece is called promotion and the effect of the new piece is immediate.

3.8.

The king may move to any adjoining square if this square is not attacked by one or more pieces of the opponent. Alternately, the king may castle: This is a move of the king and either rook of the same color on the same rank, counting as a single move of the king and executed as follows: The king is transferred from its original square two squares towards the rook, then that rook is transferred to the square the king has just crossed.

3.8.1

If the king has moved, he may not castle. The king may not castle with a rook that has moved.

3.8.2

If the square on which the king stands, or the square which it must cross, or the square which it is to occupy, is attacked by one or more of the opponent's pieces, the king may not castle. If there is any piece between the king and the rook with which castling is to be effected, the king may not castle.

3.9.

The king is said to be „in check“ if it is attacked by one or more of the opponent's pieces, even if such pieces are constrained from moving to that square because they would then leave or place their own king in check. No piece can be moved that will expose the king of the same color to check. No piece can be moved that will leave the king of the same color in check.

Article 4: The act of moving the pieces

4.1.

Each move must be made with one hand only.

4.2.

Provided that he first expresses his intention, the player having the move may adjust one or more pieces on their squares.

4.3.

Provided that he not expresses his intention to adjust, if the player having the move deliberately touches on the chessboard one or more pieces of his own, he must move the first touched piece that can be moved. Provided that he not expresses his intention to adjust, if the player having the move deliberately touches on the chessboard one or more pieces of his opponent's, he must capture the first touched piece, which can be captured. Provided that he not expresses his intention to adjust, if the player having the move deliberately touches on the chessboard one piece of each color, he must capture the opponent's piece with his piece. Provided that he not expresses his intention to adjust, if the player having the move deliberately touches on the chessboard one piece of each color, and if he may not capture the opponent's piece with his piece, he must move the first touched piece which can be moved or capture the first touched piece that can be captured. If it is unclear, whether the player's own piece or his opponent's was touched first, the player's own piece shall be considered to have been touched before his opponent's.

4.4.

a) If a player deliberately touches his king and his rook he must castle on that side if it is legal to do so.

b) If a player deliberately touches a rook and then deliberately touches his king he may not castle on that side on that move. If a player deliberately touches a rook and then his king, he must move with the first piece touched that can be moved.

c) If a player intends to castle and the player touches the king or the king and a rook at the same time and castling on that side is illegal, the player must make another legal move with his king which may include castling on the other side. If a player intends to castle and the player touches the king or the king and a rook at the same time and castling on that side is illegal and the king has no legal move, the player may make any legal move.

d) If a player promotes a pawn and the piece has touched the square of promotion, the choice of the piece is finalized.

4.5.

If the player cannot move any of the touched pieces of his own and if the player cannot capture any of the touched pieces of his opponent's, the player may make any legal move.

4.6.

When, as a part of a legal move, a piece has been released on a square, it cannot then be moved to another square. a) If a piece is captured and the captured piece has been removed from the chessboard and the player, having placed his own piece on its new square, has released this capturing piece from his hand, the move has been made. b) If the player castles and the player's hand has released the rook on the square previously crossed by the king, the move has been made. If the player castles and the player has re-leased the king from his hand, the move is not yet made, but the player no longer has the right to make any move other than castling on that side, if this is legal. c) If a player promotes a pawn, the pawn has been removed from the chessboard, and the player's hand has released the new piece after placing it on the promotion square, the move has been made. If a player promotes a pawn and the player has released from his hand the pawn that has reached the promotion square, the move is not yet made and the player may not play the pawn to another square.

4.7.

A player forfeits his right to a claim against his opponent's violation of Article 4.3 or 4.4, once he deliberately touches a piece.

Article 5: The completion of the game

5.1.

a) If a player checkmates his opponent's king, he wins the game. If a player checkmates his opponent's king and the move was legal, the game immediately ends.

b) If a player resigns, his opponent wins the game. If a player resigns, the game immediately ends.

5.2.

a) If the player having the move cannot make a move and his king is not in check, the game is drawn. If the player having the move cannot make a move and his king is not in check, he game is said to end in stalemate. If the player having the move cannot make a move and his king is not in check and the move producing the stalemate position was legal, the game im-mediately ends.

b) If neither player can checkmate the opponent's king with any series of legal moves, the game is drawn. If neither player can checkmate the opponent's king with any series of legal moves, the game is in a dead position. If neither player can checkmate the opponent's king with any series of legal moves and the move producing the position was legal, the game immediately ends.

c) If the players agree to drawing the game, the game is drawn. If the game is drawn, the game immediately ends.

d) If any identical position is about to appear on the chessboard at least three times or any identical position has appeared on the chessboard at least three times, the game may be drawn.

e) The game may be drawn if each player has made at least the last 50 consecutive moves without the movement of any pawn and without any capture.

