Austausch von Argumenten in webbasierten Systemen

Bachelorarbeit

von

Emil Warkentin

aus

Düsseldorf

vorgelegt am

Lehrstuhl für Rechnernetze
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

August 2017

Betreuer:
Tobias Krauthoff, M. Sc.

Zusammenfassung

Um Argumentationen im Internet darzustellen gibt es verschiedene Systeme mit unterschiedlichen Vor- und Nachteilen. Um diese Systeme zu vereinen wurde das Argument Interchange Format entwickelt. Ziel dieser Arbeit war es eine Schnittstelle zu entwickeln, die einen Austausch der Argumente zwischen dem Dialog-basierten Argumentationssystem und dem Argument Interchange Format ermöglicht. Dabei sollen die Argumentationen möglichst vollständig übertragen werden. Argumente im Dialog-basierten Argumentationssystem bestehen dabei aus Prämissen und Schlussfolgerungen. Für diese Argumente wurde eine geeignete Übersetzung in das Argument Interchange Format gewählt, welches die Argumente als gerichteten Graph mit Knoten und Kanten darstellt. Zusätzlich wurde auch eine geeignete Übersetzung für die Übertragung der Argumente aus dem Argument Interchange Format in das Dialog-basierte Argumentationssystem implementiert. Durch die entwickelte Schnittstelle lassen sich die Diskussionen aus dem Dialog-basierten Argumentationssystem nahe zu vollständig in die Datenbank Definition des Argument Interchange Format, die AIFdb, übertragen und umgekehrt. Dazu wird zusätzlich ein Web Interface bereitgestellt, um den Austausch ohne große Umwege zu realisieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Im Internet existieren verschiedene Ansätze um Diskussionen darzustellen. Zum einen gibt es Foren, welche zwar sehr einfach zu handhaben sind, aber gerade bei vielen Benutzern und Standpunkten sehr unübersichtlich werden. Oftmals stehen zusammenhanglose Beiträge untereinander und Beiträge, welche inhaltlich gleich sind, werden nach und nach wiederholt. Argumentationskarten, also die Repräsentation von Beiträgen als Knoten in Graphen, deren Bezug zu anderen Beträgen durch Kanten abgebildet wird, sind dagegen übersichtlicher und sortierter, neigen aber, durch eine gewisse Einarbeitungszeit, dazu einen Großteil der Benutzer auszuschließen.

Um sowohl die Einfachheit eines Forum, als auch die Übersichtlichkeit und Komplexität von Argumentationskarten zu vereinen, wird zurzeit am Lehrstuhl Technik sozialer Netzwerke das Dialogbasierte Argumentationssystem, kurz *D-BAS*, entwickelt, welches den Benutzer in einen zeitversetzten Dialog [KBBM16] mit anderen Benutzern treten lässt. Dabei wird der Benutzer nach und nach mit Argumenten konfrontiert, die mit dem eigenen Standpunkt im Konflikt stehen und kann auf diese dann entsprechend eingehen.

Das *Argument Interchange Format*, kurz *AIF*, ist ein Format für Argumente, welches Diskussionen als Argumentationskarten speichert. Ziel des AIF ist es, ein einheitliches und einfaches Format zu bieten, um sowohl die Entwickelung neuer Argumentationssysteme zu erleichtern, als auch die vorhandene Systeme zu verbinden [CMR+06]. Durch verschiedene Import- und Exportfunktionen existiert mittlerweile in der *AIFdb*, einer relationalen Datenbank Definition [LBRS12] des AIF, eine große Vielfalt von Argumentationen aus den gän-

gigsten Systemen, wie Rationale [VG07] und Carneades [GPW07].