C. Die annotierte Fassung der FIDE-Regeln

Listing C.1: Die annotierten FIDE Laws of Chess.

```
/* Article 1: The nature and objectives of the game of chess */
/* 1.1. The game of chess is played between two opponents who move their pieces
    alternately on a square board called a 'chessboard'. */
/* The game of chess is played between two opponents. */
[ #The game_of_chess|PAT #is played|ACT #between *two opponents|AG ] .

/* They alternatey move their pieces on a square board called a 'chessboard'.*/
[ They|AG $alternately move|ACT their|POSS ^pieces|{HAB,PAT}
  #on [ #a $square ^board|{PAT,FIN} called|ACT #a chessboard|FIC ]|LOC_POS ] .

[ @They|EQD @opponents|EQK ] .
[ @their|EQD @They|EQK ] .

/* The player with the white pieces commences the game. */
[ [ #The ^player|POSS #{with the} $white pieces|HAB ]|AG commences|ACT #the game|PAT ] .
[ @game|EQD @game_of_chess|EQK ] .
[ @player|EQB @They|EQS ] .

/* A player is said to 'have the move', when his opponent's move has been 'made'. */
/* [ A player has the move, after his opponent made his move. ] */
/* Fixed: Zeitlicher Ablauf mit PRE und SUCC kodieren */
[ [ #A player|POSS #has #the move|HAB ]|SUCC #after
  [ his|CONTII opponent|{POSS,CONT} made|ACT #his move|{HAB,PAT} ]|PRE ] .
[ @his|EQD @player|EQK ] .

/* 1.2. The objective of each player is to place the opponent's king 'under attack'
   in such a way that the opponent has no legal move. */
[ [ #The ^objective|HAB #of #each player|POSS ]|THEII #is
  [ #to place|ACT #the opponent's|POSS king|{HAB, STATII, PAT}
    under_attack|STAT ]|{THE,ACT}
  [ #in #such #a #way #that #the opponent|POSS #has *no $legal move|HAB ]|INST ] .

[ @player|EQB They|EQS ] .
[ @opponent's|EQD @opponent|EQK ] .
[ @player|CONTII @opponent|CONT ] .

/* The player who achieves this goal is said to have 'checkmated' the opponent's king
   and to have won the game.*/
/* The player who achieves this goal has checkmated the opponen's king
   and wins the game. */
[ #The player|AG [ who|AG achieves|ACT #this goal|PAT ]|SUM
  { [ #has ^checkmated|ACT #the opponent's|POSS king|{HAB,PAT} ] AND
    [ ^wins|ACT #the @game|PAT ]
  }|ACT
] .
[ @game|EQD @game_of_chess|EQK ] .
[ @who|EQD player|EQK ] .
```

```
[ @player|EQB @They|EQS ] .
[ @opponent's|EQB @They|EQS ] .
[ @player|CONTII @opponent's|CONT ] .
[ goal|EQ objective|EQ ] .

/* Leaving one's own king under attack, exposing one's own king to attack and also
   'capturing' the opponent's king are not allowed. */
[ { [ Leaving|ACT one's|POSS #own king|{HAB,STATII} under_attack|STAT ],
    [ exposing|ACT @one's|POSS #own king|{HAB,STATII} to_attack|STAT ] AND #also
    [ capturing|ACT #the opponent's|POSS king|HAB ]
  }|MODAL_WHAT #are not_allowed|MODAL_MOD ] .
[ @one's1|EQB @They|EQS ] .
[ @one's2 @They|EQS ] .

/* The opponent whose king has been checkmated has lost the game. */
[ #The @opponent|AG [ whose|POSS king|{HAB, STATII} #has #been checkmated|STAT ]|SUM
  #has lost|ACT #the @game|PAT ] .

[ @whose|EQD @opponent|EQK ] .

/* 1.3. If the position is such that neither player can possibly checkmate, the game
   is drawn. */
[ [ #If #the #position #is #such #that
    *neither @player|{AG,MODAL_WHO} can|MODAL_MOD #possibly checkmate|{MODAL_WHAT,ACT}
  ]|SUM #the @game|STATII #is drawn|STAT
] .

/* Article 2: The initial position of the pieces on the chessboard */
/* 2.1. The chessboard is composed of an 8x8 grid of 64 equal squares alternately light
 * (the 'white' squares) and dark (the 'black' squares). */
/* The chessboard is composed of an 8x8 grid of 64 equal squares which are alternately light and dark. */
[ #The chessboard|OMN #is #composed #of #an $eight_by_eight
    [ ^grid|OMN #of *64 $equal
    [ ^squares|QUALII #which #are $alternately { light AND dark }|QUAL ]|PAR
    ]|PAR
] .

/* The chessboard is placed between the players in such a way that the near corner square
 * to the right of the player is white. */

/* 2.2. At the beginning of the game one player has 16 light-coloured pieces (the 'white' pieces);
 * the other has 16 dark-coloured pieces (the 'black' pieces): */
[ #At #the [ ^beginning|HAB #of #the game|POSS ]|TEMP_ORIG
  #one playerA|POSS #has *16 $light_coloured piecesW|HAB #{(the 'white' pieces);}
  the_other|POSS #has *16 $dark_coloured piecesB|HAB #{(the 'black' pieces) }
] .
[ @playerA|EQB @They|EQS] .
[ @the_other|EQB @They|EQS ] .
[ @playerA|CONTII @the_other|CONT ] .

/* These pieces are as follows:
// Symboldefinitionen... Annoation sinnlos!
A white king, usually indicated by the symbol
A white queen, usually indicated by the symbol
Two white rooks, usually indicated by the symbol
Two white bishops, usually indicated by the symbol
Two white knights, usually indicated by the symbol
Eight white pawns, usually indicated by the symbol
A black king, usually indicated by the symbol
A black queen, usually indicated by the symbol
Two black rooks, usually indicated by the symbol
Two black bishops, usually indicated by the symbol
Two black knights, usually indicated by the symbol
```

```
111 Eight black pawns, usually indicated by the symbol */
112 [ #These pieces|OMN #{are as follows}
113   { *one $white king,
114     *one $white queen,
115     *Two $white rooks,
116     *Two $white bishops,
117     *Two $white knights,
118     *Eight $white pawns,
119     *one $black king,
120     *one $black queen,
121     *Two $black rooks,
122     *Two $black bishops,
123     *Two $black knights AND
124     *Eight $black pawns }|PAR
125 ] .
126
127 /* 2.3. The initial position of the pieces on the chessboard is as follows:*/
128 // Sinnlos - Bild muss beschrieben werden! --> Machen wir für die Aufgabe nicht!
129
130 /* 2.4. The eight vertical columns of squares are called 'files'. */
131 [ #The *eight $vertical [ ^columns|OMN #of @squares|PAR ]|FIN #are #called files|FIC ] .
132
133 /* The eight horizontal rows of squares are called ranks'. */
134 [ #The *eight $horizontal [ ^rows|OMN #of @squares|PAR ]|FIN #are #called ranks|FIC ] .
135
136 /* A straight line of squares of the same colour, touching corner to corner, is called a 'diagonal'.*/
137 [ #A $straight
138   [ ^line|OMN
139     [ #of ^@squares|QUALII #of #the $same colour|QUAL touching|ACT corner_to_corner|LOC_DIM ]|{QUAL,PAR}
140   ]|FIN #is #called #a diagonal|FIC
141 ] .
142
143 /* Article 3: The moves of the pieces */
144 /* 3.1 It is not permitted to move a piece to a square occupied by a piece of the same colour. */
145 /* A piece may not move to a square which is occupied by a piece of the same colour. */
146 [ #A piece|MODAL_WHO may_not|MODAL_MOD
147   [ move|ACT #to #a
148     [ ^square|PAT #which #is occupied|ACT #by #a piece|{AG,QUALII}
149       #of #the $same colour|QUAL ]|{LOC_DEST,STATII}
150   ]|MODAL_WHAT
151 ] .
152
153 /* If a piece moves to a square occupied by an opponent's piece the latter is captured and removed
154  * from the chessboard as part of the same move. */
155 [ [ #If #a piece|AG moves|ACT #to
156     [ #a ^square|PAT occupied|ACT #by #an opponent's|POSS piece|{HAB,AG} ]|LOC_DEST
157   ]|SUM #the latter|PAT #is {captured AND removed}|{ACT,PAR} #from #the chessboard|LOC_ORIG
158   #{as part of the same} move|OMN
159 ] .
160 [ @which|EQD @square|EQK ] .
161 [ @latter|EQD @piece ] .
162
163 /* A piece is said to attack an opponent's piece if the piece could make a capture on that square
164  * according to Articles 3.2 to 3.8. */
165 /* A piece_A attacks an opponent's piece_b if the piece could make a capture on that square. */
166 [ #A pieceA|AG attacks|ACT #an
167   [ opponent's|POSS piece|HAB ]|PAT [ #if #the @pieceA|{AG,MODAL_WHO} could|MODAL_MOD
168     [ make_a_capture|ACT on_that_square|LOC_POS]|MODAL_WHAT ]|SUM
169   #{ according to Articles 3_2 to 3_8}
170 ] .
171
172 /* A piece is considered to attack a square, even if such a piece is constrained from moving to that square
173 * because it would then leave or place the king of its own colour under attack. */
```

```
174 /* A piece attacks a square even if this piece cannot move to that square because it would the leave
175  * the king of it's own colour under attack or place the king under attack. */
176 [ #A piece|AG attacks|ACT #a square|PAT [ #even #if #this @piece|MODAL_WHO cannot|MODAL_MOD
177   [ move|ACT #to #that @square|LOC_DEST ]|MODAL_WHAT #because
178     { [ it|AG #would #then leave|ACT #the king|{PAT,QUALII,STATII}
179         #of #its $own colour|QUAL under_attac|STAT ] OR
180       [ place|ACT #the @king|{PAT,STATII} under_attack|STAT ]
181     }|CAUP
182   ]|CAUM
183 ] .
184 [ @it|EQD @piece|EQK ] .
185
186 /* 3.2. The bishop may move to any square along a diagonal on which it stands. */
187 [ #The bishop|{AG,MODAL_WHO} may|MODAL_MOD
188   [ move|ACT #to *any square|LOC_DEST #along #a
189     [ ^diagonal|PAT #on #which it|AG stands|ACT ]|LOC_POS ]|MODAL_WHAT
190 ] .
191 [ @it|EQK @bishop|EQK ] .
192 [ @which|EQD @diagonal|EQK ] .
193
194 /* 3.3. The rook may move to any square along the file or the rank on which it stands. */
195 [ #The rook|{AG,MODAL_WHO} may|MODAL_MOD [ move|ACT #to $any square|LOC_DEST #along #the
196   { file OR #the rank}|LOC_POS
197   #on #which it|AG stands|ACT]|MODAL_WHAT ] .
198 [ @it|EQD @rook|EQK ] .
199
200 /* 3.4. The queen may move to any square along the file, the rank or a diagonal on which it stands. */
201 [ #The queen|{AG,MODAL_WHO} may|MODAL_MOD  [ move|ACT #to $any square|LOC_DEST #along
202   { #the file, #the rank OR #a diagonal}|LOC_POS #on #which it|AG stands|ACT ]|MODAL_WHAT ] .
203
204 /* 3.5. When making these moves the bishop, rook or queen may not move over any intervening pieces. */
205 [ #When #making #these #moves {#the bishop, rook AND queen}|{AG,MODAL_WHO} may_not|MODAL_MOD
206   [ move_over|ACT $any $intervening pieces|PAT ]|MODAL_WHAT ] .
207
208 /* 3.6. The knight may move to one of the squares nearest to that on which it stands
209  * but not on the same rank, file or diagonal. */
210 [ #The knight|{AG,MODAL_WHO} may|MODAL_MOD [ move|ACT #to *one square|{LOC_DEST,PAR}
211   #{of the [ ^squares nearest to that on which it stands}
212   #{but not on the $same { rank, file, OR diagonal \} ]|OMN}
213 ]|MODAL_WHAT ] .
214
215 /* 3.7. a. The pawn may move forward to the unoccupied square immediately in front of it
216  * on the same file. */
217 [ #The pawn|{AG,MODAL_WHO} may|MODAL_MOD
218   [ move_forward|ACT #to #the $unoccupied square|LOC_DEST $immetiately_in_front_of_it
219   #on #the $same file|LOC_POS ]|MODAL_WHAT ] .
220
221 /* b. On its first move the pawn may move as in (a); alternatively it may advance two squares along the
222  * same file provided both squares are unoccupied. */
223 /* On its first move the pawn may advance one or two squares along the same file provided both squares
224  * are unoccupied. */
225 /* On its first move the pawn may move one or two squares forward along the same file provided both
226  * squares are unoccupied. */
227 [ #On #its $first move|TEMP_POS #the pawn|{AG,MODAL_WHO} may|MODAL_MOD
228   [ move|ACT {*one OR *two} squares|LOC_DIM #forward along_the_same_file|LOC_POS ]|MODAL_WHAT
229     [ #provided *both @squares|STATII #are unoccupied|STAT ]|SUM ] .
230
231 /* c. The pawn may move to a square occupied by an opponent's piece, which is diagonally in front of it on
232  * an adjacent file, capturing that piece. */
233 [ #The pawn|{AG,MODAL_WHO} may|MODAL_MOD
234     {
235         [ move|ACT #to
236             [ #a ^square|PAT occupied|ACT #by #an opponent's|POSS piece|{HAB,AG} #{which is}
```

```
237             $diagonally_in_front_of_it<<4 #on #an $adjacent file|LOC_POS ]|LOC_DEST
238         ] AND
239         [ capturing|ACT #that @piece|PAT ]
240     }|MODAL_WHAT
241 ] .
242 [ @it|EQD @pawn|EQK ] .
243
244
245 /* d. A pawn attacking a square crossed by an opponent's pawn which has advanced two squares in one move
246  * from its original square may capture this opponent's pawn as though the latter had been moved only
247  * one square. */
248 /* A pawn attacking a square crossed by an opponent's pawn which has advanced two squares in one move
249  * from its original square may capture this opponent's pawn in the following move as though the latter
250  * had been moved only one square. */
251 /* A pawn attacking a square crossed by an opponent's pawn which has moved two squares forward in one move
252  * from its original square may capture this opponent's pawn in the following move as though the latter
253  * had been moved only one square. */
254 [ #A
255     [ ^pawn|AG attacking|ACT #a
256         [ ^square|PAT crossed|ACT #by #an opponent's|POSS
257             [ ^pawn_o|{AG,POSS} #which #has moved|ACT *two squares|PAT #forward #in *one move|MOD
258               #{from its} original_square|{LOC_ORIG,HAB}
259             ] |{HAB,AG}
260         ]|PAT
261     ]|{AG,MODAL_WHO} may|MODAL_MOD
262     [ capture|ACT #this opponent's|POSS @pawn_o|{HAB,PAT}
263         [ #as #though #the latter|PAT had_been_moved|ACT *only_one square|LOC_DIM
264         ]|MOD
265     ]|MODAL_WHAT  ] .
266 [ @latter|EQD @pawn_o|EQK ] .
267
268 /* This capture is only legal on the move following this advance and is called an 'en passant' capture. */
269 // Legal im oberen Satz bereits eingefügt.
270 /* This capture is called an 'en passant'capture. */
271 [ #This @capture|FIN #is #called #an en_passant_capture|FIC ] .
272
273 /* When a pawn reaches the rank furthest from its starting position it must be exchanged as part of the
274  * same move for a new queen, rook, bishop or knight of the same colour. */
275 [ [#When #a pawn|{AG,POSS} reaches|ACT #the rank|PAT #furthest #from starting_position|HAB ]|SUM
276   it|{PAT,SUBSII} #must #be exchanged|{ACT,PAR} #as #part #of #the $same move|OMN #for #a $new
277   {queen, rook, bishop OR knight}|{PAT,QUALII,SUBS} #of #the $same colour|QUAL ] .
278 [ @it|EQD @pawn|EQK ] .
279
280 /* The player's choice is not restricted to pieces that have been captured previously. */
281 [ #The [ player's|POSS ^choice|HAB ]|PAT #is not_restricted|ACT #to [ ^pieces|STATII #{that have been}
282   $previously captured|STAT ]|MOD ] .
283
284 /*This exchange of a pawn for another piece is called 'promotion'. */
285 [ #This [ ^exchange|ACT #of #a pawn|{PAT,SUBS} #for $another piece|SUBSII ]|FIN
286   #is #called promotion|FIC ] .
287
288 /* The effect of the new piece is immediate. */
289
290 /* 3.8. a. There are two different ways of moving the king, by:
291 a1. moving to any adjoining square not attacked by one or more of the opponent's pieces.
292  or
293 a2. 'castling'. This is a move of the king and either rook of the same colour on the same rank, counting
294     as a single move of the king and executed as follows: the king is transferred from its original square
295     two squares towards the rook, then that rook is transferred to the square the king has just crossed.
296
297 (1) The right for castling has been lost:
298 a. if the king has already moved, or
299 b. with a rook that has already moved
```

```
300
301 (2) Castling is prevented temporarily
302 c. if the square on which the king stands, or the square which it must cross, or the square which it is to
303    occupy, is attacked by one or more of the opponent's pieces.
304 d. if there is any piece between the king and the rook with which castling is to be effected. */
305
306 /* The king may move to any adjoining square if this square is not attacked by
307   one or more pieces of the opponent's. */
308 [ #The king|{AG,MODAL_WHO} may|MODAL_MOD [ move|ACT #to *any $adjoining square|LOC_DEST
309   [ #if #this @square|PAT #is not_attacked|ACT #by *one_or_more pieces|{HAB,AG}
310   #{of the} opponent|POSS ]|SUM ]|MODAL_WHAT ] .
311
312 /* The right for castling has been lost if the king has already moved
313  * or with a rook that has already moved. */
314 /* If the king has already moved, the right for castling has been lost. */
315 /* If the king has already moved, he may not castle. */
316 /* If the king has moved, he may not castle. */
317 [ [ #If #the king|AG has_moved|ACT]|SUM he|{AG,MODAL_WHO} may_not|MODAL_MOD castle|{ACT,MODAL_WHAT} ] .
318 [ @he|EQD @king|EQK ] .
319
320 /* The king may not castle with a rook that has already moved. */
321 /* The king may not castle with a rook that has moved. */
322 [ #The king|MODAL_WHO may_not|MODAL_MOD [ castle|ACT #with #a
323   [ ^rook|PAT #that #has moved|ACT ]|INSTP ]|MODAL_WHAT ] .
324
325 /* If the square on which the king stands, or the square which it must cross, or the square which it
326  * is to occupy, is attacked by one or more of the opponent's pieces, the king may not castle. */
327 [ #If [ { [ #the ^square|PAT #on #which #the king|AG stands|ACT ], #or
328           [ #the ^square|PAT #which it|AG must_cross|ACT ] OR
329           [ #the ^square|PAT #which @it|AG #is #to occupy|ACT ] }|PAT #is attacked|ACT #by *one_or_more
330             [#of #the opponent's|POSS ^pieces|HAB ]|AG ]|SUM #the
331   @king|{AG,MODAL_WHO} may_not|MODAL_MOD castle|{ACT,MODAL_WHAT} ] .
332 [ @it|EQD @king|EQK ] .
333
334 /* If there is any piece between the king and the rook with which castling is to be effected,
335 * the king may not castle */
336 [ #If [ #there #is *any piece|ACT #between #the king|DUX #and #the
337   [ ^rook|PAT #{with which} castling|ACT #is #to #be #effected ]|COM ]|SUM #the
338   @king|{AG,MODAL_WHO} may_not|MODAL_MOD castle|{ACT,MODAL_WHAT} ] .
339
340
341 /* 3.9. The king is said to be 'in check' if it is attacked by one or more of the opponent's pieces,
342  * even if such pieces are constrained from moving to that square because they would then leave or
343  * place their own king in check. */
344 [ #The king|STATII #is #said #to #be in_check|STAT #if
345     [ it|PAT #is attacked|ACT #by *one_or_more #of
346         [ #the opponent's|POSS ^pieces|HAB
347         ]|AG #{even if such pieces are constrained from moving to that square because they would then}
348             #{leave or place their own king in check }
349     ]|SUM
350 ] .
351 [ @it|EQD @king|EQK ] .
352
353 /* Article 4: The act of moving the pieces */
354 /* 4.1. Each move must be made with one hand only. */
355 [ *Each move|PAT must|MODAL_MOD be_made|{ACT,MODAL_WHAT} #with *one hand|INSTP #only ] .
356
357 /* 4.2. Provided that he first expresses his intention (e.g. by saying ''j'adoube'' or ''I adjust''),
358  * the player having the move may adjust one or more pieces on their squares. */
359 [ [ #Provided #that he|{AG,POSS} $first expresses|ACT #his intention|{HAB,PAT} ]|SUM #the
360     player|{AG,MODAL_WHO,STATII} having_the_move|STAT may|MODAL_MOD
361   [ adjust|ACT *one_or_more pieces|PAT on_their_squares|LOC_POS ]|MODAL_WHAT ] .
362 [ @he|EQD player|EQK ] .
```

```
363
364 /* 4.3. Except as provided in Article 4.2, if the player having the move deliberately touches on
365  *      the chessboard
366  * a. one or more of his own pieces, he must move the first piece touched that can be moved, or
367  * b. one or more of his opponent's pieces, he must capture the first piece touched, which can be
368  *    captured, or
369  * c. one piece of each colour, he must capture the opponent's piece with his piece or, if this is illegal,
370  *    move or capture the first piece touched which can be moved or captured. */
371
372 /* Provided that he not expresses his intention to adjust, if the player having the move deliberately
373  * touches on the chessboard one or more pieces of his own, he must move the first touched piece that
374  * can be moved. */
375 [
376     {
377     sec_4_3_cond:PHRASE[#Provided #that he|{AG,POSS} not_expresses|ACT
378       [ his|POSS ^intention_to_adjust|HAB ]|PAT] AND
379     [ #if #the player|{AG,STATII} having_the_move|STAT $deliberately touches|AC
380       on_the_chessboard|LOC_POS *one_or_more pieces|{HAB,PAT} #of @his|POSS #own ]
381     }|SUM
382   he|MODAL_WHO must|MODAL_MOD
383   [ move|ACT #the $first $touched piece|{PAT, MODAL_WHO} #that $can_be_moved<<2 ]|MODAL_WHAT
384 ] .
385 [ @piece|PAR @pieces|OMN ] .
386 [ @he|EQD @player|EQK ] .
387 [ @his|EQD @player|EQK ] .
388
389 /* Provided that he not expresses his intention to adjust, if the player having the move deliberately
390  * touches on the chessboard one or more pieces of his opponent's, he must capture the first touched
391  * piece, which can be captured. */
392 [
393     {
394         @sec_4_3_cond AND
395         [ #the player|{AG,STATII} having_the_move|STAT $deliberately touches|ACT on_the_chessboard|LOC_POS
396           *one_or_more pieces|{PAT,HAB} #of his|CONTII opponent|{CONT,POSS} ]
397     }|SUM
398     he|MODAL_WHO must|MODAL_MOD [ capture|ACT #the $first $touched piece|{PAT, MODAL_WHO} #which
399       $can_be_captured<<2 ]|MODAL_WHAT
400 ] .
401
402 [ piece|PAR pieces|OMN ] .
403 [ @he|EQD @player|EQK ] .
404 [ @his|EQD @player|EQK ] .
405
406 /* Provided that he not expresses his intention to adjust, if the player having the move deliberately
407  * touches on the chessboard one piece of each colour, he must capture the opponent's piece with
408  * his piece. */
409 [ { @sec_4_3_cond AND [ #the player|{AG,STATII} having_the_move|STAT $deliberately touches|ACT
410     on_the_chessboard|LOC_POS *one piece|{PAT,QUALII} #of $each colour|QUAL ]}|SUM
411   he|{AG,MODAL_WHO} must|MODAL_MOD [ capture|ACT #the [ opponent's|POSS ^piece|HAB ]|PAT #with
412   [ his|POSS ^piece|HAB ]|INST ]|MODAL_WHAT ] .
413 [ @he|EQD @player|EQK ] .
414 [ @his|EQD @player|EQK ] .
415 [ @player|CONT @opponent's|CONTII ] .
416
417 /* Provided that he not expresses his intention to adjust, if the player having the move deliberately
418  * touches on the chessboard one piece of each colour, and if he may not capture the opponent's piece
419  * with his piece, he must move the first touched piece which can be moved or capture the first touched
420  * piece that can be  captured. */
421 [
422     {
423         @sec_4_3_cond,
424         [ #the player|{AG,STATII} having_the_move|STAT $deliberately touches|ACT on_the_chessboard|LOC_POS
425           *one piece|{PAT,QUALII} #of $each colour|QUAL ] AND
```

```
426         [ #if he|{AG,MODAL_WHO} may_not|MODAL_MOD [ capture|ACT #the [ opponent's|POSS piece_o|HAB ]|PAT
427           #with [ his|POSS ^piece_p|HAB ]|MOD]|MODAL_WHAT ]
428     }|SUM
429     he|{AG,MODAL_WHO} must|MODAL_MOD [ move|ACT #the $first $touched
430         [ ^piece|PAT #which can|MODAL_MOD be_moved|ACT ]|PAT
431     ]|MODAL_WHAT
432 ] .
433 /* MENGE
434 piece_o <= piece.
435 piece_p <= piece. */
436 [ @he|EQD @player|EQK ] .
437 [ @his|EQD @player|EQK ] .
438 [ @player|CONT @opponent's|CONTII ] .
439
440 /* 4.4. a. If a player deliberately touches his king and rook he must castle on that side if it is legal
441  * to do so. */
442 /* If a player deliberately touches his king and rook and if it is legal to castle,
443  * he must castle on that side. */
444 /* If a player deliberately touches his king and rook and if it is legal to castle,
445  * he must castle with that rook. */
446 [
447     {
448         sec_4_4_a_cond:PHRASE[ #If #a player|AG $deliberately touches|ACT
449             {
450                 [ his|POSS ^king|HAB ] AND
451                 [ his|POSS ^rook|HAB ]
452             }|PAT
453         ] AND
454         [ #if #it #is legal|STATII #to castle|STAT ]
455     }|SUM
456     he|{AG,MODAL_WHO} must|MODAL_MOD [ castle|ACT #with #that @rook|INSTP ]|MODAL_WHAT
457 ] .
458 [ @he|EQD @player|EQK ] .
459
460 /* b. If a player deliberately touches a rook and then his king he is not allowed to castle on that side
461  * on that move and the situation shall be governed by Article 4.3(a). */
462 /* If a player deliberately touches a rook and then his king, he is not allowed to castle on that side
463  * on that move. */
464 // shall be governed by Article 4.3(a) --> he must move with the first piece touched that can be moved.
465 /* If a player deliberately touches a rook and then his king, he must make a move with the first piece
466  * touched that can be moved. */
467 [
468     sec_4_4_b_cond:PHRASE[ #If #a player|AG
469         [ $deliberately ^touches|ACT his|POSS king|{HAB,PAT} ]|PRE #and #then
470         [ $deliberately ^touches|ACT his|POSS rook|{HAB,PAT} ]|SUCC
471     ]|SUM
472     he|{AG,MODAL_WHO} may_not|MODAL_MOD
473       [ castle|ACT on_that_side|LOC_POS on_that_move|TEMP_POS ]|MODAL_WHAT
474 ] .
475
476 [ @he|EQD @player|EQK ] .
477
478 [ @sec_4_4_b_cond|SUM he|{AG,MODAL_WHO} must|MODAL_MOD [ make_a_move|ACT #with #the $first $touched
479   piece|{PAT,STATII} #that can_be_moved|STAT ]|MODAL_WHAT ] .
480 [ @he|EQD @player|EQK ] .
481
482 /* c. If a player, intending to castle, touches the king or king and rook at the same time, but castling
483  *    on that side is illegal, the player must make another legal move with his king which may include
484  *    castling on the other side. */
485 /* If a player intends to castle and the player touches the king or the king and a rook at the same time
486  * and castling on that side is illegal, the player must make another legal move wirh his king which may
487  * include castling on the other side. */
488 /* SALE kann keine Nested-Sets, deswegen Doppelung*/
```

```
[
    {
        sec_4_4_c_cond_1:PHRASE[ #If #a player|AG intends_to_castle|ACT], #and
        sec_4_4_c_cond_2_a:PHRASE[ #the @player|AG touches|ACT #the king|PAT ] AND
        sec_4_4_c_cond_3:PHRASE[ castling|STAT on_that_side|LOC_POS is_illegal|STAT]
    }|SUM #the @player|{AG,MODAL_WHO} must|MODAL_MOD [
        make|ACT $another $legal move|{PAT,OMN} #with his|POSS king|{HAB,INST} #which #may #include
          [ ^castling|ACT on_the_other_side|LOC_POS ]|PAR
    ]|MODAL_WHAT
] .

[
    {
        @sec_4_4_c_cond_1,
        sec_4_4_c_cond_2_b:PHRASE[ #the @player|AG touches|ACT
          { #the king AND #a rook}|PAT at_the_same_time|TEMP_POS ] AND
        @sec_4_4_c_cond_3
    }|SUM #the @player|{AG,MODAL_WHO} must|MODAL_MOD [
        make|ACT $another $legal move|{PAT,OMN} #with his|POSS king|{HAB,INST}
        #which #may #include [ ^castling|ACT on_the_other_side|LOC_POS ]|PAR
    ]|MODAL_WHAT
] .
[ @his|EQD @player|EQK ] .

/* If the king has no legal move, the player is free to make any legal move. */
/* If a player intends to castle and the player touches the king or the king and a rook at the same time
 * and castling on that side is illegal and the king has no legal move, the player may make any legal
 * move. */
[
    {
        sec_4_4_c_cond_1,
        sec_4_4_c_cond_2,
        sec_4_4_c_cond_3 AND
        [#if #the king|POSS #has *no $legal move|HAB ]
    }|SUM #the player|{AG,MODAL_WHO} may|MODAL_MOD [ make|ACT *any $legal move|PAT ]|MODAL_WHAT
] .

/* d. If a player promotes a pawn, the choice of the piece is finalised, when the piece has touched the
 *    square of promotion. */
[
    {
        [ #If #a player|AG promotes|ACT #a pawn|PAT] AND
        [#the piece|AG has_touched|ACT #the square_of_promotion|PAT ]
    }|SUM #the choice|{THEII, STATII} #of #the @piece|THE is_finalised|STAT
] .

/* 4.5. If none of the pieces touched can be moved or captured, the player may make any legal move. */
/* If the player cannot move any of touched pieces of his own and if the player cannot capture any of the
 * touched pieces of his opponent's, the player may make any legal move. */
[
    {
        [ #If #the player|{AG,MODAL_WHO} cannot|MODAL_MOD
          [ move|ACT *any $touched piece|{PAT,HAB} #of his|POSS #own]|MODAL_WHAT ] AND
        [ #if #the @player|{AG,MODAL_WHO} cannot|MODAL_MOD
          [ capture|ACT *any #of #the $touched pieces|{PAT,HAB} #of his|CONTII opponent's|{CONT,POSS}
          ]|MODAL_WHAT ]
    }|SUM #the player|{AG,MODAL_WHO} may|MODAL_MOD [ make|ACT *any $legal move|PAT ]|MODAL_WHAT
] .

/* 4.6. When, as a legal move or part of a legal move, a piece has been released on a square,
 *      it cannot then be moved to another square. */
/* When as a legal move a piece has been released on a square, it cannot then be moved
 * to another square. */
```

```
552 /* When as a part of a legal move, a piece has been released on a square,
553  * it cannot then be moved to another square. */
554 [ [ #When #as #a #part #of #a $legal move|OMN #a piece|PAT has_been_released|{ACT,PAR} #on #a
555     square|LOC_POS ]|SUM it|PAT cannot|MODAL_MOD #then be_moved|ACT #to $another square|LOC_DEST ] .
556 [ @it|EQD @piece|EQK ] .
557
558 /* If a piece has been captured and the captured piece has been removed from the chessboard and the player
559  * has placed his own capturing piece on its new square and the player has released this piece from his
560  * hand, the move has been made. */
561 [ #If
562     {
563         [ #a piece|PAT #has #been captured|ACT ], #and
564         [ #the $captured @piece|PAT #has #been removed|ACT #from #the chessboard|LOC_DEST ], #and
565         [ #the player|AG #has placed|ACT [ his|POSS #own $capturing ^piece|HAB ]|PAT #on
566           [ its|POSS $new ^square|HAB ]|LOC_DEST ] AND
567         [ #the @player|AG #has released|ACT #this @piece|PAT #from #his hand|LOC_ORIG ]
568     }|SUM #the
569   move|STATII has_been_made|STAT
570 ] .
571
572 /* If the player castles and the player has released the rook on the square previously crossed by the
573  * king, the move has been made. */
574 [ #If
575     {
576         [ #the player|AG castles|ACT ] AND
577         [ #the @player|AG #has released|ACT #the rook|PAT #on #the
578           [ ^square|PAT $previously crossed|ACT #by #the king|AG ]|LOC_DEST ]
579     }|SUM #the
580     move|STATII has_been_made|STAT
581 ] .
582
583 /* If the player castles and the player has released the king from his hand, the move is not yet made
584  * but the player no longer has the right to make any other move than castling on that side, if this is
585  * legal. */
586 /* If the player castles and the player has released the king from his hand, the move is not yet made
587  * and the player may only castle on that side. */
588 [ #If
589     {
590         [ #the player|AG castles|ACT ] AND
591         [ #the @player|AG #has released|ACT #the king|PAT #from #his hand|LOC_ORIG ]
592     }|SUM #the
593     move|STAT #is not_yet_made|STATII #but #the player|{AG,MODAL_WHO} may|MODAL_MOD
594       [ castle|AG on_that_side|LOC_POS ]|MODAL_WHAT #{if this is legal}
595 ] .
596
597
598 /* If the player promotes a pawn and the pawn has been removed from the chessboard and the player's hand
599  * has released the new piece after placing it on the promotion square, the move has been made. */
600 [ #If
601     { [ #the player|AG promotes|ACT #a pawn|PAT ], #and
602         [ #the @pawn|PAT #has #been removed|ACT #from #the chessboard|LOC_ORIG ] AND
603         [
604             [ #the player's|POSS hand|{HAB,AG} #has released|ACT #the $new piece|PAT ]|PRE #after
605             [ placing|ACT it|PAT #on #the promotion_square|LOC_DEST ]|SUCC
606         ]
607     }|SUM #the
608     move|STATII has_been_made|STAT
609 ] .
610 [ @it|EQD @piece|EQK ] .
611
612 /* If the player promotes a pawn and the player has released the pawn from his hand on the promotion
613  * square, the move is not yet made and the player may not play the pawn to another square. */
614 [ #If
```

```
    {
        [ #the player|AG promotes|ACT #a pawn|PAT ] AND
        [ #the @player|AG #has released|ACT #the pawn|PAT #from #his hand|LOC_ORIG #on #the
          promotion_square|LOC_DEST ]
    }|SUM
    {
        [#the move|STATII #is not_made|STAT] AND
        [ #the player|{AG,MODAL_WHO} may_not|MODAL_MOD
            [ play|ACT #the pawn|PAT #to $another square|LOC_DEST ]|MODAL_WHAT
        ]
    }|SUMII
] .

/* 4.7. A player forfeits his right to a claim against his opponent's violation of Article
 * 4.3 or 4.4, once he deliberately touches a piece. */
[ #A player|AG forfeits|ACT
    [ his|POSS ^right_to_a_claim|{THEII,HAB} #against
        [ @his|CONTII opponent's|CONT ^violation|THEII #of { Article_4_3 OR  Article_4_4 }|THE
        ]|THE
    ]|PAT
    [ #once he|AG $deliberately touches|ACT #a piece|PAT ]|SUM
] .
[ @he|EQD @player|EQK ] .

/* Article 5: The completion of the game */

/* 5.1. a. The game is won by the player who has checkmated his opponent's king. */
/* If a player checkmates his opponent's king, he wins the game. */
[ sec_5_1_a_cond:PHRASE[ #If #a player|{AG,CONTII} checkmates|ACT #his opponent's|{POSS,CONT}
  king|{HAB,PAT} ]|SUM he|AG wins|ACT #the game|PAT ] .
[ @he|EQD @player@EQK ] .


/* This immediately ends the game, provided that the move producing the checkmate position was
 * a legal move. */
/* If a player checkmates his opponent's king and the move|CREA producing the checkmate
 * position|OPUS was a legal move, the game immediately ends. */
[ { [ #If #a player|AG checkmates|ACT [ #his opponent's|POSS ^king|HAB ]|PAT ] AND
    [ #the move|CREA #producing #the checkmate_position|OPUSP #was $legal<<2]
  }|SUM #the game|AG $immediately ends|ACT ] .
[ player|CONTII opponent's|CONT ] .

/* b. The game is won by the player whose opponent declares he resigns.  */
/* If a player resigns, his opponent wins the game. */
[ sec_5_1_b_cond:PHRASE[ #If #a player|AG resigns|ACT]|SUM his|CONTII opponent|{AG,CONT}
  wins|ACT #the game|PAT ] .
[ @his|EQD @player@EQK ] .

/* This immediately ends the game. */
/* If @sec_5_1_b_cond, the game immediately ends. */
[ #If @sec_5_1_b_cond|SUM #the game|AG $immediately ends|ACT ] .

/* 5.2. a. The game is drawn when the player to move has no legal move and his king is not in check. */
/* If the player having the move cannot make a move and his king is not in check, the game is drawn. */
[
    {
        sec_5_2_a_cond_1:PHRASE [ #If #the player|{AG,MODAL_WHO,STATII} having_the_move|STAT
          cannot|MODAL_MOD make_a_move|MODAL_WHAT ] AND
        sec_5_2_a_cond_2:PHRASE[ his|POSS king|{HAB,STATII} is_not_in_check|STAT ]
    }|SUM #the game|STATII is_drawn|STAT
] .
[ @his|EQD @player@EQK ] .

```

```
678 /* The game is said to end in 'stalemate'. */
679 /* If sec_5_2_a_cond, the game ends in stalemate. */
680 /* FIXED (3.12.2007 Tom): end|ACT und game|STATII stalemate|STAT */
681 [ {@sec_5_2_a_cond_1 AND @sec_5_2_a_cond_2}|SUM #the game|AG ends_in_stalemate|ACT ] .
682
683 /* This immediately ends the game, provided that the move producing the stalemate position was legal. */
684 /* If the game ends in stalemate, the game immediately ends. */
685 [ #If [#the game|{AG,STATII} ends|ACT #in stalemate|STAT ]|SUM #the game|AG $immediately ends|ACT ] .
686
687 /* b. The game is drawn when a position has arisen in which neither player can checkmate the opponent's
688  *    king with any series of legal moves. */
689 /* If a position has arisen in which neither player can checkmate the opponent's king with any series of
690  * legal moves, the game is drawn. */
691 /* If neither player can checkmate the opponent's king with any series of legal moves,
692  * the game is drawn. */
693 [
694     sec_5_2_b_cond:PHRASE[ #If *neither player|{AG,MODAL_MOD} can|MODAL_MOD
695         [ checkmate|ACT #the opponent's|POSS king|HAB #with *any
696             [ ^series|OMN #of $legal moves|PAR
697             ]|INSTP
698         ]|MODAL_WHAT
699     ]|SUM #the game|STATII is_drawn|STAT
700 ] .
701
702 /* The game is said to end in a 'dead position'. */
703 /* If @sec_5_2_b_cond, the game ends in a dead_position. */
704 [ #If @sec_5_2_b_cond|SUM #the game|{AG,STATII} ends|ACT #in #a dead_position|STAT ] .
705
706 /* This immediately ends the game, provided that the move producing the position was legal. */
707 /* If the game is in a dead position, the game immediately ends. */
708 [ [ #If #the game|STATII #is #in #a dead_position|STAT ]|SUM #the game|AG $immediately ends|ACT ] .
709
710 /* c. The game is drawn upon agreement between the two players during the game. */
711 /* If the players agree to drawing the game, the game is drawn. */
712 [ sec_5_2_c_cond:PHRASE[ #If #the players|AG agree|ACT #to drawing_the_game|PAT ]|SUM
713   #the game|PAT is_drawn|ACT ] .
714
715 /* This immediately ends the game. (See Article 9.1) */
716 /* If the game is drawn, the game ends. */
717 [ [ #If #the game|STATII is_drawn|STAT ]|SUM #the game|AG $immediately ends|ACT ] .
718
719 /* d. The game may be drawn if any identical position is about to appear or has appeared on the chessboard
720  *    at least three times. (See Article 9.2) */
721 /* If any identical position is about to appear on the cheassboard for the third time or has appeared on
722  * the chessboard at least three times, the game may be drawn. */
723 [ #If
724   {
725     [ *any $identical position|AG is_about_to_appear|ACT on_the_chessboard|LOC_POS
726       for_the_third_time|FREQ ] OR
727     [ *any $identical position|AG has_appeared|ACT on_the_chessboard|LOC_POS
728       at_least_three_times|FREQ ]
729   }|SUM #the game|PAT may|MODAL_MOD be_drawn|{ACT,MODAL_WHAT}
730 ] .
731
732 /* e. The game may be drawn if each player has made at least the last 50 consecutive moves without the
733  *    movement of any pawn and without any capture. (See Article 9.3) */
734 [ #The game|PAT may|MODAL_MOD be_drawn|MODAL_WHAT
735     [ #if *each player|AG has_made|ACT *at_least_the_last_50 $consecutive moves|{OMN,PAT} #without #the
736         { [ ^movement|ACT #of #any pawn|PAT ] AND #without #any capture
737         }|NOT PAR
738     ]|SUM
739 ] .
740
```

```
741 /* Article 4: The act of moving the pieces */
742 /* 4.1. Each move must be made with one hand only. */
743 [ *Each move|PAT must|MODAL_MOD be_made|{ACT,MODAL_WHAT} #with one_hand_only|INSTP ] .
744
745 /* 4.2. Provided that he first expresses his intention (e.g. by saying ''j'adoube'' or ''I adjust''),
746  *      the player having the move may adjust one or more pieces on their squares. */
747 [ [ #Provided #that he|{AG,POSS} $first expresses|ACT #his intention|{HAB,PAT} ]|SUM #the
748     player|{AG,MODAL_WHO,STATII} having_the_move|STAT may|MODAL_MOD
749       [ adjust|ACT *one_or_more pieces|PAT on_their_squares|LOC_POS ]|MODAL_WHAT ] .
750 [ @he|EQD player|EQK ] .
751
752 /* 4.3. Except as provided in Article 4.2, if the player having the move deliberately touches on the
753  *      chessboard
754  * a. one or more of his own pieces, he must move the first piece touched that can be moved, or
755  * b. one or more of his opponent's pieces, he must capture the first piece touched, which can be
756  *    captured, or
757  * c. one piece of each colour, he must capture the opponent's piece with his piece or, if this is
758  *    illegal, move or capture the first piece touched which can be moved or captured. */
759
760 /* Provided that he not expresses his intention to adjust,  if the player having the move deliberately
761  * touches on the chessboard one or more pieces of his own, he must move the first touched piece that
762  * can be moved. */
763 [
764     {
765     sec_4_3_cond:PHRASE[#Provided #that he|{AG,POSS} not_expresses|ACT
766       [ his|POSS ^intention_to_adjust|HAB ]|PAT] AND
767     [ #if #the player|{AG,STATII} having_the_move|STAT $deliberately touches|ACT on_the_chessboard|LOC_POS
768       *one_or_more pieces|{HAB,PAT} #of @his|POSS #own ]
769     }|SUM
770   he|MODAL_WHO must|MODAL_MOD
771   [ move|ACT #the $first $touched piece|{PAT, MODAL_WHO} #that $can_be_moved<<2 ]|MODAL_WHAT
772 ] .
773 [ @piece|PAR @pieces|OMN ] .
774 [ @he|EQD @player|EQK ] .
775 [ @his|EQD @player|EQK ] .
776
777 /* Provided that he not expresses his intention to adjust, if the player having the move deliberately
778  * touches on the chessboard one or more pieces of his opponent's, he must capture the first touched
779  * piece, which can be captured. */
780 [
781     {
782         @sec_4_3_cond AND
783         [ #the player|{AG,STATII} having_the_move|STAT $deliberately touches|ACT on_the_chessboard|LOC_POS
784           *one_or_more pieces|{PAT,HAB} #of his|CONTII opponent|{CONT,POSS} ]
785     }|SUM
786     he|MODAL_WHO must|MODAL_MOD [ capture|ACT #the $first $touched piece|{PAT, MODAL_WHO} #which
787     $can_be_captured<<2 ]|MODAL_WHAT
788 ] .
789 [ piece|PAR pieces|OMN ] .
790 [ @he|EQD @player|EQK ] .
791 [ @his|EQD @player|EQK ] .
792
793 /* Provided that he not expresses his intention to adjust, if the player having the move deliberately
794  * touches on the chessboard one piece of each colour, he must capture the opponent's piece with his
795  * piece. */
796 [ { @sec_4_3_cond AND [ #the player|{AG,STATII} having_the_move|STAT $deliberately touches|ACT
797     on_the_chessboard|LOC_POS *one piece|{PAT,QUALII} #of $each colour|QUAL ]}|SUM
798   he|{AG,MODAL_WHO} must|MODAL_MOD [ capture|ACT #the [ opponent's|POSS ^piece_o|HAB ]|PAT #with
799     [ his|POSS ^piece_p|HAB ]|INST ]|MODAL_WHAT ] .
800 [ @he|EQD @player|EQK ] .
801 [ @his|EQD @player|EQK ] .
802 [ @player|CONT @opponent's|CONTII ] .
803
```

```
804 /* Provided that he not expresses his intention to adjust, if the player having the move deliberately
805  * touches on the chessboard one piece of each colour, and if he may not capture the opponent's piece
806  * with his piece, he must move the first touched piece which can be moved or capture the first touched
807  * piece that can be  captured. */
808 [
809     {
810         @sec_4_3_cond,
811         [ #the player|{AG,STATII} having_the_move|STAT $deliberately touches|ACT on_the_chessboard|LOC_POS
812           *one piece|{PAT,QUALII} #of $each colour|QUAL ] AND
813         [ #if he|{AG,MODAL_WHO} may_not|MODAL_MOD [ capture|ACT #the [ opponent's|POSS piece_o|HAB ]|PAT
814           #with [ his|POSS ^piece_p|HAB ]|MOD]|MODAL_WHAT ]
815     }|SUM
816     he|{AG,MODAL_WHO} must|MODAL_MOD [ move|ACT #the $first $touched
817         [ ^piece|PAT #which can|MODAL_MOD be_moved|ACT ]|PAT
818     ]|MODAL_WHAT
819 ] .
820 [ @he|EQD @player|EQK ] .
821 [ @his|EQD @player|EQK ] .
822 [ @player|CONT @opponent's|CONTII ] .
823
824 /* 4.4. a. If a player deliberately touches his king and rook he must castle on that
825  * side if it is legal to do so. */
826 /* If a player deliberately touches his king and rook and if it is legal to castle,
827  * he must castle with that rook.. */
828 [
829     {
830         sec_4_4_a_cond:PHRASE[ #If #a player|AG $deliberately touches|ACT
831             {
832                 [ his|POSS ^king|HAB ] AND
833                 [ his|POSS ^rook|HAB ]
834             }|PAT
835         ] AND
836         [ #if #it #is legal|STATII #to castle|STAT ]
837     }|SUM
838     he|{AG,MODAL_WHO} must|MODAL_MOD [ castle|ACT #with #that @rook|INSTP ]|MODAL_WHAT
839 ] .
840 [ @he|EQD @player|EQK ] .
841
842 /* b. If a player deliberately touches a rook and then his king he is not allowed to castle on that side
843  * on that move and the situation shall be governed by Article 4.3(a). */
844 /* If a player deliberately touches a rook and then his king, he is not allowed to castle on that side on
845  * that move. */
846 // shall be governed by Article 4.3(a) --> he must move with the first piece touched that can be moved.
847 /* If a player deliberately touches a rook and then his king, he must make a move with the first piece
848  * touched that can be moved. */
849 [
850     sec_4_4_b_cond:PHRASE[ #If #a player|AG
851         [ $deliberately ^touches|ACT his|POSS king|{HAB,PAT} ]|PRE #and #then
852         [ $deliberately ^touches|ACT his|POSS rook|{HAB,PAT} ]|SUCC
853     ]|SUM
854     he|{AG,MODAL_WHO} may_not|MODAL_MOD
855       [ castle|ACT on_that_side|LOC_POS on_that_move|TEMP_POS ]|MODAL_WHAT
856 ] .
857
858 [ @he|EQD @player|EQK ] .
859
860 [ @sec_4_4_b_cond|SUM he|{AG,MODAL_WHO} must|MODAL_MOD [ make_a_move|ACT #with #the $first $touched
861   piece|{PAT,STATII} #that can_be_moved|STAT ]|MODAL_WHAT ] .
862 [ @he|EQD @player|EQK ] .
863
864 /* c. If a player, intending to castle, touches the king or king and rook at the same time, but castling
865  * on that side is illegal, the player must make another legal move with his king which may include
866  * castling on the other side. */
```

```
867 /* If a player intends to castle and the player touches the king or the king and a rook at the same time
868  * and castling on that side is illegal, the player must make another legal move wirh his king which may
869  * include castling on the other side. */
870 /* SALE kann keine Nested-Sets, deswegen doppelung*/
871 [
872     {
873         sec_4_4_c_cond_1:PHRASE[ #If #a player|AG intends_to_castle|ACT], #and
874         sec_4_4_c_cond_2_a:PHRASE[ #the @player|AG touches|ACT #the king|PAT ] AND
875         sec_4_4_c_cond_3:PHRASE[ castling|STAT on_that_side|LOC_POS is_illegal|STAT]
876     }|SUM #the @player|{AG,MODAL_WHO} must|MODAL_MOD [
877         make|ACT $another $legal move|{PAT,OMN} #with his|POSS king|{HAB,INST} #which #may #include
878         [ ^castling|ACT on_the_other_side|LOC_POS ]|PAR
879     ]|MODAL_WHAT
880 ] .
881
882 [
883     {
884         @sec_4_4_c_cond_1,
885         sec_4_4_c_cond_2_b:PHRASE[ #the @player|AG touches|ACT { #the king AND #a rook}|PAT
886           at_the_same_time|TEMP_POS ] AND
887         @sec_4_4_c_cond_3
888     }|SUM #the @player|{AG,MODAL_WHO} must|MODAL_MOD [
889         make|ACT $another $legal move|{PAT,OMN} #with his|POSS king|{HAB,INST} #{which may include}
890           [ ^castling|ACT on_the_other_side|LOC_POS ]|PAR
891     ]|MODAL_WHAT
892 ] .
893 [ @his|EQD @player|EQK ] .
894
895 /* If the king has no legal move, the player is free to make any legal move. */
896 /* If a player intends to castle and the player touches the king or the king and a rook at the same time
897  * and castling on that side is illegal and the king has no legal move, the player may make
898  * any legal move. */
899 [
900     {
901         sec_4_4_c_cond_1,
902         sec_4_4_c_cond_2,
903         sec_4_4_c_cond_3 AND
904         [#if #the king|POSS #has *no $legal move|HAB ]
905     }|SUM #the player|{AG,MODAL_WHO} may|MODAL_MOD [ make|ACT *any $legal move|PAT ]|MODAL_WHAT
906 ] .
907
908 /* d. If a player promotes a pawn, the choice of the piece is finalised, when the piece has touched the
909  *    square of promotion. */
910 [
911     {
912         [ #If #a player|AG promotes|ACT #a pawn|PAT] AND
913         [#the piece|AG has_touched|ACT #the square_of_promotion|PAT ]
914     }|SUM #the choice|{THEII, STATII} #of #the @piece|THE is_finalised|STAT
915 ] .
916
917 /* 4.5. If none of the pieces touched can be moved or captured, the player may make any legal move. */
918 /* If the player cannot move any of touched pieces of his own and if the player cannot capture any of the
919  * touched pieces of his opponent's, the player may make any legal move. */
920 [
921     {
922         [ #If #the player|{AG,MODAL_WHO} cannot|MODAL_MOD [ move|ACT *any $touched piece|{PAT,HAB}
923           #of his|POSS #own]|MODAL_WHAT ] AND
924         [ #if #the @player|{AG,MODAL_WHO} cannot|MODAL_MOD [ capture|ACT *any #of #the $touched
925           pieces|{PAT,HAB} #of his|CONTII opponent's|{CONT,POSS} ]|MODAL_WHAT ]
926     }|SUM #the player|{AG,MODAL_WHO} may|MODAL_MOD [ make|ACT *any $legal move|PAT ]|MODAL_WHAT
927 ] .
928
929 /* 4.6. When, as a legal move or part of a legal move, a piece has been released on a square,
```

```
930  *      it cannot then be moved to another square. */
931 /* When as a legal move a piece has been released on a square,
932  * it cannot then be moved to another square. */
933 /* When as a part of a legal move, a piece has been released on a square,
934  * it cannot then be moved to another square. */
935 [ [ #When #as #a #part #of #a $legal move|OMN #a piece|PAT has_been_released|{ACT,PAR} #on #a
936     square|LOC_POS ]|SUM it|PAT cannot|MODAL_MOD #then be_moved|ACT #to $another square|LOC_DEST ] .
937 [ @it|EQD @piece|EQK ] .
938
939 /* If a piece has been captured and the captured piece has been removed from the chessboard and the
940  * player has placed his own capturing piece on its new square and the player has released this piece
941  * from his hand, the move has been made. */
942 [ #If
943     {
944         [ #a piece|PAT #has #been captured|ACT ], #and
945         [ #the $captured @piece|PAT #has #been removed|ACT #from #the chessboard|LOC_DEST ], #and
946         [ #the player|AG #has placed|ACT [ his|POSS #own $capturing ^piece|HAB ]|PAT #on
947           [ its|POSS $new ^square|HAB ]|LOC_DEST ] AND
948         [ #the @player|AG #has released|ACT #this @piece|PAT #from #his hand|LOC_ORIG ]
949     }|SUM #the
950   move|STATII has_been_made|STAT
951 ] .
952
953 /* If the player castles and the player has released the rook on the square previously crossed by the
954  * king, the move has been made. */
955 [ #If
956     {
957         [ #the player|AG castles|ACT ] AND
958         [ #the @player|AG #has released|ACT #the rook|PAT #on #the
959           [ ^square|PAT $previously crossed|ACT #by #the king|AG ]|LOC_DEST ]
960     }|SUM #the
961     move|STATII has_been_made|STAT
962 ] .
963
964 /* If the player castles and the player has released the king from his hand, the move is not yet made but
965  * the player no longer has the right to make any other move than castling on that side, if this is
966  * legal. */
967 /* If the player castles and the player has released the king from his hand, the move is not yet made
968  * and the player may only castle on that side. */
969 [ #If
970     {
971         [ #the player|AG castles|ACT ] AND
972         [ #the @player|AG #has released|ACT #the king|PAT #from #his hand|LOC_ORIG ]
973     }|SUM #the
974     move|STAT #is not_yet_made|STATII #but #the player|{AG,MODAL_WHO} may|MODAL_MOD
975       [ castle|AG on_that_side|LOC_POS ]|MODAL_WHAT #{if this is legal}
976 ] .
977
978
979 /* If the player promotes a pawn and the pawn has been removed from the chessboard and the player's hand
980  * has released the new piece after placing it on the promotion square, the move has been made. */
981 [ #If
982     { [ #the player|AG promotes|ACT #a pawn|PAT ], #and
983         [ #the @pawn|PAT #has #been removed|ACT #from #the chessboard|LOC_ORIG ] AND
984         [
985             [ #the player's|POSS hand|{HAB,AG} #has released|ACT #the $new piece|PAT ]|PRE #after
986             [ placing|ACT it|PAT #on #the promotion_square|LOC_DEST ]|SUCC
987         ]
988     }|SUM #the
989     move|STATII has_been_made|STAT
990 ] .
991 [ @it|EQD @piece|EQK ] .
992
```

```
/* If the player promotes a pawn and the player has released the pawn from his hand on the promotion
 * square, the move is not yet made and the player may not play the pawn to another square. */
[ #If
    {
        [ #the player|AG promotes|ACT #a pawn|PAT ] AND
        [ #the @player|AG #has released|ACT #the pawn|PAT #from #his hand|LOC_ORIG
          #on #the promotion_square|LOC_DEST ]
    }|SUM
    {
        [#the move|STATII #is not_made|STAT] AND
        [ #the player|{AG,MODAL_WHO} may_not|MODAL_MOD
            [ play|ACT #the pawn|PAT #to $another square|LOC_DEST ]|MODAL_WHAT
        ]
    }|SUMII
] .

/* 4.7. A player forfeits his right to a claim against his opponent's violation of Article 4.3 or 4.4,
 *      once he deliberately touches a piece. */
[ #A player|AG forfeits|ACT
    [ his|POSS ^right_to_a_claim|{THEII,HAB} #against
        [ @his|CONTII opponent's|CONT ^violation|THEII #of { Article_4_3 OR  Article_4_4 }|THE
        ]|THE
    ]|PAT
    [ #once he|AG $deliberately touches|ACT #a piece|PAT ]|SUM
] .
[ @he|EQD @player|EQK ] .
```