Um einerseits Zugriff auf die Vielzahl an Argumentationen in der AIFdb zu bekommen, zum anderen um die Möglichkeit diese Argumentationen mit dem D-BAS zu betrachten und zu erweitern, ist es notwendig eine Schnittstelle zu entwickeln. Diese soll die von den Benutzern angefertigten Diskussionen aus dem D-BAS in die AIFdb übertragen, aber auch die Möglichkeit bieten, Argumentationen aus der AIFdb in das D-BAS zu importieren. Dabei muss eine geeignete Übersetzung der Diskussionen aus dem D-BAS, in das AIF gefunden werden, welche möglichst verlustfrei exportiert. Umgekehrt muss auch die Übersetzung aus dem AIF in das D-BAS Format mit allen Argumentationskarten, welche in der AIFdb existieren, funktionieren. Ein Benutzer soll also ohne zusätzliche Arbeit in der Lage sein, die importierte Argumentation im D-BAS zu verwenden. Das Ganze soll als *microservice* realisiert werden, um unabhängig von D-BAS und AIF zu arbeiten.

In der nachfolgenden Arbeit werden zunächst in Kapitel 2 das D-BAS und das AIF erläutert, außerdem die Art wie Argumente und Diskussionen im D-BAS gespeichert sind. Zusätzlich werden noch die Funktionen der relationalen Datenbank Implementation des AIFs, die AIFdb beschrieben. Anschließend wird in Kapitel 3 ein kurzer Überblick über die Systeme Carneades und Rationale gegeben, da diese eigene Übersetzungen ins AIF bieten. Im darauf folgenden Kapitel4 geht es um den praktischen Teil der Arbeit, in dem zuerst die Funktionen und der Aufbau der Schnittstelle dargestellt werden. Danach folgt die Arbeitsweise der Exportfunktionen, mit deren Hilfe Diskussionen aus dem D-BAS in das AIF übertragen werden. Der zweite Teil des Kapitels stellt die Importfunktion aus dem AIF in das D-BAS dar. Zum Schluss enthält Kapitel5 ein Fazit sowie einen Ausblick welche funktionalen Erweiterungen noch zu erwägen sind.

Kapitel 2

Grundlagen

2.1 Dialog basiertes Argumentationssystem

In den folgenden zwei Abschnitten wird zuerst das D-BAS und seine Arbeitsweise vorgestellt. Danach folgt der Aufbau der Datenbank, die das D-BAS nutzt und die Art und Weise wie das D-BAS ein Argument speichert.

2.1.1 Beschreibung

Das Dialogbasierte Argumentationssystem, kurz D-BAS, ist ein System, welches dazu dient Diskussionen im Internet zu verbessern. Da Foren häufig schlecht für viele Benutzer und viele Themen skalieren, soll das D-BAS dem Benutzer ermöglichen, „ohne eine Einarbeitungszeit an groß angelegten Online Diskussionen“ [KBBM16] teilnehmen zu können, welche dennoch übersichtlich sind. Dazu wählt der Nutzer ein für ihn relevantes Thema aus, zu dem er diskutieren möchte und wählt dann entweder den Standpunkt von anderen Leuten aus oder gibt seinen eigenen Standpunkt an. Aus allen zu dem Standpunkt relevanten Argumenten wählt das D-BAS dann eines aus und konfrontiert den Benutzer damit. Daraufhin fordert das D-BAS den Benutzer auf zu dem Argument eine Rückmeldung, in Form von Zustimmung oder Ablehnung, zu geben. Je nach Auswahl, kann der Benutzer die Ablehnung wahlweise mit den Argumenten von anderen Benutzern oder seinen eigenen begründen, woraufhin das

D-BAS wieder gegenargumentiert. Dadurch entsteht nach und nach ein Dialog zwischen D-BAS und Benutzer. Dieser Dialog endet dann, wenn einer der Beiden kein Argument mehr für seinen Standpunkt vorbringen kann oder der Benutzer von einem anderen Standpunkt überzeugt wurde. Das D-BAS speichert dabei die Argumente des Benutzers und verwendet diese wieder, um damit mit anderen Benutzern in Dialog zu treten. Das D-BAS repräsentiert also alle Benutzer gleichzeitig, wählt aber immer nur ein Argument aus, wodurch trotz vieler verschiedener Benutzer ein einfacher Dialog entsteht. Da das D-BAS die Diskussionen leitet, kann es weder vorkommen, dass ein Benutzer ein Argument wiederholt vorgesetzt bekommt, noch dass es nicht zum Thema passt. Dies verringert die ”Redundanz und Balkanisierung und verringert das Auftreten von logischen Irrtümern” [KBBM16], was, zusammen mit der Einfachheit eines Dialoges, einen klaren Vorteil gegenüber einer Diskussion in einem Forum oder Ähnlichem bietet. [KBBM16]

2.1.2 Interner Aufbau

Für die verschieden Funktionen des D-BAS existiert eine zentrale Datenbank. Zum einen existieren Tabellen die die Interaktion mit dem Benutzer regulieren. Um zu verhindern das ein Benutzer ein Argument mehrfach sieht, dienen unter anderen die Tabellen *seen_arguments* und *seen_statements*. Für das „review“ System benutzt die D-BAS Datenbank Tabellen wie *review_deletes*, *review_edits* und *review_canceled*. Da, sowohl das „review“ System, als auch die Interaktion mit dem Benutzer für den Im- und Export der Diskussionen keine Relevanz haben, werden sie nicht näher betrachtet. [KBBM16]

Die Diskussionen sind unter der Tabelle *issues* gespeichert, welche das Thema, zum Beispiel „Kann Birdy fliegen“, speichert. Zu einem Thema existieren verschiedene *statements*, welche den textlichen Bestandteil einer Prämisse, Schlussfolgerung oder einer Aussage bilden. Eine Aussage stellt dabei den Startpunkt einer Diskussion dar, zum Beispiel „Birdy kann fliegen“. Aussagen können nun von Argumenten unterstützt oder angegriffen werden. Argumente bestehen dabei aus Prämissengruppen und Schlussfolgerungen, siehe Abbildung 2.1. Zum Beispiel bildet die Prämissengruppe, die aus den Prämissen „Birdy ist ein Vogel“ und „Vögel können normalerweise fliegen“ besteht, zusammen mit der Schlussfolgerung „Birdy kann fliegen“ das Argument „Birdy ist ein Vogel und Vögel können normalerweise fliegen, deshalb kann Birdy fliegen“. Prämissen, beziehungsweise Prämissengruppen, können dabei entweder unterstützend für eine Schlussfolgerung sein oder nicht unterstützend. Falls

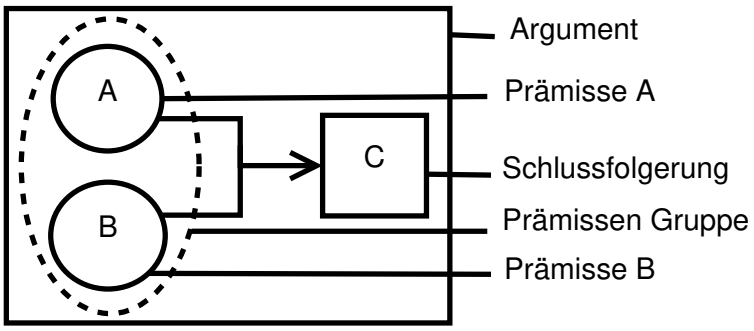

Abbildung 2.1: Schematischer Aufbau eines Argumentes

sie nicht unterstützend sind, folgt aus der Prämissengruppe genau die Negation der Schlussfolgerung, zum Beispiel bildet die Prämisse „Birdy ist ein Hund“ und die Schlussfolgerung „Birdy kann fliegen“ , das Argument „Birdy ist ein Hund und kann deshalb nicht fliegen“. Außerdem können Argumente auch das Folgern einer Schlussfolgerung aufgrund der Prämissen widerlegen. Die Prämissengruppe greift somit die Prämissen des Argumentes an, aus denen die Schlussfolgerung gezogen wird. Zum Beispiel widerlegt die Prämisse „Birdy ist kein normaler Vogel“ , dass das Folgern der Schlussfolgerung aus den Prämissen „Birdy ist ein Vogel“ und „Vögel können normalerweise fliegen“ möglich ist. Dies stellt einen so genannten „Undercut“ dar. [KBBM16]