D. Die annotierte Fassung der Ludo-Regeln

Listing D.1: Die annotierten Ludo-Regeln.

```
/* Ludo */
/* The board consists of a directed 40 field ring in form of a cross. */
[ #The board|OMN #{consists of a} $directed *40 field|PAR ring|{OMN,FIN,PAR}
  #{in form of a} cross|FIC ] .

/* Every 10th field serves as a starting field for a player. */
[ *Every_10th field|FIN #serves #as #a
  starting_field|{FIC,HAB} #{for a} player|POSS ] .

/* Directly in front of each starting field is a junction to four consecutive goal
  fields of the player according to the starting field. */
/*[ Directly in front of each starting field is a junction to *four $consecutive
  goal_fields of the player according to the starting_field. ]*/

/* 1. There are four players in a cyclic order: red, blue, yellow and green.*/
// Todo: Wie wird die "cyclic order" annotiert?
/* There are four players: One red player, one blue player, one yellow player
  and one green player. */
[ #{There are} *four players|OMN
  { *One $red player,
    *one $blue player,
    *one $yellow player AND
    *one $green player
  }|PAR
] .

/* 2. Every player owns four figures which are not in the game initially. */
[ *Every player|POSS #owns *four figures|HAB ] .
[ #The figures|STATII #are not_in_the_game|STAT initially|TEMP_POS ] .

/* 3. The players throw a six-sided dice one after the other.*/
[ #The @players|AG throw|ACT *a $six_sided dice|PAT one_after_the_other|MOD ] .
// Todo: one after the other

/* 4. If one of the following moves is possible, the player must choose one:*/
/* (a) The player may move any of his figures in the game forward by the exact
  number of dots on the dice. */
[ #The player|{AG,MODAL_WHO} may|MODAL_MOD
  [ move_forward|ACT
    [ #by #the $exact ^number_of_dots|HAB #on #the dice|POSS ]|MOD
    *any #of his|POSS figures|{HAB,PAT} in_the_game|LOC_POS ]|MODAL_WHAT
] .
[ @his|EQD @player|EQK ] .
[ @player|PAR @players|OMN ] .

/* (b) When the player has a six and still has figures outside the game, he may put
  one figure on his start field. */
```

```
48 /* When the player throws a six and the player still has figures outside the game, he
49   may put one figure on his start field. */
50 [ { [ #When #the player|AG throws|ACT #a six|PAT ] AND
51     [ #the @player|POSS #still has|ACT figures|{HAB,PAT} outside_the_game|LOC_POS ]
52   }|SUM
53   he|{AG,MODAL_WHO} may|MODAL_MOD
54   [ put|ACT *one figure|PAT #on his|POSS start_field|{LOC_POS,HAB} ]|MODAL_WHAT
55 ] .
56 [ @he|EQD @player|EQK ].
57 [ @player|PAR @players|OMN ] .
58
59 /* If the destination field of a move is occupied by a figure of its own,
60   the move is not allowed.*/
61 /* If a figure of its own occupies the destination field of a move,
62   the figure must not move. */
63 [ [ #If #a figure_b|{AG,HAB,COM} #of its|{POSS,DUX} #own occupies|ACT #the
64     destination_field|{PAT,HAB} #of #a move|POSS ]|SUM #the
65   figure|{AG,MODAL_WHO} must_not|MODAL_MOD move|{ACT,MODAL_WHAT}
66 ] .
67 [ @its|EQD @figure|EQK ] .
68
69 /* If it is occupied by an opponents figure, the opponents figure is taken
70   off the game. */
71 /* If a figure of its opponent occupies the destination field of a move,
72   the opponents figure is taken off the game. */
73 [ [ #If #a figure|{AG,HAB} #of its|CONTII opponent|{CONT,POSS} occupies|ACT
74   #the destination_field|{PAT,HAB} #of #a move|POSS ]|SUM #the
75   opponents|POSS figure|{HAB,PAT} is_taken_off_the_game|ACT
76 ] .
77 // Todo: Ist die ACT korrekt?
78
79 /* The figures may neither visit nor pass the start field of the according player
80   except from outside the game. */
81 /* If a players figure is inside the game, it may not pass the start field of the
82   according player and may not visit the start field of the according player. */
83 [ [ #If #a players|POSS figure|{STATII,HAB} #is inside_the_game|STAT ]|SUM
84   it|{AG,MODAL_WHO} may_not|MODAL_MOD
85   { [ pass|ACT #the start_field|{PAT,HAB} #of #the $according player|POSS ] AND
86     [ visit|ACT #the start_field|{PAT,HAB} #of #the $according @player|POSS ]
87   }|MODAL_WHAT
88 ] .
89 [ @it|EQD @figure|EQK ] .
90 [ @players|EQD @player|EQK ] .
91 [ @player|PAR @players|OMN ] .
92
93 // Todo: "players" ist das selbe Objekt wie "player"
94 // --> hier über "players = player" gelöst... Das ist nicht elegant!
95 /* Reasoning:
96  * Wie kann festgestellt werden, dass ein dekliniertes Substantiv
97  * (1x Nom, 1x Akk) auf dasselbe Objekt zeigt?
98  */
99
100 /* If a players figure is not inside the game, it may pass the start field of the
101   according player and may visit the start field of the according player. */
102 [ [ #If #a players|POSS figure|{STATII,HAB} #is #not inside_the_game|STAT ]|SUM
103   it|{AG,MODAL_WHO} may|MODAL_MOD
104   { [ pass|ACT #the start_field|{PAT,HAB} #of #the $according player|POSS ] AND #may
105     [ visit|ACT #the start_field|{PAT,HAB} #of #the $according @player|POSS ]
106   }|MODAL_WHAT
107 ] .
108 [ @it|EQD @figure|EQK ] .
109 [ @players|EQD @player|EQK ] .
110 [ @player|PAR @players|OMN ] .
```

```
111
112 /* 5. Having thrown a six, the player must play again,
113   otherwise the next player must play. */
114 /* If a player throws a six, the player must play again,
115   otherwise the next player must play. */
116 /* If a player throws a six, the player must play again. */
117 // Todo: otherwise wird ähnlich aufgelöst, wie except
118 /* Todo: play again != play --> wie soll das again aufgelöst werden?
119  * Tom: Das ist egal - wenn er spielen muss ist es irrelevant, ob er nochmal spielt
120  *  oder ob er spielt.
121  * Mathias: ?!? Ok, wenn das so ist, dann folgt direkt --> #again */
122 [ [ #If #a player|AG throws|ACT #a six|PAT ]|SUM #the
123   @player|{AG,MODAL_WHO} must|MODAL_MOD play|{ACT,MODAL_WHAT} #again ] .
124 [ @player|PAR @players|OMN ] .
125
126 /* If a player throws no six, the next player must play. */
127 [ [ #If #a player|AG throws|ACT *no six|PAT ]|SUM #the
128   $next player|{AG,MODAL_WHO} must|MODAL_MOD play|{ACT,MODAL_WHAT}
129 ] .
130
131 // Todo: muss $next hier verwendet werden, oder kann man es weglassen,
132 // da keine Referenz -->  player != player
133
134 /* 6. After completing a round around the board a figure must enter the
135   goal fields of the according player. */
136 // Todo: After mit If auflösen
137 // Todo: Damit man den according player bestimmen kann, muss man die
138 // Figur mit einem player verknüpfen, dann kann man referenzieren.
139 /* If a players figure completes a round around the board, it must enter
140   the goal fields of the according player. */
141 [ [ #If #a figure|AG completes|ACT #a round|PAT around_the_board|LIM ]|SUM
142   it|{AG,MODAL_MOD} must|MODAL_MOD
143   [ enter|ACT #the goal_fields|{PAT,HAB} #of #the $according player|POSS ]|MODAL_WHAT
144 ] .
145 [ @it|EQD @figure|EQK ] .
146 [ @player|PAR @players|OMN ] .
147
148 /* The first player who fills all four goal fields wins. */
149 // Todo: all wurde auskommentiert --> es gibt insgesamt nur 4 also ist klar, dass es
150 // sich bei "4" um alle handelt.
151 // Könnte man "all" kodieren, könnte man die "four" auskommentieren
152 [ [ #If #a player|AG fills|ACT #all *four goal_fields|PAT ]|SUM
153   he|AG wins|ACT ] .
154 [ @player|EQK @he|EQD ] .
155 [ @player|PAR @players|OMN ] .
```

E. Listings

E.1. Vorbereitungsregeln

Listing E.1: Vorbereitung: Berechnen der transitiven Hülle.

```
rule createTransitiveClosureForEqualNodes {
  a:PHRASE-e:OBJECTROLE->x:CONSTITUENT;
  y:CONSTITUENT;

  if {
    x.VISITED == true;
    x.VALUE == y.VALUE;
  }

  negative {x-alreadyCreated:EQUALNODES->y;}

  modify {
    x-:EQUALNODES->y;
    emit("<!--Inserting edge for transitive closure between "
        + x.NAME + " and " + y.NAME + "-->\n");
  }
}
```

Listing E.2: Vorbereitung: Verschmelzen von Knoten.

```
/* Einstiegsregel */
rule mergeEqualNodes {
    modify {
        exec(mergeEqualNodes_keep[*]);
        exec(mergeEqualNodes_redirectNext[*]);
        exec(mergeEqualNodes_redirectNextLast[*]);
        exec(mergeEqualNodes_drop[*]);
    }
}

/* Merge nodeB into nodeA
 * if [ nodeA|EQK nodeB|EQD ].
 *
 * Remove connections to nodeB and recreate them pointing to nodeA */
rule mergeEqualNodes_keep {
    connectingPhrase:PHRASE-eq_1:EQK->nodeA:CONSTITUENT;
    connectingPhrase-eq_2:EQD->nodeB:CONSTITUENT;

    otherPhrase:PHRASE-originalEdge:ROLE->nodeB;

    modify {
        otherPhrase-newEdge:typeof(originalEdge)->nodeA;
        delete(originalEdge);
        emit("<!-- mergeEqualNodes: Replacing " + nodeB.NAME
           + " with " + nodeA.NAME + " -->\n");
    }
```

```
27 }
28
29 rule mergeEqualNodes_redirectNext {
30     connectingPhrase:PHRASE-eq_keep:EQK->nodeKeep:CONSTITUENT;
31     connectingPhrase-eq_drop:EQD->nodeDrop:CONSTITUENT;
32     pred:PHRASE-e_next_1:NEXT->connectingPhrase;
33     connectingPhrase-e_next_2:NEXT->succ:PHRASE;
34
35     modify {
36       delete(e_next_1);
37       delete(e_next_2);
38
39       pred-:NEXT->succ;
40     }
41 }
42
43 rule mergeEqualNodes_redirectNextLast {
44     connectingPhrase:PHRASE-eq_keep:EQK->nodeKeep:CONSTITUENT;
45     connectingPhrase-eq_drop:EQD->nodeDrop:CONSTITUENT;
46     pred:PHRASE-e_next_1:NEXT->connectingPhrase;
47
48     negative {
49         connectingPhraseNAC:PHRASE-eq_keepNAC:EQK->nodeKeepNAC:CONSTITUENT;
50         connectingPhraseNAC-eq_dropNAC:EQD->nodeDropNAC:CONSTITUENT;
51         predNAC:PHRASE-e_next_1NAC:NEXT->connectingPhraseNAC;
52         /* nur, wenn kein succ vorhanden! */
53         connectingPhraseNAC-e_next_2NAC:NEXT->succNAC:PHRASE;
54     }
55
56     modify {
57       delete(e_next_1);
58     }
59 }
60
61 /* Merge nodeB into nodeA
62  * if [ nodeA@EQK nodeB|EQD ].
63  *
64  * Remove connection Phrase if and only if there is no other edge to be redirected
65  */
66 rule mergeEqualNodes_drop {
67     connectingPhrase:PHRASE-eq_keep:EQK->nodeKeep:CONSTITUENT;
68     connectingPhrase-eq_drop:EQD->nodeDrop:CONSTITUENT;
69
70     negative {
71         connectingPhraseNAC:PHRASE-eq_keepNAC:EQK->nodeKeepNAC:CONSTITUENT;
72         connectingPhraseNAC-eq_dropNAC:EQD->nodeDropNAC:CONSTITUENT;
73
74         /* no other edge pointing to nodeDrop */
75         -e:Edge->nodeDropNAC;
76     }
77
78     negative {
79         connectingPhraseNAC:PHRASE-eq_keepNAC:EQK->nodeKeepNAC:CONSTITUENT;
80         connectingPhraseNAC-eq_dropNAC:EQD->nodeDropNAC:CONSTITUENT;
81
82         /* no other edge pointing to connectingPhrase (next-edge!) */
83         -en:Edge->connectingPhraseNAC;
84     }
85
86     modify {
87         delete(eq_drop);
88         delete(eq_keep);
89         delete(connectingPhrase);
```

```
90          delete(nodeDrop);
91          emit("<!-- mergeEqualNodes: Removing helper-phrase ... -->\n");
92      }
93  }
94
95  /* Einstiegsregel */
96  rule mergeEqualSets {
97      modify {
98          exec(mergeEqualSetNodes_createSet[*]);
99          exec(mergeEqualSetNodes_addToSet[*]);
100         exec(mergeEqualSetNodes_removeSetIdentifier[*]);
101     }
102 }
103
104 rule mergeEqualSetNodes_removeSetIdentifier {
105     a:PHRASE-rolle:ROLE->set:AND_SET;
106     a-rolleDel:typeof(rolle)->identifier:WORD;
107     identifier-rolleDel2:SET_CLOSURE->set;
108
109     modify {
110         delete(rolleDel);
111         delete(rolleDel2);
112         delete(identifier);
113     }
114 }
115
116 rule mergeEqualSetNodes_createSet {
117     connectingPhrase:PHRASE-eMenge:EQB->menge:WORD;
118     connectingPhrase-eTeil:EQS->teil:WORD;
119
120     usingPhrase:PHRASE-rolle:ROLE->menge;
121
122     negative {
123         connectingPhraseNAC:PHRASE-eMengeNAC:EQB->mengeNAC:WORD;
124         connectingPhraseNAC-eTeilNAC:EQS->teilNAC:WORD;
125         mengeNAC-:SET_CLOSURE->setNAC:AND_SET;
126     }
127
128     modify {
129         set:AND_SET;
130         eval { set.NAME = "AND-SET " + menge.NAME + ":"; }
131         menge-:SET_CLOSURE->set;
132         usingPhrase-:typeof(rolle)->set;
133     }
134 }
135
136 rule mergeEqualSetNodes_addToSet {
137     connectingPhrase:PHRASE-eMenge:EQB->mengeBezeichnung:WORD;
138     connectingPhrase-eTeil:EQS->teil:WORD;
139     usingPhrase:PHRASE-rolle:ROLE->mengeBezeichnung;
140     mengeBezeichnung-:SET_CLOSURE->set:AND_SET;
141     usingPhrase-rolle2:typeof(rolle)->set;
142
143     negative {
144         connectingPhraseNAC:PHRASE-eMengeNAC:EQB->mengeBezeichnungNAC:WORD;
145         connectingPhraseNAC-eTeilNAC:EQS->teilNAC:WORD;
146         usingPhraseNAC:PHRASE-rolleNAC:ROLE->mengeBezeichnungNAC;
147         mengeBezeichnungNAC-:SET_CLOSURE->setNAC:AND_SET;
148         teilNAC-:SET_CLOSURE->setNAC;
149     }
150
151     modify {
152         teil-:SET_CLOSURE->set;
```

```
153         set-:typeof(rolle)->teil;
154
155         eval { set.NAME = set.NAME + ", " + teil.NAME;}
156
157         exec(removeEqSet1(connectingPhrase)
158             || removeEqSet2(connectingPhrase));
159     }
160
161 }
162
163 rule removeEqSet1(a:PHRASE) {
164     pred:PHRASE-:NEXT->a;
165     a-:NEXT->succ:PHRASE;
166     a-e1:EQS->:WORD;
167     a-e2:EQB->:WORD;
168
169     modify {
170         delete(a);
171         delete(e1);
172         delete(e2);
173         pred-:NEXT->succ;
174     }
175 }
176
177 rule removeEqSet2(a:PHRASE) {
178     pred:PHRASE-:NEXT->a;
179     a-e1:EQS->:WORD;
180     a-e2:EQB->:WORD;
181
182     negative {
183         predNAC:PHRASE-:NEXT->a;
184         a-:NEXT->succNAC:PHRASE;
185     }
186
187     modify {
188         delete(a);
189         delete(e1);
190         delete(e2);
191     }
192 }
```

Listing E.3: prepareMultiplicities.grg

```
1  rule prepareMultiplicities {
2      modify {
3          exec(replaceMultOne); exec(replaceMultTwo);
4          /* exec(replaceMultThree); exec(replaceMultFour); exec(replaceMultFive);
5           * exec(replaceMultSix); exec(replaceMultSeven); exec(replaceMultEight); */
6
7          exec(replaceMultAny);
8      }
9  }
10
11 // Dont look below this line ;)
12 rule replaceMultAny {
13     modify { exec(replaceMultAny_1 [*]); exec(replaceMultAny_2 [*]); }
14 }
15
16 rule replaceMultAny_1{
17     :CONSTITUENT-:HAS_ATTRIBUTE->a:ATTRIBUTE;
18     if { a.NAME=="MULTIPLICITY"; }
19     if { a.VALUE=="any"; }
20
```

```
21      modify{ eval{ a.VALUE="0..*";} }
22  }
23
24  rule replaceMultAny_2{
25      :CONSTITUENT-:HAS_ATTRIBUTE->a:ATTRIBUTE;
26      if { a.NAME=="MULTIPLICITY"; }
27      if { a.VALUE=="Any"; }
28
29      modify{ eval{ a.VALUE="0..*";} }
30  }
31
32  rule replaceMultOne {
33      modify {
34          exec(replaceMultOne_1 [*]); exec(replaceMultOne_2 [*]);
35
36          /* An/an: exec(replaceMultOne_3 [*]); exec(replaceMultOne_4 [*]);
37           * A/a:   exec(replaceMultOne_5 [*]); exec(replaceMultOne_6 [*]); */
38      }
39  }
40
41  rule replaceMultTwo {
42      modify { exec(replaceMultTwo_1 [*]); exec(replaceMultTwo_2 [*]); }
43  }
44
45  rule replaceMultOne_1{
46      :CONSTITUENT-:HAS_ATTRIBUTE->a:ATTRIBUTE;
47
48      if { a.NAME=="MULTIPLICITY"; }
49      if { a.VALUE=="one"; }
50
51      modify { eval { a.VALUE="1"; } }
52  }
53
54  rule replaceMultOne_2{
55      :CONSTITUENT-:HAS_ATTRIBUTE->a:ATTRIBUTE;
56
57      if { a.NAME=="MULTIPLICITY"; }
58      if { a.VALUE=="One"; }
59
60      modify { eval { a.VALUE="1"; } }
61  }
62
63  rule replaceMultTwo_1{
64      :CONSTITUENT-:HAS_ATTRIBUTE->a:ATTRIBUTE;
65
66      if { a.NAME=="MULTIPLICITY"; }
67      if { a.VALUE=="two"; }
68
69      modify { eval { a.VALUE="2"; } }
70  }
71
72  rule replaceMultTwo_2{
73      :CONSTITUENT-:HAS_ATTRIBUTE->a:ATTRIBUTE;
74      if { a.NAME=="MULTIPLICITY"; }
75      if { a.VALUE=="Two"; }
76
77      modify { eval { a.VALUE="2"; } }
78  }
```

E.2. Ausgaberegeln

Listing E.4: emitQuestionnaire.grg

```
rule emitQuestionnaire {
  modify {
    emit("<?xml version=\"1.0\"?>\n");
    emit("<?xml-stylesheet type=\"text/xsl\" href=\"Questionnaire.xsl\"?>\n");
    emit("<phrases>\n");
    exec ( emitNextSatz[*] );
    emit("</phrases>\n");
  }
}
rule emitNextSatz {
  act:CONSTITUENT-e_next:NEXT->next:PHRASE;
  if {
    act.VISITED == true;
    e_next.VISITED == false;
    next.VISITED == false;
  }
  modify {
    eval{ e_next.VISITED = true; }
    exec(emitQuestions(next));
    exec(emitSubclauses(next)[*]);
  }
}

/* Create questions for a single phrase:*/
rule emitQuestions(a:PHRASE) {
  if { a.VISITED==false; }
  modify {
    eval {a.VISITED=true;}
    emit ("<phrase value=\"" + a.VALUE + "\">\n");
    exec ( emitAll(a));
    emit ("</phrase>\n");
  }
}
```

Listing E.5: emitAll.grg

```
/* Calls all emitting rules for a given phrase --> emit_*(a) */
rule emitAll(a:PHRASE) {
  modify {
    exec(emitObjectsAndAttributes(a)[*] | emitObjectsAndAttributesInSets(a)[*]
       | emitACT1(a)[*] | emitACT2(a)[*] | emitACT3(a)[*] | emitACT4(a)[*]
       | emitCAU(a)[*]
       | emitCOMPCOMPII(a)[*]
       | emitCONTCONTII(a)[*]
       | emitCREAOPUS(a)[*]
       | emitDONRECPHAB(a)[*] | emitDONHAB(a)[*] | emitDONRECP(a)[*]
       | emitDON(a)[*] | emitRECP(a)[*]
       | emitDUXCOM(a)[*]
       | emitEXPNOTSTIM(a)[*] | emitEXPSTIM(a)[*] | emitEXPNOT(a)[*] | emitEXP(a)[*]
       | emitNOT(a)[*] | emitSTIM(a)[*]
       | emitFAVFAUBEN(a)[*] | emitFAVBEN(a)[*] | emitFAVFAU(a)[*] | emitFAV(a)[*]
       | emitFAU(a)[*] | emitBEN(a)[*]
       | emitFINFIC(a)[*]
       | emitFREQ(a)[*]
       | emitINSTROLE1(a)[*] | emitINSTROLE2(a)[*]
       | emitMOD1(a)[*] | emitMOD2(a)[*]
       | emitOMNPAR_ACT_more(a)[*] | emitOMNPARS_ACT_single(a)[*]
```

```
      | emitOMNPARS_CLASS_more(a)[*] | emitOMNPARS_CLASS_single(a)[*]
      | emitPOSSHAB(a)[*]
      | emitPRESUCC(a)[*]
      | emitQUALQUALII(a)[*]
      | emitSTATSTATII(a)[*]
      | emitSUBSUBSII(a)[*]
      | emitSUM(a)[*]
      | emitTHETHEII(a)[*]
      | emitTEMPLOC(a)[*]
      );
  }
}
```

Listing E.6: emitAct.grg

```
rule emitACT1(a:PHRASE) {
  a-e_ag:AG->nodeAg:CONSTITUENT;
  a-e_p:PAT->nodePat:CONSTITUENT;
  a-e_ac:ACT->nodeAct:CONSTITUENT;

  if {
    e_ag.VISITED == false;
    e_p.VISITED == false;
    nodeAct.VISITED == false;
  }
  modify {
    eval{e_ag.VISITED = true;}
    eval{e_p.VISITED = true;}
    eval{nodeAct.VISITED = true;}
    emit("<element>\n");
    emit("  <question>&quot;" + nodeAct.VALUE + "&quot; modeled...</question>\n");
    emit("  <option>as method of &quot;" + nodeAg.VALUE + "&quot;</option>\n");
    emit("  <option>as method of &quot;" + nodePat.VALUE + "&quot;</option>\n");
    emit("  <option>in a state chart (as state or as transition)</option>\n");
    emit("  <option>as relation between &quot;" + nodeAg.VALUE + "&quot; and &quot;"
          + nodePat.VALUE + "&quot;</option>\n");
    emit("</element>\n");
    exec(emitATTRIBUTES(nodeAct));
  }
}

rule emitACT2(a:PHRASE) {
  a-e_ag:AG->nodeAg:CONSTITUENT;
  a-e_ac:ACT->nodeAct:CONSTITUENT;

  negative {
    nodeAg;
    nodeAct;
    a-e_p:PAT->nodePat:CONSTITUENT;
  }

  if {
    e_ag.VISITED == false;
    nodeAct.VISITED == false;
  }

  modify {
    eval{e_ag.VISITED = true;}
    eval{nodeAct.VISITED = true;}
    emit("<element>\n");
    emit("  <question>&quot;" + nodeAct.VALUE + "&quot; modeled...</question>\n");
    emit("  <option>as method of &quot;" + nodeAg.VALUE + "&quot;</option>\n");
    emit("  <option>in a state chart (as state or as transition)</option>\n");
```

```
49      emit("</element>\n");
50      exec(emitATTRIBUTES(nodeAct));
51    }
52
53  }
54  rule emitACT3(a:PHRASE) {
55    a-e_p:PAT->nodePat:CONSTITUENT;
56    a-e_ac:ACT->nodeAct:CONSTITUENT;
57
58    negative {
59      nodeAct;
60      nodePat;
61      a-e_a:AG->nodeAg:CONSTITUENT;
62    }
63
64    if {
65      nodeAct.VISITED == false;
66      e_p.VISITED == false;
67    }
68    modify {
69      eval{e_p.VISITED = true;}
70      eval{e_ac.VISITED = true;}
71      emit("<element>\n");
72      emit("  <question>&quot;" + nodeAct.VALUE + "&quot; modeled...</question>\n");
73      emit("  <option>as method of &quot;" + nodePat.VALUE + "&quot;</option>\n");
74      emit("  <option>in a state chart (as state or as transition)</option>\n");
75      emit("</element>\n");
76      exec(emitATTRIBUTES(nodeAct));
77    }
78
79  }
80  rule emitACT4(a:PHRASE) {
81    a-e_ac:ACT->nodeAct:CONSTITUENT;
82
83    negative {
84      nodeAct;
85      a-e_a:AG->nodeAg:CONSTITUENT;
86    }
87
88    negative {
89      nodeAct;
90      a-e_p:PAT->nodePat:CONSTITUENT;
91    }
92
93    if {
94      nodeAct.VISITED == false;
95    }
96    modify {
97      eval{nodeAct.VISITED = true;}
98      emit("<element>\n");
99      emit("  <question>&quot;" + nodeAct.VALUE + "&quot; modeled...</question>\n");
100     emit("  <option>in a state chart (as state or as transition)</option>\n");
101     emit("</element>\n");
102     exec(emitATTRIBUTES(nodeAct));
103   }
104 }
```

Listing E.7: emitAttributesAndMultiplicities.grg

```
1   rule emitATTRIBUTES(nodeA:CONSTITUENT) {
2     modify {
3       emit("<!-- emitAttributes(nodeA) -->\n");
4       exec(emitMULTIPLICITY(nodeA)[*]);
```

```
5     exec(emitATTRIBUTE_ObjectType(nodeA)[*]);
6     exec(emitATTRIBUTE_MethodType(nodeA)[*]);
7     emit("<!-- End of emitAttributes(nodeA) -->\n");
8   }
9 }
10
11 rule emitMULTIPLICITY(nodeA:CONSTITUENT) {
12   nodeA-e:HAS_ATTRIBUTE->attribute:ATTRIBUTE;
13
14   if {
15     e.VISITED == false;
16     attribute.NAME == "MULTIPLICITY";
17   }
18
19   modify {
20     eval{ e.VISITED = true; }
21     emit("<!-- emitMULTIPLICITY(a) -->\n");
22     emit("<element>\n");
23     emit("  <question>Multiplicity &quot;" + attribute.VALUE + "&quot; of &quot;"
24           + nodeA.NAME + "&quot;modeled...</question>\n");
25     emit("  <option>as multiplicity</option>\n");
26     emit("  <option>in an OCL constraint</option>\n");
27     emit("</element>\n");
28     emit("<!-- End of emitMULTIPLICITY(a) -->\n\n");
29   }
30 }
31
32 rule emitATTRIBUTE_ObjectType(nodeA:CONSTITUENT) {
33   nodeA-e:HAS_ATTRIBUTE->attribute:ATTRIBUTE;
34   if {
35     e.VISITED == false;
36     attribute.NAME != "MULTIPLICITY";
37   }
38   modify {
39     eval{ e.VISITED = true; }
40     emit("<!-- emitATTRIBUTE_ObjectType(a) -->\n");
41     emit("<element>\n");
42     emit("  <question>Attribute &quot;" + attribute.NAME + "&quot; of &quot;"
43           + nodeA.NAME + "&quot;modeled...</question>\n");
44     emit("  <option>as Boolean function (with a parameter)</option>\n");
45     emit("  <option>as scalar function</option>\n");
46     emit("  <option>as state of &quot;" + nodeA.NAME + "&quot;</option>\n");
47     emit("  <option>as attribute of &quot;" + nodeA.NAME + "&quot;</option>\n");
48     emit("</element>\n");
49     emit("<!-- End of emitATTRIBUTE_ObjectType(a) -->\n\n");
50   }
51 }
52
53 rule emitATTRIBUTE_MethodType(nodeA:CONSTITUENT) {
54   nodeA-e:HAS_ATTRIBUTE->attribute:ATTRIBUTE;
55   if {
56     e.VISITED == false;
57     attribute.NAME != "MULTIPLICITY";
58   }
59   modify {
60     eval{ e.VISITED = true; }
61     emit("<!-- emitATTRIBUTE_MethodType(a) -->\n");
62     emit("<element>\n");
63     emit("  <question>Attribute &quot;" + attribute.NAME + "&quot; of &quot;"
64           + nodeA.NAME + "&quot;modeled...</question>\n");
65     emit("  <option>as parameter of the function &quot;" + nodeA.NAME + "&quot;</option>\n");
66     emit("  <option>as part of the name of &quot;" + nodeA.NAME + "&quot;</option>\n");
67     emit("</element>\n");
```

```
68      emit("<!-- end of emitATTRIBUTE_MethodType(a) -->\n\n");
69    }
70  }
```

Listing E.8: emitCau.grg

```
1   rule emitCAU (a:PHRASE) {
2     a-e_1:CAUROLE->:CONSTITUENT;
3
4     if {
5       e_1.VISITED == false;
6     }
7
8     modify{
9       eval{
10        e_1.VISITED = true;
11      }
12      emit("<!-- emitCAU(a) -->\n");
13      emit("<element>\n");
14      emit("  <note>... CAU should not be modeled</note>\n");
15      emit("</element>\n");
16      emit("<!-- end of emitCAU(a) -->\n");
17    }
18  }
```