2.2 Argument Interchange Format

In den folgenden beiden Abschnitten wird erklärt, welche Ziele das AIF verfolgt und wie Argumente im AIF gespeichert werden. Außerdem wird die AIFdb erklärt, eine relationale Datenbank Definition des AIF, mitsamt einer kurzen Erklärung, wie das Format eines Argumentes in der AIFdb aussieht.

2.2.1 Beschreibung

Das *Argument Interchange Format*, kurz AIF, ist ein Ansatz, um eine „geteilte vereinbarte Notation für Argumente und Argumentationen“ [CMR+06] zu bieten. Diese soll Daten von verschiedenen Argumentationen angemessen repräsentieren und den Austausch von Daten

zwischen verschieden Argument-basierten Systemen ermöglichen [CMR+06]. Dazu soll das AIF nur ein Grundkonzept darstellen, welches erweiterbar ist, um eine hohe Kompatibilität zwischen verschiedenen Systemen zu gewährleisten. [CMR+06]

Das AIF unterscheidet zwischen den Informationen eines Arguments und seinen Zusammenhängen.Das AIF speichert diese Unterscheidungen dann als gerichteten Graph ab, welcher durch Kanten und zwei Knotentypen, namentlich *scheme nodes* und *information nodes*, kurz *S-Nodes* und *I-Nodes*, ein so genanntes *argument network* bildet. Die I-Nodes repräsentieren dabei den textlichen Bestandteil einer Aussage. S-Nodes verknüpfen zwei Knoten und stellen so den logischen Zusammenhang zwischen ihnen her. In der Grundidee des AIF sind S-Nodes entweder „rule of inference application nodes“ , kurz *RA-Nodes* oder „conflict application nodes“ , kurz *CA-Nodes* [CMR+06]. Im AIF existieren dazu noch so genannte ”preference application nodes”, kurz *PA-Nodes*, welche jedoch, bedingt durch den Aufbau der Diskussionen im D-BAS, nicht benötigt werden und somit im Sinne der Übersichtlichkeit außen vor gelassen werden. [CMR+06]

Knoten werden durch Kanten miteinander verbunden, wobei zwischen zwei I-Nodes immer mindestens ein S-Node stehen muss. Das AIF erlaubt nicht I-Nodes direkt mit anderen I-Nodes zu verbinden, da sonst der argumentative Sachverhalt zwischen den beiden Knoten fehlt. Dieser wird durch die S-Nodes ausgedrückt, welche den I-Nodes ihre Funktion als Prämisse oder Schlussfolgerung zuteilen. S-Nodes können aber auch mit anderen S-Nodes verbunden werden, um verschiedene argumentative Sachverhalte einer Diskussion auszudrücken, wie zum Beispiel das Anzweifeln der Schlussfolgerung aufgrund einer bestimmten Prämisse. Eine Folge von miteinander in Verbindung stehender S-Nodes, muss allerdings mit einem I-Node enden, beziehungsweise anfangen. S-Nodes können also nicht für sich alleine stehen. Die nachfolgende Tabelle zeigt, wie man durch die Knoten die verschiedenen argumentativen Zusammenhänge, welche für die Übersetzung aus dem D-BAS eine Rolle spielen, im AIF darstellen kann. [CMR+06]

	Zu I-Node	Zu RA-Node	Zu CA-Node
Von I-Node		Aussage, die unterstützende Prämisse bildet	Aussage, die nicht unterstützende Prämisse bildet
Von RA-Node	Ableiten einer Schlussfolgerung aus Prämisse		
Von Ca Node	Ableiten der negierten Schlussfolgerung	Prämisse widerlegt die Ableitung der Schlussfolgerung	Prämisse widerlegt die Wiederlegung der Ableitung der Schlussfolgerung

Tabelle 2.1: Erläuterung der relevanten Relationen von Knoten, basierend auf [CMR+06]

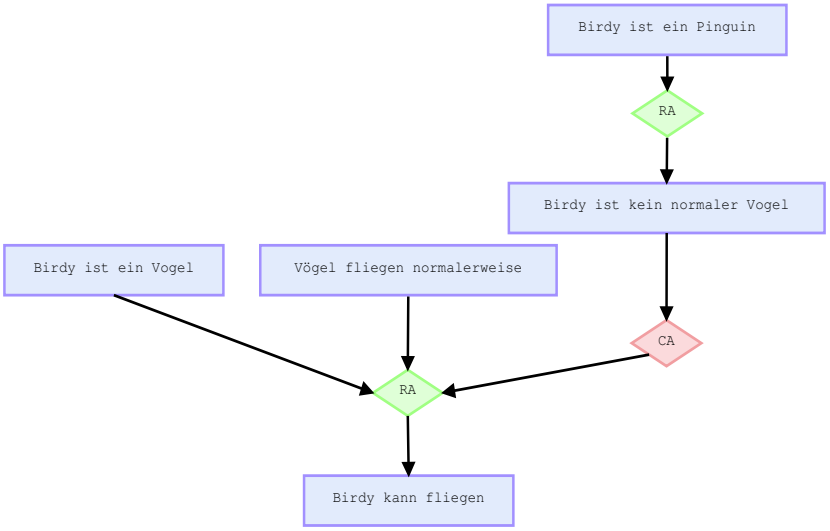

Abbildung 2.2: Graphische Darstellung einer Diskussion in AIFDB

2.2.2 AIFdb

Die AIFdb [1] ist eine Datenbank, in welcher Diskussionen im *argument interchange format* gespeichert und geladen werden können. Die gespeicherten Diskussionen sind für den Benutzer als Graph einsehbar und über eine Suchfunktion aufzufinden. Außerdem bietet die AIFdb die Möglichkeit, diesen Graph in eine Vielzahl an Formaten zu übersetzten, wie zum

[1] http://www.aifdb.org/

Beispiel Carneades[2], Rationale[3] und Dot[4]. Des Weiteren können die Graphen mithilfe von verschiedenen Tools evaluiert und analysiert werden. Die, für die nachfolgende Arbeit, wichtigste Funktion ist der Export des Graphen aus der AIFdb als *nodeSet*, welches man über eine Schnittstelle als JSON Objekt exportieren kann. [BGLR12]

In diesem *nodeSet* ist der Graph in *nodes* und *edges* unterteilt. *Nodes* bestehen aus einer eindeutigen *nodeID*, aus einem Eintrag *text*, einem Eintrag *type* und einem Eintrag *timestamp*. *Edges* bestehen aus einer eindeutigen *edgeID*, einer *fromID* und einer *toID*, welche angeben, von welchem Knoten, zu welchem Knoten eine Kante verläuft und aus einer *formEdgeID*, welche die Form der Kante bestimmt und standardmäßig auf *null* initialisiert wird. [BGLR12]

[2]http://carneades.fokus.fraunhofer.de/carneades/
[3]https://www.rationaleonline.com/
[4]http://www.graphviz.org/

Kapitel 3

Verwandte Arbeiten

Da es eine Hauptfunktion des AIF ist, Daten zwischen verschiedenen Argumentationssystemen auszutauschen, existiert in der AIFdb die Funktion, in die gängigsten Formate für diese Systeme zu exportieren.