Listing E.9: emitCompCompii.grg

```
1   rule emitCOMPCOMPII (a:PHRASE) {
2     a-e_1:COMP->comparius:CONSTITUENT;
3     a-e_2:COMPII->compariens:CONSTITUENT;
4
5     if {
6       e_1.VISITED == false;
7       e_2.VISITED == false;
8     }
9
10    modify{
11      eval{
12        e_1.VISITED = true;
13        e_2.VISITED = true;
14      }
15      emit("<!-- emitCOMPCOMPII(a) -->\n");
16      emit("<element>\n");
17      emit("  <question>&quot;" + compariens.VALUE + "&quot; and &quot;"
18            + comparius.VALUE + "&quot; can be compared...</question>\n");
19      emit("  <option>using a (boolean) method</option>\n");
20      emit("</element>\n");
21      emit("<!-- end of emitCOMPCOMPII(a) -->\n");
22    }
23  }
```

Listing E.10: emitContContii.grg

```
1   rule emitCONTCONTII (a:PHRASE) {
2     a-e_1:CONT->nodeA:CONSTITUENT;
3     a-e_2:CONTII->nodeB:CONSTITUENT;
4
5     if {
6       e_1.VISITED == false;
7       e_2.VISITED == false;
8     }
```

```
9
10  modify{
11    eval{
12      e_1.VISITED = true;
13      e_2.VISITED = true;
14    }
15    emit("<!-- emitCONTCONTII(a) -->\n");
16    emit("<element>\n");
17    emit("  <question>&quot;" + nodeA.VALUE + "&quot; and &quot;"
18          + nodeB.VALUE + "&quot; modeled...</question>\n");
19    emit("  <option>as two instances of an (unknown) class</option>\n");
20    emit("  <option>as two subclasses of an (unknown) class</option>\n");
21    emit("</element>\n");
22    emit("<!-- end of emitCONTCONTII(a) -->");
23  }
24 }
```

Listing E.11: emitCreaOpus.grg

```
1  rule emitCREAOPUS (a:PHRASE) {
2    a-e_2:CREAROLE->node_crea:CONSTITUENT;
3    a-e_3:OPUSROLE->node_opus:CONSTITUENT;
4
5    if {
6      e_2.VISITED == false;
7      e_3.VISITED == false;
8    }
9
10   modify{
11     eval{
12       e_2.VISITED = true;
13       e_3.VISITED = true;
14     }
15     emit("<!-- emitCREAOPUS(a) -->\n");
16     emit("<element>\n");
17     emit("  <question>&quot;" + node_opus.VALUE + "&quot; and &quot;" + node_crea.VALUE
18           + "&quot; modeled...</question>\n");
19     emit("  <option>as method with return value &quot;" + node_opus.VALUE
20           + "&quot; of &quot;" +  node_crea.VALUE + "&quot;</option>\n");
21     emit("  <option>as design pattern (Factory, Creator, Lightweight...) to create "
22           + "&quot;" +  node_opus.VALUE + "&quot; with &quot;" + node_crea.VALUE
23           + "&quot;</option>\n");
24     emit("  <option>in a state chart (&quot;" + node_crea.VALUE + "&quot;, &quot;"
25           + node_opus.VALUE + "&quot;)</option>\n");
26     emit("</element>\n");
27   }
28 }
```

Listing E.12: emitDonRecpHab.grg

```
1  rule emitDONRECPHAB (a:PHRASE) {
2    a-e_1:DON->node_don:CONSTITUENT;
3    a-e_2:RECP->node_recp:CONSTITUENT;
4    a-e_3:HAB->node_hab:CONSTITUENT;
5
6    if {
7      e_1.VISITED == false;
8      e_2.VISITED == false;
9    }
10
11   modify{
12     eval{
```

```
13        e_1.VISITED = true;
14        e_2.VISITED = true;
15        e_3.VISITED = true;
16      }
17      emit("<!-- emitDONRECPHAB(a) -->\n");
18      emit("<element>\n");
19      emit("  <question>&quot;" + node_don.VALUE + "&quot;, &quot;" + node_recp.VALUE
20            + "&quot; and &quot;" + node_hab.VALUE + "&quot; modeled...</question>\n");
21      emit("  <option>as method with according parameters for &quot;" + node_don.VALUE
22            + "&quot;, &quot;" + node_recp.VALUE + "&quot; and &quot;" + node_hab.VALUE
23            + "&quot;(communication possible?)</option>\n");
24      emit("  <option>as classes with associations</option>\n");
25      emit("</element>\n");
26    }
27  }
28
29  rule emitDONHAB (a:PHRASE) {
30    a-e_1:DON->node_don:CONSTITUENT;
31    a-e_3:HAB->node_hab:CONSTITUENT;
32
33    negative {
34      a-e_1->node_don;
35      a-e_2:RECP->node_recp:CONSTITUENT;
36      a-e_3->node_hab;
37    }
38
39    if { e_1.VISITED == false; }
40
41    modify{
42      eval{
43        e_1.VISITED = true;
44        e_3.VISITED = true;
45      }
46      emit("<!-- emitDONHAB(a) -->\n");
47      emit("<element>\n");
48      emit("  <question>&quot;" + node_don.VALUE + "&quot; and &quot;" + node_hab.VALUE
49            + "&quot; modeled...</question>\n");
50      emit("  <note>recipiens is missing</note>\n");
51      emit("  <option>as method with according parameters for &quot;" + node_don.VALUE + "&quot; and &quot;"
52            + node_hab.VALUE + "&quot; (communication possible?)</option>\n");
53      emit("  <option>as classes with associations</option>\n");
54      emit("</element>\n");
55    }
56  }
57
58  rule emitDONRECP (a:PHRASE) {
59    a-e_1:DON->node_don:CONSTITUENT;
60    a-e_2:RECP->node_recp:CONSTITUENT;
61
62    negative {
63      a-e_1->node_don;
64      a-e_2->node_recp;
65      a-e_3:HAB->node_hab:CONSTITUENT;
66    }
67
68    if {
69      e_1.VISITED == false;
70      e_2.VISITED == false;
71    }
72
73    modify{
74      eval{
75        e_1.VISITED = true;
```

```
76        e_2.VISITED = true;
77      }
78      emit("<!-- emitDONRECP(a) -->\n");
79      emit("<element>\n");
80      emit("  <question>&quot;" + node_don.VALUE + "&quot; and &quot;" + node_recp.VALUE
81            + "&quot; modeled...</question>\n");
82      emit("  <note>habitum is missing</note>\n");
83      emit("  <option>as method with according parameters for &quot;" + node_don.VALUE + "&quot; and &quot;"
84            + node_recp.VALUE + "&quot; (communication possible?)</option>\n");
85      emit("  <option>as classes with associations</option>\n");
86      emit("</element>\n");
87    }
88  }
89
90  rule emitDON (a:PHRASE) {
91    a-e_1:DON->node_don:CONSTITUENT;
92
93    negative {
94      a-e_1->node_don;
95      a-e_2:RECP->node_recp:CONSTITUENT;
96      a-e_3:HAB->node_hab:CONSTITUENT;
97    }
98
99    if { e_1.VISITED == false; }
100
101   modify{
102     eval{ e_1.VISITED = true; }
103     emit("<!-- emitDON(a) -->\n");
104     emit("<element>\n");
105     emit("  <question>&quot;" + node_don.VALUE + "&quot; modeled...</question>\n");
106     emit("  <note>habitum is missing</note>\n");
107     emit("  <note>recipiens is missing</note>\n");
108     emit("  <option>as method with according parameters for &quot;" + node_don.VALUE
109           + "&quot; (communication possible?)</option>\n");
110     emit("</element>\n");
111   }
112 }
113
114 rule emitRECP (a:PHRASE) {
115   a-e_2:RECP->node_recp:CONSTITUENT;
116
117   negative {
118     a-e_1:DON->node_don:CONSTITUENT;
119     a-e_2->node_recp;
120     a-e_3:HAB->node_hab:CONSTITUENT;
121   }
122
123   if { e_2.VISITED == false; }
124
125   modify{
126     eval{ e_2.VISITED = true; }
127     emit("<!-- emitRECP(a) -->\n");
128     emit("<element>\n");
129     emit("  <question>&quot;" + node_recp.VALUE + "&quot; modeled...</question>\n");
130     emit("  <note>habitum is missing</note>\n");
131     emit("  <note>donor is missing</note>\n");
132     emit("  <option>as method with according parameters for &quot;" + node_recp.VALUE
133          + "&quot; (communication possible?)</option>\n");
134     emit("</element>\n");
135   }
136 }
```

Listing E.13: emitDuxCom.grg

```
rule emitDUXCOM(a:PHRASE) {
  a-e:DUX->nodeDux:CONSTITUENT;
  a-e2:ACT->nodeAct:CONSTITUENT;
  a-e3:COM->nodeCom:CONSTITUENT;
  if {
    e.VISITED == false;
    e3.VISITED == false;
  }
  modify {
        eval {
          e.VISITED = true;
          e2.VISITED = true;
          e3.VISITED = true;
        }
    emit("<element>\n");
    emit("  <question>&quot;" + nodeAct.VALUE + "&quot; modeled with parameter &quot;" + nodeCom.VALUE
          + "&quot;...</question>\n");
    emit("  <option>as method void &quot;" + nodeDux.VALUE + "::" + nodeAct.VALUE  + "("  + nodeCom.VALUE
          + ")&quot;</option>\n");
    emit("</element>\n");
  }
}
```

Listing E.14: emitExpNotStim.grg

```
rule emitEXPNOTSTIM (a:PHRASE) {
  a-e_1:EXP->node_exp:CONSTITUENT;
  a-e_2:NOT->node_not:CONSTITUENT;
  a-e_3:STIM->node_stim:CONSTITUENT;

  if {
    e_1.VISITED == false;
    e_2.VISITED == false;
    e_3.VISITED == false;
  }

  modify{
    eval{
      e_1.VISITED = true;
      e_2.VISITED = true;
      e_3.VISITED = true;
    }
    emit("<!-- emitEXPNOTSTIM(a) -->\n");
    emit("<element>\n");
    emit("<question>&quot;" + node_not.VALUE + "&quot;, &quot;" + node_stim.VALUE + "&quot; and &quot;"
        + node_exp.VALUE + "&quot; modeled...</question>\n");
    emit("  <option>as method of with according parameters (communication possible?)</option>\n");
    emit("  <option>in a state chart with a signal for &quot;" + node_stim.VALUE + "&quot;</option>\n");
    emit("  <option>as classes with associations</option>\n");
    emit("</element>\n");
  }
}

rule emitEXPSTIM (a:PHRASE) {
  a-e_1:EXP->node_exp:CONSTITUENT;
  a-e_3:STIM->node_stim:CONSTITUENT;

  negative {
    a-e_1->node_exp;
    a-e_2:NOT->node_not:CONSTITUENT;
    a-e_3->node_stim;
```

```
37    }
38
39    if {
40      e_1.VISITED == false;
41      e_3.VISITED == false;
42    }
43
44    modify{
45      eval{
46        e_1.VISITED = true;
47        e_3.VISITED = true;
48      }
49      emit("<!-- emitEXPSTIM(a) -->\n");
50      emit("<element>\n");
51      emit("  <note>notio is missing</note>\n");
52      emit("  <question>&quot;" + node_exp.VALUE + "&quot; and &quot;" + node_stim.VALUE
53            + "&quot; modeled...</question>\n");
54      emit("  <option>as method with parameter &quot;" + node_stim.VALUE
55            + "&quot; (communication possible?)</option>\n");
56      emit("  <option>in a state chart with signal from &quot;" + node_stim.VALUE + "&quot; to &quot;"
57            + node_exp.VALUE + "&quot;</option>\n");
58      emit("  <option>as classes with associations</option>\n");
59      emit("</element>\n");
60    }
61  }
62
63  rule emitEXPNOT (a:PHRASE) {
64    a-e_1:EXP->node_exp:CONSTITUENT;
65    a-e_2:NOT->node_not:CONSTITUENT;
66
67    negative {
68      a-e_1->node_exp;
69      a-e_2->node_not;
70      a-e_3:STIM->node_stim:CONSTITUENT;
71    }
72
73    if {
74      e_1.VISITED == false;
75      e_2.VISITED == false;
76    }
77
78    modify{
79      eval{
80        e_1.VISITED = true;
81        e_2.VISITED = true;
82      }
83      emit("<!-- emitEXPNOT(a) -->\n");
84      emit("<element>\n");
85      emit("  <note>stimulus is missing</note>\n");
86      emit("  <question>&quot;" + node_exp.VALUE + "&quot; and &quot;" + node_not.VALUE
87            + "&quot; modeled...</question>\n");
88      emit("  <option>as method with according parameter (communication possible?)</option>\n");
89      emit("  <option>in a state chart with signal &quot;" + node_not.VALUE + "&quot; to &quot;"
90            + node_exp.VALUE + "&quot;</option>\n");
91      emit("  <option>as classes with associations</option>\n");
92      emit("</element>\n");
93    }
94  }
95
96
97  rule emitEXP (a:PHRASE) {
98    a-e_1:EXP->node_exp:CONSTITUENT;
99
```

```
100   negative {
101     a-e_1->node_exp;
102     a-e_3:STIM->node_stim:CONSTITUENT;
103   }
104   negative {
105     a-e_1->node_exp;
106     a-e_2:NOT->node_not:CONSTITUENT;
107   }
108
109   if { e_1.VISITED == false; }
110
111   modify{
112     eval{ e_1.VISITED = true; }
113     emit("<!-- emitEXP(a) -->\n");
114     emit("<element>\n");
115     emit("  <question>&quot;" + node_exp.VALUE + "&quot; modeled...</question>\n");
116     emit("  <note>stimulus is missing</note>\n");
117     emit("  <note>notio is missing</note>\n");
118     emit("  <option>as method of &quot;" + node_exp.VALUE
119          + "&quot;with according parameters (communication possible?)</option>\n");
120     emit("  <option>in a state chart with signal to &quot;" + node_exp.VALUE + "&quot;</option>\n");
121     emit("</element>\n");
122   }
123 }
124
125 rule emitNOT (a:PHRASE) {
126   a-e_2:NOT->node_not:CONSTITUENT;
127
128   negative {
129     a-e_1:EXP->node_exp:CONSTITUENT;
130     a-e_2->node_not;
131   }
132
133   negative {
134     a-e_2->node_not;
135     a-e_3:STIM->node_stim:CONSTITUENT;
136   }
137
138   if { e_2.VISITED == false; }
139
140   modify{
141     eval{ e_2.VISITED = true; }
142     emit("<!-- emitNOT(a) -->\n");
143     emit("<element>\n");
144     emit("  <question>&quot;" + node_not.VALUE + "&quot; modeled...</question>\n");
145     emit("  <note>stimulus is missing</note>\n");
146     emit("  <note>experior is missing</note>\n");
147     emit("  <option>as method with according parameter &quot;" + node_not.VALUE
148          + "&quot; (communication possible?)</option>\n");
149     emit("  <option>in a state chart with signal &quot;" + node_not.VALUE + "&quot;</option>\n");
150     emit("</element>\n");
151   }
152 }
153
154 rule emitSTIM (a:PHRASE) {
155   a-e_3:STIM->node_stim:CONSTITUENT;
156
157   negative {
158     a-e_1:EXP->node_exp:CONSTITUENT;
159     a-e_3->node_stim;
160   }
161   negative {
162     a-e_2:NOT->node_not:CONSTITUENT;
```

```
163     a-e_3->node_stim;
164   }
165
166   if { e_3.VISITED == false; }
167
168   modify{
169     eval{ e_3.VISITED = true; }
170     emit("<!-- emitSTIM(a) -->\n");
171     emit("<element>\n");
172     emit("  <question>&quot;" + node_stim.VALUE + "&quot; modeled...</question>\n");
173     emit("  <note>notio is missing</note>\n");
174     emit("  <note>experior is missing</note>\n");
175     emit("  <option>as method with according parameters called from &quot;" + node_stim.VALUE
176           + "&quot; (communication possible?)</option>\n");
177     emit("  <option>in a state chart with signal &quot;" + node_stim.VALUE + "&quot;</option>\n");
178     emit("</element>\n");
179   }
180 }
```

Listing E.15: emitFavFauBen.grg

```
1   rule emitFAVFAUBEN (a:PHRASE) {
2     a-e_1:FAVROLE->node_fav:CONSTITUENT;
3     a-e_2:FAUROLE->node_fau:CONSTITUENT;
4     a-e_3:BENROLE->node_ben:CONSTITUENT;
5
6     if {
7       e_1.VISITED == false;
8       e_2.VISITED == false;
9       e_3.VISITED == false;
10    }
11
12    modify{
13      eval{
14        e_1.VISITED = true;
15        e_2.VISITED = true;
16        e_3.VISITED = true;
17      }
18      emit("<!-- emitFAVFAUBEN(a) -->\n");
19      emit("<element>\n");
20      emit("  <question>&quot;" + node_fav.VALUE + "&quot;, &quot;" + node_fau.VALUE + "&quot; and &quot;"
21            + node_ben.VALUE + "&quot;modeled...</question>\n");
22      emit("  <option>as method with according parameters (communication possible?)</option>\n");
23      emit("  <option>as classes with associations</option>\n");
24      emit("</element>\n");
25    }
26  }
27
28  rule emitFAVBEN (a:PHRASE) {
29    a-e_1:FAVROLE->node_fav:CONSTITUENT;
30    a-e_3:BENROLE->node_ben:CONSTITUENT;
31
32    negative {
33      a-e_1->node_fav;
34      a-e_2:FAUROLE->node_fau:CONSTITUENT;
35      a-e_3->node_ben;
36    }
37
38    if {
39      e_1.VISITED == false;
40      e_3.VISITED == false;
41    }
42
```

```
43    modify{
44      eval{
45        e_1.VISITED = true;
46        e_3.VISITED = true;
47      }
48      emit("<!-- emitFAVBEN(a) -->\n");
49      emit("<element>\n");
50      emit("  <note>fautor is missing</note>\n");
51      emit("  <question>&quot;" + node_fav.VALUE + "&quot; and &quot;" + node_ben.VALUE
52            + "&quot; modeled...</question>\n");
53      emit("  <option>as method with according parameters (communication possible?)</option>\n");
54      emit("  <option>as classes with associations</option>\n");
55      emit("</element>\n");
56    }
57  }
58
59  rule emitFAVFAU (a:PHRASE) {
60    a-e_1:FAVROLE->node_fav:CONSTITUENT;
61    a-e_2:FAUROLE->node_fau:CONSTITUENT;
62
63    negative {
64      a-e_1->node_fav;
65      a-e_2->node_fau;
66      a-e_3:BENROLE->node_ben:CONSTITUENT;
67    }
68
69    if {
70      e_1.VISITED == false;
71      e_2.VISITED == false;
72    }
73
74    modify{
75      eval{
76        e_1.VISITED = true;
77        e_2.VISITED = true;
78      }
79      emit("<!-- emitFAVFAU(a) -->\n");
80      emit("<element>\n");
81      emit("  <note>beneficiens is missing</note>\n");
82      emit("  <question>&quot;" + node_fav.VALUE + "&quot; and &quot;" + node_fau.VALUE
83            + "&quot; modeled...</question>\n");
84      emit("  <option>as method with according parameters for (communication possible?)</option>\n");
85      emit("  <option>as classes with associations</option>\n");
86      emit("</element>\n");
87    }
88  }
89
90  rule emitFAV (a:PHRASE) {
91    a-e_1:FAVROLE->node_fav:CONSTITUENT;
92
93    negative {
94      a-e_1->node_fav;
95      a-e_2:FAUROLE->node_fau:CONSTITUENT;
96    }
97    negative {
98      a-e_1->node_fav;
99      a-e_3:BENROLE->node_ben:CONSTITUENT;
100   }
101
102   if { e_1.VISITED == false; }
103
104   modify{
105     eval{ e_1.VISITED = true; }
```

```
106     emit("<!-- emitFAV(a) -->\n");
107     emit("<element>\n");
108     emit("  <note>beneficiens is missing</note>\n");
109     emit("  <note>fautor is missing</note>\n");
110     emit("  <question>&quot;" + node_fav.VALUE + "&quot; modeled...</question>\n");
111     emit("  <option>method with according parameters (communication possible?)</option>\n");
112     emit("</element>\n");
113   }
114 }
115
116 rule emitFAU (a:PHRASE) {
117   a-e_2:FAUROLE->node_fau:CONSTITUENT;
118
119   negative {
120     a-e_1:FAVROLE->node_fav:CONSTITUENT;
121     a-e_2->node_fau;
122   }
123   negative {
124     a-e_2->node_fau;
125     a-e_3:BENROLE->node_ben:CONSTITUENT;
126   }
127
128   if { e_2.VISITED == false; }
129
130   modify{
131     eval{ e_2.VISITED = true; }
132     emit("<!-- emitFAU(a) -->\n");
133     emit("<element>\n");
134     emit("  <note>benulus is missing</note>\n");
135     emit("  <note>faverior is missing</note>\n");
136     emit("  <question>&quot;" + node_fau.VALUE + "&quot; modeled...</question>\n");
137     emit("  <option>method with according parameters (communication possible?)</option>\n");
138     emit("</element>\n");
139   }
140 }
141
142 rule emitBEN (a:PHRASE) {
143   a-e_3:BENROLE->node_ben:CONSTITUENT;
144
145   negative {
146     a-e_1:FAVROLE->node_fav:CONSTITUENT;
147     a-e_3->node_ben;
148   }
149
150   negative {
151     a-e_2:FAUROLE->node_fau:CONSTITUENT;
152     a-e_3->node_ben;
153   }
154
155   if { e_3.VISITED == false; }
156
157   modify{
158     eval{ e_3.VISITED = true; }
159     emit("<!-- emitBEN(a) -->\n");
160     emit("<element>\n");
161     emit("  <note>fauio is missing</note>\n");
162     emit("  <note>faverior is missing</note>\n");
163     emit("  <question>&quot;" + node_ben.VALUE + "&quot; modeled...</question>\n");
164     emit("  <option>as method with according parameters (communication possible?)</option>\n");
165     emit("</element>\n");
166   }
167 }
```

Listing E.16: emitFinFic.grg

```
rule emitFINFIC (a:PHRASE) {
  a-e_1:FIN->fingens:CONSTITUENT;
  a-e_2:FIC->fictum:CONSTITUENT;

  if {
    e_1.VISITED == false;
    e_2.VISITED == false;
  }

  modify{
    eval{
      e_1.VISITED = true;
      e_2.VISITED = true;
    }

    emit("<!-- emitFINFIC(a) -->");
    emit("<element>\n");
    emit("  <question>Relationship between &quot;" + fingens.VALUE + "&quot; and &quot;"
          + fictum.VALUE + "&quot; modeled...</question>\n");
    emit("  <option>using inheritance (usually wrong)</option>\n");
    emit("  <option>as association</option>\n");
    emit("  <option>as role</option>\n");
    emit("</element>\n");
    emit("<!-- end of emitFINFIC(a) -->");
  }
}
```

Listing E.17: emitFreq.grg

```
rule emitFREQ (a:PHRASE) {
  a-e_3:FREQ->node_freq:CONSTITUENT;

  if { e_3.VISITED == false; }

  modify{
    eval{ e_3.VISITED = true; }
    emit("<!-- emitFREQ(a) -->");
    emit("<element>\n");
    emit("  <question>as part of an state chart (&quot;" + node_freq.VALUE + "&quot;)</question>");
    emit("  <question>method with comment (counter for &quot;" + node_freq.VALUE + "&quot;)</question>\n");
    emit("</element>\n");
  }
}
```

Listing E.18: emitInst.grg

```
rule emitINSTROLE1(a:PHRASE) {
  a-e:INSTROLE->nodeA:CONSTITUENT;
  a-:ACT->nodeB:CONSTITUENT;

  if { e.VISITED == false; }

  modify {
    eval{e.VISITED = true;}
    emit("<!-- emitINSTROLE1(a) -->");
    emit("<element>\n");
    emit("  <question>&quot;" + nodeA.VALUE + "&quot; modeled...</question>\n");
    emit("  <option>as class</option>\n");
    emit("  <option>as parameter of method &quot;" + nodeB.VALUE + "&quot;</option>\n");
    emit("  <option>as attribute of an association</option>\n");
```

```
15      emit("  <option>a enum</option>\n");
16      emit("  <option>as guarded state transition</option>\n");
17      emit("</element>\n");
18    }
19  }
20
21  rule emitINSTROLE2 (a:PHRASE) {
22    a-e_3:INSTROLE->node_inst:CONSTITUENT;
23
24    negative { a-:ACT->nodeAct:CONSTITUENT; }
25
26    if { e_3.VISITED == false; }
27
28    modify{
29      eval{ e_3.VISITED = true; }
30      emit("<!-- emitINSTROLE2(a) -->");
31      emit("<error>The role &quot;INST&quot;may only be used in conjunction with "
32         + "the role &quot;ACT&quot;!</error>\n");
33    }
34  }
```

Listing E.19: emitMod.grg

```
1   rule emitMOD1 (a:PHRASE) {
2     a-e:MOD->nodeA:CONSTITUENT;
3     a-e2:ACT->nodeB:CONSTITUENT;
4
5     if { e.VISITED == false; }
6
7     modify {
8       eval{ e.VISITED = true; }
9       emit("<element>\n");
10      emit("  <question>&quot;" + nodeA.VALUE + "&quot; modeled...</question>\n");
11      emit("  <option>as class</option>\n");
12      emit("  <option>as parameter of method &quot;" + nodeB.VALUE + "&quot;</option>\n");
13      emit("  <option>as attribute of an association</option>\n");
14      emit("  <option>as an enumeration</option>\n");
15      emit("  <option>guarded state transition (&quot;" + nodeA.VALUE + "&quot;)</option>\n");
16      emit("</element>\n");
17    }
18
19  }
20
21  rule emitMOD2 (a:PHRASE) {
22    a-e_3:MOD->node_mod:CONSTITUENT;
23
24    negative { a-e2:ACT->nodeB:CONSTITUENT; }
25
26    if { e_3.VISITED == false; }
27
28    modify{
29      eval{ e_3.VISITED = true; }
30      emit("<!-- emitMOD(a) -->");
31      emit("<error>The role &quot;MOD&quot;may only be used in conjunction with"
32         + "the role &quot;ACT&quot;!</error>\n");
33    }
34  }
```

Listing E.20: objectRole.grg

```
1   rule emitObjectsAndAttributes(a:PHRASE) {
2     a-role:OBJECTROLE->nodeA:CONSTITUENT;
```

```
3
4   if { nodeA.VISITED == false; }
5
6   modify {
7     emit("<!-- emitObjectsAndAttributes -->\n");
8     exec(helper_emitObjectsAndAttributes(nodeA));
9     emit("<!-- End of emitObjectsAndAttributes -->\n");
10  }
11 }
12
13 rule emitObjectsAndAttributesInSets(a:PHRASE) {
14   a-someEdge:Edge->set:SET;
15   set-role:OBJECTROLE->nodeA:CONSTITUENT;
16
17   if { nodeA.VISITED == false; }
18
19   modify {
20     emit("<!-- emitObjectsAndAttributesInSets -->\n");
21     exec(helper_emitObjectsAndAttributes(nodeA));
22     emit("<!-- End of emitObjectsAndAttributesInSets -->\n");
23   }
24 }
25
26 rule emitOBJECTROLE_class(nodeA:CONSTITUENT) {
27   if { nodeA.VISITED == false; }
28
29   negative { someNode:CONSTITUENT-:EQUALNODES->nodeA; }
30
31   modify {
32     emit("<!-- emitOBJECTROLE_class -->\n");
33     emit("<element>\n");
34     emit("  <question>&quot;" + nodeA.VALUE + "&quot; modeled...</question>\n");
35     emit("  <option>as class</option>\n");
36     emit("  <option>as role</option>\n");
37     emit("  <option>as instance</option>\n");
38     emit("</element>\n");
39     emit("<!-- End of emitOBJECTROLE_class -->\n");
40   }
41 }
42
43 rule helper_emitObjectsAndAttributes(nodeA:CONSTITUENT) {
44   modify {
45     exec(emitOBJECTROLE_class(nodeA));
46     exec(createTransitiveClosureForEqualNodes[*]);
47     exec(helper_emitOBJECTROLE_class_markVisited(nodeA));
48     exec(emitATTRIBUTES(nodeA));
49   }
50 }
51
52 rule helper_emitOBJECTROLE_class_markVisited(nodeA:CONSTITUENT) {
53   modify{
54     eval{nodeA.VISITED = true;}
55   }
56 }
```

Listing E.21: omnPars.grg

```
1 /* OMN-PARS for methods */
2 rule emitOMNPAR_ACT_more (a:PHRASE) {
3   a-e_2:OMN->node_omn:CONSTITUENT;
4   a-e_3:PAR->node_par:CONSTITUENT;
5   a-e_1:PAR->node_par2:CONSTITUENT;
6
```

```
7   negative { a-:OBJECTROLE->node_omn; }
8   negative { a-:OBJECTROLE->node_par; }
9
10  if {
11    e_2.VISITED == false;
12    e_3.VISITED == false;
13    e_1.VISITED == false;
14  }
15
16  modify{
17    /* Only visit this part - maybe there's more... */
18    eval{ e_3.VISITED = true; }
19    emit("<!-- emitOMNPAR_ACT(a) -->\n");
20    emit("<element>\n");
21    emit("  <question>Part and whole (&quot;" + node_omn.VALUE+ "&quot; and &quot;"
22          + node_par.VALUE+ "&quot;) modeled...</question>\n");
23    emit("  <option>as part of an sequence diagram (&quot;" + node_omn.VALUE + "&quot; calls &quot;"
24          + node_par.VALUE+ "&quot;)</option>\n");
25    emit("  <option>ascomment for method &quot;" + node_omn.VALUE+ "&quot; (subcall &quot;"
26          + node_par.VALUE+ "&quot;)</option>\n");
27    emit("</element>\n");
28  }
29  }
30
31  rule emitOMNPARS_ACT_single (a:PHRASE) {
32    a-e_2:OMN->node_omn:CONSTITUENT;
33    a-e_3:PAR->node_par:CONSTITUENT;
34
35    negative { a-:OBJECTROLE->node_omn; }
36    negative { a-:OBJECTROLE->node_par; }
37
38    if {
39      e_2.VISITED == false;
40      e_3.VISITED == false;
41    }
42
43    modify{
44      eval{
45        e_2.VISITED = true;
46        e_3.VISITED = true;
47      }
48      emit("<!-- emitOMNPARS_ACT(a) -->\n");
49      emit("<element>\n");
50      emit("  <question>Part and whole (&quot;" + node_omn.VALUE+ "&quot; and &quot;"
51            + node_par.VALUE+ "&quot;) modeled...</question>\n");
52      emit("  <option>part of an sequence diagram (&quot;" + node_omn.VALUE+ "&quot; calls &quot;"
53            + node_par.VALUE+ "&quot;)</option>\n");
54      emit("  <option>comment for method &quot;" + node_omn.VALUE+ "&quot; (subcall &quot;"
55            + node_par.VALUE+ "&quot;)</option>\n");
56      emit("</element>\n");
57    }
58  }
59
60  /* OMN-PARS for classes */
61  rule emitOMNPARS_CLASS_more (a:PHRASE) {
62    a-e_2:OMN->node_omn:CONSTITUENT;
63    a-e_3:PAR->node_par:CONSTITUENT;
64    a-e_1:PAR->node_par2:CONSTITUENT;
65
66    negative { a-:ACT->node_omn; }
67    negative { a-:ACT->node_par; }
68
69    if {
```

```
70      e_2.VISITED == false;
71      e_3.VISITED == false;
72      e_1.VISITED == false;
73    }
74
75    modify{
76      /* Only visit this part - maybe there's more... */
77      eval{ e_3.VISITED = true; }
78      emit("<!-- emitOMNPARS__CLASS(a) -->\n");
79      emit("<element>\n");
80      emit("  <question>Part and whole (&quot;" + node_omn.VALUE+ "&quot; and &quot;"
81            + node_par.VALUE+ "&quot;) modeled...</question>\n");
82      emit("  <option>aggregation: &quot;" + node_par.VALUE+ "&quot; is part of &quot;"
83            + node_omn.VALUE+ "&quot;</option>\n");
84      emit("  <option>inheritance: &quot;" + node_omn.VALUE+ "&quot; is superclass of &quot;"
85            + node_par.VALUE+ "&quot;</option>\n");
86      emit("</element>\n");
87    }
88  }
89
90  rule emitOMNPARS_CLASS_single (a:PHRASE) {
91    a-e_2:OMN->node_omn:CONSTITUENT;
92    a-e_3:PAR->node_par:CONSTITUENT;
93
94    negative { a-:ACT->node_omn; }
95    negative { a-:ACT->node_par; }
96
97    if {
98      e_2.VISITED == false;
99      e_3.VISITED == false;
100   }
101
102   modify{
103     eval{
104       e_2.VISITED = true;
105       e_3.VISITED = true;
106     }
107     emit("<!-- emitOMNPARS_CLASS(a) -->\n");
108     emit("<element>\n");
109     emit("  <question>Part and whole (&quot;" + node_omn.VALUE+ "&quot; and &quot;"
110           + node_par.VALUE+ "&quot;) modeled...</question>\n");
111     emit("  <option>aggregation: &quot;" + node_par.VALUE+ "&quot; is part of &quot;"
112           + node_omn.VALUE+ "&quot;</option>\n");
113     emit("  <option>inheritance: &quot;" + node_omn.VALUE+ "&quot; is superclass of &quot;"
114           + node_par.VALUE+ "&quot;</option>\n");
115     emit("</element>\n");
116   }
117 }
```

Listing E.22: emitPossHab.grg

```
1   rule emitPOSSHAB (a:PHRASE) {
2     a-e_1:POSS->posessor:CONSTITUENT;
3     a-e_2:HAB->habitum:CONSTITUENT;
4
5     if {
6       e_1.VISITED == false;
7       e_2.VISITED == false;
8     }
9
10    modify{
11      eval{
12        e_1.VISITED = true;
```

```
13      e_2.VISITED = true;
14    }
15    emit("<!-- emitPOSSHAB(a) -->\n");
16    emit("<element>\n");
17    emit("  <question>Relationship between &quot;" + posessor.VALUE + "&quot; and &quot;"
18          + habitum.VALUE + "&quot; modeled...</question>\n");
19    emit("  <option>as association</option>\n");
20    emit("</element>\n");
21    emit("<!-- End of emitPOSSHAB(a) -->\n");
22  }
23 }
```

Listing E.23: emitPreSucc.grg

```
1  rule emitPRESUCC (a:PHRASE) {
2    a-e_2:PRE->node_pre:CONSTITUENT;
3    a-e_3:SUCC->node_succ:CONSTITUENT;
4
5    if {
6      e_2.VISITED == false;
7      e_3.VISITED == false;
8    }
9
10   modify{
11     eval{
12       e_2.VISITED = true;
13       e_3.VISITED = true;
14     }
15     emit("<!-- emitPRESUCC(a) -->\n");
16     emit("<element>\n");
17     emit("  <question>&quot;" + node_pre.VALUE + "&quot; and &quot;" +  node_succ.VALUE
18           + "&quot; modeled...</question>\n");
19     emit("  <option>as part of a sequence diagram</option>\n");
20     emit("  <option>as part of an activity diagram</option>\n");
21     emit("  <option>as successive states in an state chart</option>\n");
22     emit("  <option>as method with a comment</option>\n");
23     emit("</element>\n");
24
25   }
26 }
```

Listing E.24: emitQualQualii.grg

```
1  rule emitQUALQUALII (a:PHRASE) {
2    a-e_1:QUAL->node1:CONSTITUENT;
3    a-e_2:QUALII->node_ii:CONSTITUENT;
4
5    if {
6      e_1.VISITED == false;
7      e_2.VISITED == false;
8    }
9
10   modify{
11     eval{
12       e_1.VISITED = true;
13       e_2.VISITED = true;
14     }
15     emit("<!-- emitQUALQUALII(a) -->\n");
16     emit("<element>\n");
17     emit("  <question>Quality &quot;" + node1.VALUE + "&quot; of &quot;" + node_ii.VALUE
18           + "&quot; modeled...</question>\n");
19     emit("  <option>as association between &quot;" + node1.VALUE + "&quot; and &quot;" + node_ii.VALUE
```

```
20          + "&quot;</option>\n");
21      emit("  <option>as attribute of &quot;" + node_ii.VALUE + "&quot; as &quot;" + node1.VALUE
22          + "&quot;-enumeration (<i>which</i> quality?)</option>\n");
23      emit("  <option>as boolean function of &quot;" + node_ii.VALUE
24          + "&quot; (does it have <i>this</i> quality?)</option>\n");
25      emit("  <option>as scalar function of &quot;" + node_ii.VALUE
26          + "&quot; (to which grade does it have the quality?)</option>\n");
27      emit("  <option>as getters and setters of &quot;" + node_ii.VALUE
28          + "&quot; for the attribute</option>\n");
29      emit("  <option>as enumeration-type for &quot;" + node1.VALUE + "&quot;</option>\n");
30      emit("  <option>as classes for &quot;" + node1.VALUE + "&quot; and &quot;"
31          + node_ii.VALUE + "&quot;</option>\n");
32      emit("</element>\n");
33      emit("<!-- end of emitQUALQUALII(a) -->\n");
34    }
35  }
```