Eines dieser gängigen Formate ist Carneades. Carneades arbeitet, ähnlich wie das D-BAS, mit *statements* und *arguments*. Es bietet sowohl die Möglichkeit die Diskussionen zu evaluieren, als auch sie in einem Graph, ähnlich dem des AIF, anzuzeigen. Argumente im Carneades Graph haben eine *polarity* entweder *pro* oder *con*. Außerdem bestehen Argumente noch aus einem oder mehreren *statements* welche entweder die Prämissen bilden und als einer Schlussfolgerung dienen. Optional haben Argumente in Carneades auch noch *execptions*, welche gleichbedeutend mit dem aus dem D-BAS bekannten Undercut sind. Aus der Polarität der Prämissen folgt, ob die Prämisse die Schlussfolgerung unterstützt oder ob aus der Prämisse genau das Gegenteil der Schlussfolgerung folgt. Die Übersetzung in das AIF erstellt aus den Prämissen mit der Polarität *pro* RA-Nodes und aus den Prämissen mit der Polarität *con* CA-Nodes. Aus den *statements* werden die I-Nodes gebildet, wobei in Carneades, im Gegensatz zum AIF, auch negative Prämissen existieren. Dieses Problem wird durch das Umschreiben der *statements* in das Gegenteil umgangen, was allerdings beim Importieren aus dem AIF in Carneades zu einem veränderten Graphen führt. [BGLR12]

Ein anderes Format ist Rationale. Rationale ist ein „Argument Karten Software Paket“ [VG07]. Die Karten werden dazu per *Drag and Drop* aus Prämissen, Schlussfolgerungen und Wiederlegungen aufgebaut und mit Aussagen beschrieben. Außerdem lassen sich Aussagen direkt

mit Quellen verknüpfen, wie zum Beispiel wissenschaftliche Arbeiten und Internetseiten. Ein Nachteil von Rationale ist, dass aufgrund seiner Komplexität eine gewisse Einarbeitungszeit benötigt wird. Für die Übersetzung vom Rationale Format in das AIF existiert zum derzeitigen Zeitpunkt keine frei zugängliche wissenschaftliche Arbeit.

Kapitel 4

Austausch von Argumenten zwischen D-BAS und AIF

4.1 Schnittstelle

Um dem Benutzer eine übersichtliche und einfache Möglichkeit zu geben Diskussionen zu im- und exportieren, läuft die Schnittstelle auf einem Java Spark[1] Webserver und ist unter localhost:5678 zu erreichen. Der Server verbindet sich beim Start mit der D-BAS Datenbank und gibt dem Benutzer entsprechende Rückmeldung, falls keine Verbindung möglich ist. Ist der Server mit der Datenbank verbunden kann der Benutzer Diskussionen mithilfe einem Interface(Abbildung 4.1) im- und exportieren.

Für den Export wählt der Benutzer aus einem Dropdown Menü aus, welches Thema er vom D-BAS in das AIF übertragen möchte. Nach der Auswahl des Themas wird die ID des ausgewählten Themas per Ajax Request an den Server übermittelt, welcher daraufhin ein JSON Objekt aus der Diskussion im D-BAS erstellt. Dieses wird dann in eine Datei geschrieben, welche der Server dann über die AIF Schnittstelle[2] in die AIFdb hochlädt. Während des Vorgangs erhält der Benutzer durch ein Ladesymbol die Bestätigung, dass die Diskussion importiert wird. Ist der Vorgang beendet, schickt der Server eine Rückmeldung an die Schnittstelle, welche daraufhin dem Benutzer anzeigt, dass der Export erfolgreich war. Zusätzlich dazu

[1] http://sparkjava.com/
[2] http://ws.arg.tech/

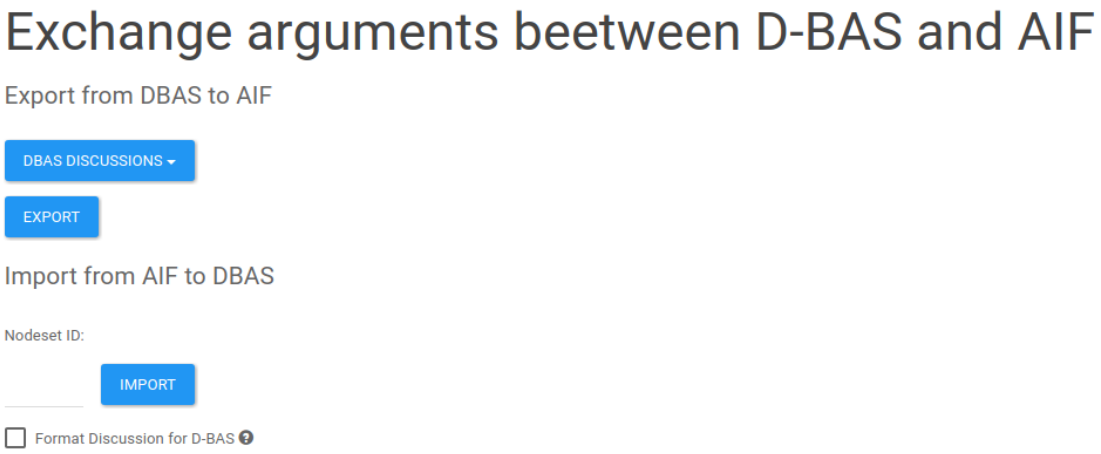

Abbildung 4.1: Schnittstelle zum Austausch von Argumenten

erhält der Benutzer die *nodeSetID*, unter der sich die exportierte Diskussionen in der AIFdb finden lässt.

Für den Import muss der Benutzer vorher in der AIFdb eine Diskussion auswählen und die *NodeSetID* kopieren. Diese trägt der Benutzer dann an der entsprechenden Stelle in der Schnittstelle ein. Der Server holt sich nun von der AIF Schnittstelle den der *NodeSetID* entsprechenden Graphen als *JSON Objekt* und beginnt das Objekt in das D-BAS Format zu übertragen. Nach der Übersetzung fügt die Schnittstelle die Diskussion der D-BAS Datenbank hinzu. Ist der Vorgang erfolgreich, bekommt der Benutzer eine entsprechende Rückmeldung und kann die importierte Diskussion im D-BAS benutzen. Nach den ersten Imports von Diskussionen aus dem AIF ist aufgefallen, dass Argumente mehrfach in der Datenbank des D-BAS existierten. Da die Argumente, in Form von duplizierten S-Nodes, allerdings schon im AIF mehrfach vorhanden sind, bietet die Schnittstelle optional die Funktion die Diskussion zu formatieren. Durch die Formatierung werden die vervielfachten Kanten und Knoten, welche keine logische Funktion haben, vor dem Einfügen in die D-BAS Datenbank gelöscht.

4.2 Export vom D-BAS zu AIF

Der Export vom D-BAS in das AIF erfolgt in den folgenden sieben Schritten:

1. 4.2.1 Daten aus D-BAS Datenbank beziehen
2. 4.2.2 I-Nodes aus Aussagen erstellen
3. 4.2.3 S-Nodes aus Prämissen und Schlussfolgerung erstellen
4. 4.2.4 Kanten für S-Nodes aus Prämissen und Schlussfolgerung erstellen
5. 4.2.5 S-Nodes aus Undercuts erstellen
6. 4.2.6 Kanten für S-Nodes aus Undercuts erstellen
7. 4.2.7 Hochladen des erstellten JSON Objekts in die AIFdb