Listing E.25: emitStatStatii.grg

```
1   rule emitSTATSTATII (a:PHRASE) {
2     a-e_2:STATII->node_ii:CONSTITUENT;
3     a-e_3:STAT->node1:CONSTITUENT;
4
5     if {
6       e_2.VISITED == false;
7       e_3.VISITED == false;
8     }
9
10    modify{
11      eval{
12        e_2.VISITED = true;
13        e_3.VISITED = true;
14      }
15      emit("<!-- emitSTATSTATII(a) -->\n");
16      emit("<element>\n");
17      emit("  <question>State &quot;" + node1.VALUE + "&quot; of &quot;" + node_ii.VALUE
18          + "&quot; modeled?</question>\n");
19      emit("  <option>part of a sequence diagram</option>\n");
20      emit("  <option>part of an activity diagram</option>\n");
21      emit("  <option>part of an state chart</option>\n");
22      emit("  <option>a method of another class (implicit)</option>\n");
23      emit("</element>\n");
24      emit("<!-- End of emitSTATSTATII(a) -->\n");
25    }
26  }
```

Listing E.26: emitSubphrases.grg

```
1   rule emitSubclauses(phrase:PHRASE) {
2     phrase-:ROLE->einElement:CONSTITUENT;
3     subphrase:PHRASE-:ROLE->einElement;
4
5     if { subphrase.VISITED == false; }
6
7     negative {
8       -:NEXT->subphrase;
9     }
10
11    modify {
12      exec (emitQuestions(subphrase));
13      exec (emitSubclauses(subphrase)[*]);
14    }
```

```
15 }
```

Listing E.27: emitSubsSubsii.grg

```
1  rule emitSUBSUBSII (a:PHRASE) {
2    a-e_2:SUBSII->node_ii:CONSTITUENT;
3    a-e_3:SUBS->node1:CONSTITUENT;
4
5    if {
6      e_2.VISITED == false;
7      e_3.VISITED == false;
8    }
9
10   modify{
11     eval {
12       e_2.VISITED = true;
13       e_3.VISITED = true;
14     }
15     emit("<!-- emitSUBSUBII(a) -->");
16     emit("<element>\n");
17     emit("  <question>Substitution of &quot;" + node1.VALUE + "&quot; and &quot;"
18           + node_ii.VALUE + "&quot; modeled...</question>");
19     emit("  <option>&quot;" + node1.VALUE + "&quot; and &quot;" + node_ii.VALUE
20           + "&quot; have the same parent class (inheritance with an unknown superclass)</option>");
21     emit("</element>\n");
22   }
23 }
```

Listing E.28: emitSum.grg

```
1  rule emitSUM (a:PHRASE) {
2    a-e_3:SUM->node_sum:CONSTITUENT;
3    if { e_3.VISITED == false; }
4
5    modify{
6      eval{ e_3.VISITED = true; }
7      emit("<!-- emitSUM(a) -->\n");
8      emit("<element>\n");
9      emit("  <question>Precondition &quot;" +node_sum.VALUE + "&quot; modeled...</question>\n");
10     emit("  <option>as part of a sequence diagram</option>\n");
11     emit("  <option>as part of an activity diagram</option>\n");
12     emit("  <option>as OCL-constraint or invariant</option>\n");
13     emit("  <option>as part of an state chart (as state?)</option>\n");
14     emit("  <option>as if-statement in an comment</option>\n");
15     emit("  <option>the needed attributes can be checked</option>\n");
16     emit("</element>\n");
17     emit("<!-- End of emitSUM(a) -->\n");
18   }
19 }
```

Listing E.29: emitTempLoc.grg

```
1  rule emitTEMPLOC(a:PHRASE) {
2    a-e:TEMPLOCROLE->nodeA:CONSTITUENT;
3    if { e.VISITED == false; }
4
5    modify {
6      eval{e.VISITED = true;}
7      emit("<element>\n");
8      emit("  <question>&quot;" + nodeA.VALUE + "&quot; modeled...</question>\n");
9      emit("  <option>as instance of (unknown) class</option>\n");
```

```
10      emit("</element>\n");
11    }
12  }
```

Listing E.30: emitTheTheii.grg

```
1   rule emitTHETHEII (a:PHRASE) {
2     a-e_2:THEII->node_ii:CONSTITUENT;
3     a-e_3:THE->node1:CONSTITUENT;
4
5     if {
6       e_2.VISITED == false;
7       e_3.VISITED == false;
8     }
9
10    modify{
11      eval{
12        e_2.VISITED = true;
13        e_3.VISITED = true;
14      }
15      emit("<!-- emitTHETHEII(a) -->\n");
16      emit("<element>\n");
17      emit("  <question>&quot;" + node1.VALUE + "&quot; and &quot;" + node_ii.VALUE
18            + "&quot; modeled...</question>\n");
19      emit("  <option>as association between &quot;" + node1.VALUE + "&quot; and &quot;"
20            + node_ii.VALUE + "&quot;</option>\n");
21      emit("</element>\n");
22    }
23  }
```

E.3. Sonstige Regeln

Listing E.31: getNextPhrase.grg

```
1   /* Find next phrase that has not been visited yet */
2   rule emitNextSatz {
3     act:CONSTITUENT-e_next:NEXT->next:PHRASE;
4     if {
5       act.VISITED == true;
6       e_next.VISITED == false;
7       next.VISITED == false;
8     }
9     modify {
10      eval{ e_next.VISITED = true; }
11      exec(emitQuestions(next));
12    }
13  }
14
15  /* Create questions for a single phrase */
16  rule emitQuestions(a:PHRASE) {
17    if { a.VISITED==false; }
18    modify {
19      eval {a.VISITED=true;}
20      emit ("<satz value=\"" + a.VALUE + "\">\n");
21      exec ( emitAll(a));
22      emit ("</satz>\n");
23    }
24  }
```

Listing E.32: resetVisited.grg

```
rule reSetVisited {
  modify {
    exec(helper_reSetVisitedNode[*]);
    exec(helper_reSetVisitedNextEdge[*]);
    exec(helper_reSetVisitedRole[*]);
    exec(helper_reSetVisitedHasAttribute[*]);
  }
}

rule helper_reSetVisitedNode {
  x:CONSTITUENT;
  if {
    x.VISITED == true;
    x.VALUE != "root";
  }
  modify { eval { x.VISITED = false ;} }
}

rule helper_reSetVisitedNextEdge {
  :Node-x:NEXT->:Node;
  if { x.VISITED == true; }
  modify { eval { x.VISITED = false ;} }
}

rule helper_reSetVisitedRole {
  :Node-x:ROLE->:Node;
  if { x.VISITED == true; }
  modify { eval { x.VISITED = false ;} }
}

rule helper_reSetVisitedHasAttribute {
  :Node-x:HAS_ATTRIBUTE->:Node;
  if { x.VISITED == true; }
  modify { eval { x.VISITED = false ;} }
}
```

E.4. Ein XSL-T Stylesheet zur Anzeige einer Prüfliste

Listing E.33: XSL-T Stylesheet: Questionnaire.xsl.

```
<?xml version="1.0" encoding="ISO-8859-11"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN" />
<xsl:param name="w3">http://www.w3.org/</xsl:param>

<xsl:template match="/">
<html>
<head>
  <title>Questionnaire</title>
  <style type="text/css">
  table {
    border-collapse: collapse; border: 1px solid black;
  }
```

```
18    table th {
19      border-top: 1px solid black; text-align: left; background-color: #BBBBFF; font-weight: normal;
20    }
21
22    table tr.poss td {
23      font-style: italics; text-align:right; border-top: 1px solid dashed; border-right: 1px solid dashed;
24    }
25
26    table tr.poss td {
27      text-align:right; border-top: 1px solid dashed; border-right: 1px solid dashed;
28    }
29
30    table tr.question td {
31      vertical-align: top; border-top: 1px solid black; font-weight: bold;
32    }
33
34    table tr.error td {
35      color: red;
36    }
37
38    </style>
39  </head>
40  <body>
41
42  <table>
43     <xsl:apply-templates />
44  </table>
45
46  </body>
47  </html>
48  </xsl:template>
49
50  <xsl:template match="phrase">
51    <thead>
52      <tr>
53        <th colspan="5">
54          <b>Phrase: </b><xsl:value-of select="@value"/>
55        </th>
56      </tr>
57    </thead>
58
59    <tbody>
60    <xsl:apply-templates />
61    </tbody>
62  </xsl:template>
63
64  <xsl:template match="error">
65      <tr class="error">
66        <td colspan="3"><b>ERROR:</b><xsl:value-of select="." /></td>
67      </tr>
68  </xsl:template>
69
70  <xsl:template match="element">
71
72  <xsl:variable name="rowCount" select="count(option)+3" />
73      <tr class="question">
74        <td rowspan="{$rowCount}"><xsl:value-of select="question" /></td>
75        <td></td> <td>found</td> <td>not found</td>
76      </tr>
77
78      <xsl:for-each select="option">
79      <tr class="poss">
80        <td><xsl:value-of select="." /></td><td /> <td />
```

```
81      </tr>
82      </xsl:for-each>
83      <tr class="poss">
84        <td>differently (please enter): </td> <td colspan="2"/>
85      </tr>
86
87      <tr class="poss">
88        <td>explicitly not (please enter):</td> <td colspan="2"/>
89      </tr>
90
91      <xsl:for-each select="note">
92      <tr class="note">
93        <td colspan="3"><b>NOTE:</b><xsl:value-of select="." /></td>
94      </tr>
95      </xsl:for-each>
96  </xsl:template>
97
98  </xsl:stylesheet>
```

Literaturverzeichnis

[AEHE07] Aranda, Jorge ; Ernst, Neil ; Horkoff, Jennifer ; Easterbrook, Steve: A Framework for Empirical Evaluation of Model Comprehensibility. In: *MISE '07: Proceedings of the International Workshop on Modeling in Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2953–4, 7 9

[BG07] Blomer, Jakob ; Geiss, Rubino: The GrGen.NET User Manual / Universität Karlsruhe, IPD Goos. Version: July 2007. http://www.info.uni-karlsruhe.de/papers/TR_2007_5.pdf. 2007 (2007-5). – Forschungsbericht. – ISSN 1432-7864 8

[Den07] Denninger, Oliver: *Erweiterung des Kantenkonzepts deklarativer Graphersetzungssysteme von Einfachkanten über Hyperkanten zu SSuperkanten"*, Lehrstuhl Prof. Tichy, Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe (TH), Diplomarbeit, März 2007 5

[Der08] Derre, Bugra: Graphersetzungssysteme als Werkzeuge für UML Modelltransformationen / Universität Karlsruhe, IPD Tichy. 2008. – Forschungsbericht. – Studienarbeit am Institut für Pragrammstrukturen und Datenorganisation (IPD) der Universität Karlsruhe (TH) 27

[DGG08] Denninger, Oliver ; Gelhausen, Tom ; Geiss, Rubino: Applications and Rewriting of Omnigraphs – Exemplified in the Domain of MDD. In: Schürr, A. (Hrsg.) ; Nagl, M. (Hrsg.) ; Zündorf, A. (Hrsg.): *Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '07)* Bd. NN, Springer, 2008 5

[DSC04] Deng, Min ; Stirewalt, R.E. K. ; Chong, Betty H. C.: Retrieval by construction: A traceability technique to support verification and validation of UML formalizations. In: *Internal Journal of Software Engineering and Knowledge Engineering* 15 (2004), Oktober, S. 837–872. http://dx.doi.org/10.1142/S0218194005002531. – DOI 10.1142/S0218194005002531 10

[GBG+06] Geiss, Rubino ; Batz, Gernot V. ; Grund, Daniel ; Hack, Sebastian ; Szalkowski, Adam M.: GrGen: A Fast SPO-Based Graph Rewriting Tool. In: Corradini, A. (Hrsg.) ; Ehrig, H. (Hrsg.) ; Montanari, U. (Hrsg.) ; Ribeiro, L. (Hrsg.) ; Rozenberg, G. (Hrsg.): *Graph Transformations* -

ICGT 2006, Springer, 2006 (Lecture Notes in Computer Science), S. 383 – 397 8

[GDG08] Gelhausen, Tom ; Derre, Bugra ; Geiss, Rubino: Customizing GrGen.NET for Model Transformation. In: *GraMoT 2008*, 2008 27

[Geb06] Gebhart, Michael: Erzeugung natürlicher Sprache aus semantischen Graphen / Universität Karlsruhe, IPD Tichy. 2006. – Forschungsbericht. – Studienarbeit am Institut für Pragrammstrukturen und Datenorganisation (IPD) der Universität Karlsruhe (TH) 19

[GPR06] Gruhn, Volker ; Pieper, Daniel ; Röttgers, Carsten: *MDA*. Springer, 2006. – ISBN 3–540–28744–2, 978–3–540–28744–5 3

[GT07] Gelhausen, Tom ; Tichy, Walter F.: Thematic Role based Generation of UML Models from Real World Requirements. In: *First IEEE International Conference on Semantic Computing (ICSC 2007)* Bd. 0. Irvine, CA, USA : IEEE Computer Society, September 2007, 282-289 ix, 4, 5

[KG07] Kroll, Moritz ; Geiss, Rubino: Developing Graph Transformations with GrGen.NET / Universität Karlsruhe, Fakultät für Informatik, IPD Goos. Version: 2007. http://www.info.uni-karlsruhe.de/papers/agtive_2007_grgennet.pdf. 2007. – Forschungsbericht 6

[KG08] Körner, Sven J. ; Gelhausen, Tom: Improving Automatic Model Creation using Ontologies. In: *Proceedings of the Twntieth International Conference on Software Engineering & Knowledge Engineering (SEKE'2008)*, 2008. – ISBN ???, S. 691–696 6

[Küh06] Kühne, Thomas: Matters of (Meta-) Modeling. In: *Software and Systems Modeling* Volume 5 (2006), 12, Nr. Issue 4, 369–385. http://dx.doi.org/10.1007/s10270-006-0017-9. – DOI 10.1007/s10270–006–0017–9 3

[Lan06] Lange, Christian F. J.: Improving the quality of UML models in practice. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*. New York, NY, USA : ACM, 2006. – ISBN 1–59593–375–1, S. 993–996 ix, 12

[LASE00] Laitenberger, Oliver ; Atkinson, Colin ; Schlich, Maud ; Emam, Khaled E.: An experimental comparison of reading techniques for defect detection in UML design documents. In: *J. Syst. Softw.* 53 (2000), Nr. 2, S. 183–204. http://dx.doi.org/10.1016/S0164-1212(00)00052-2. – DOI 10.1016/S0164–1212(00)00052–2. – ISSN 0164–1212 11

[LC04] Lange, Christian F. J. ; Chaudron, Michel R. V.: An Empirical Assessment of Completedness in UML Designs. In: *EASE '04: Proceedings of the 8th Internation Conference on Empirical Assessment in Software Engineering* Bd. 920, 2004, S. 111–119 10

[LC05] Lange, Christian F. ; Chaudron, Michel R.: Managing Model Quality in UML-Based Software Development. In: Chaudron, M.R.V. (Hrsg.): *Proc. 13th IEEE International Workshop on Software Technology and Engineering Practice*, 2005, S. 7–16 12

[LC06] Lange, Christian F. J. ; Chaudron, Michel R. V.: Effects of Defects in UML models: An Experimental Investigation. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*. New York, NY, USA : ACM, 2006. – ISBN 1–59593–375–1, S. 401–411 11

[LCM06] Lange, Christian F. ; Chaudron, Michel R. ; Muskens, Johan: In Practice: UML Software Architecture and Design Description. In: *IEEE Software* 23 (2006), Nr. 2, S. 40–46. http://dx.doi.org/10.1109/MS.2006.50. – DOI 10.1109/MS.2006.50. – ISSN 0740–7459 11

[LSS94] Lindland, Odd I. ; Sindre, Guttorm ; Sølvberg, Arne: Understanding Quality in Conceptual Modeling. In: *IEEE Softw.* 11 (1994), Nr. 2, S. 42–49. http://dx.doi.org/10.1109/52.268955. – DOI 10.1109/52.268955. – ISSN 0740–7459 13

[MA07] Mohagheghi, Parastoo ; Aagedal, Jan: Evaluating Quality in Model-Driven Engineering. In: *MISE '07: Proceedings of the International Workshop on Modeling in Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2953–4, S. 6 9

[MM03] Miller, J. ; Mukerji, J.: MDA Guide Version 1.0.1 / Object Management Group (OMG). Version: June 2003. http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf. 2003 (omg/03-06-01). – Forschungsbericht ix, 3, 4

[MSUW04] Mellor, Stephen J. ; Scott, Kendall ; Uhl, Axel ; Weise, Dirk: *MDA distilled*. Addison-Wesley, 2004. – ISBN 0–201–78891–8 3

[SCHM+04] Settimi, R. ; Cleland-Huang, O. J.and Ben K. J.and Ben Khadra ; Mody, J. ; Lukasik, W. ; DePalma, C.: Supporting software evolution through dynamically retrieving traces to UML artifacts. In: *IWPSE '04: Proceedings of the Principles of Software Evolution, 7th International Workshop*. Washington, DC, USA : IEEE Computer Society, 2004. – ISBN 0–7695–2211–4, S. 49–54 10

[SDC05] Stirewalt, R. E. K. ; Deng, Min ; Cheng, Betty H. C.: UML formalization is a traceability problem. In: *TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*. New York, NY, USA : ACM, 2005. – ISBN 1–59593–243–7, S. 31–36 10

[WCF04] (World Chess Federation), FIDE Fédération I. É. ; FIDE - Fédération Internationale des Échecs (World Chess Federation) (Hrsg.): *The Chess Handbook*. FIDE - Fédération Internationale des Échecs (World Chess Federation), October 2004. http://www.fide.com/info/handbook?id=32&view=category ix, 17

[Wol06] Wolf, Peter: Interaktive Auswahl mittels kopfgesteuerter Phrasenstrukturgrammatiken erstellter Analysen von Sätzen deutscher Sprache / Universität Karlsruhe, IPD Tichy. 2006. – Forschungsbericht. – Studienarbeit am Institut für Pragrammstrukturen und Datenorganisation (IPD) der Universität Karlsruhe (TH) 6

Index