4.2.1 Daten aus D-BAS Datenbank beziehen

Im ersten Schritt müssen zunächst aus der Datenbank des D-BAS alle relevanten Informationen extrahiert werden. Im Gegenteil zum D-BAS besitzt das AIF keine Möglichkeit einen Dialog mit den Benutzern zu führen. Daher fallen alle Tabellen aus der Datenbank weg, welche den Dialog im D-BAS regeln. Außerdem fällt das Review System raus, da es auch keine Verwendung bei der Übersetzung findet. Es bleiben die Tabellen *issues*, *textversions*, *statements*, *premises*, *premisegroups*, *arguments*, *users*. Da das AIF zwar die Möglichkeit bietet neue Benutzer zu registrieren, aber nicht nach einer erfolgreichen Registrierung die relevante ID als Rückgabewert abzufragen, muss die Tabelle *users* gestrichen werden. Jeder Eintrag in der Tabelle *premises* hat eine *premisegroup_uid* gesetzt, welche direkt übernommen werden kann, weshalb die Tabelle *premisegroups* nicht übersetzt wird. Die Tabelle *issues* wird nur benötigt um das Dropdown Menü in der Schnittstelle zu befüllen, für die Übersetzung der Diskussion in das AIF wird sie jedoch nicht mehr benötigt. Damit bleiben *textversions*, *statements*, *premises* und *arguments* als zu übersetzende Tabellen.

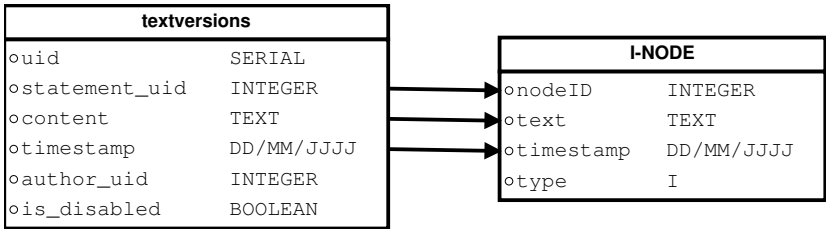

Abbildung 4.2: Übersetzung der Tabelle Textversions zu I-Node

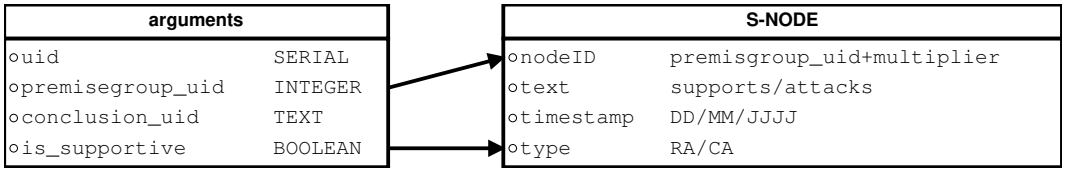

Abbildung 4.3: Übersetzung der Tabelle Arguments zu S-Node

4.2.2 Übersetzten der Aussagen in I-Nodes

Die AIF Schnittstelle erlaubt das Hochladen von JSON Dateien, welche der AIF Notation folgen. Es muss also eine Übersetzung dieser vier Tabellen in das AIF Format erfolgen. Dazu werden zwei *JSON Arrays* angelegt, eins für die *edges* und eins für die *nodes*. Zuerst werden aus den Tabellen *textversions* und *statements* die I-Nodes erstellt. Wichtig hierbei ist, dass nur die *statements* übersetzt werden, welche der ausgewählten *issue* entsprechen. Für jedes Statement wird ein Knoten als JSON Objekt erstellt, welcher als *nodeID* die UID des *statements* erhält, als *text* den Eintrag *content* aus der Tabelle *textversions*, als *typ* I und als *timestamp* den entsprechenden aus der Tabelle *textversions*, siehe Abbildung 4.2. Der Eintrag *is_startpoint* muss nicht übersetzt werden, da Knoten, die Startpunkte sind, bei der Erstellung der Kanten keine ausgehenden Kanten erhalten. Danach werden alle erstellten JSON Objekte dem *edges* JSON Array hinzugefügt, womit die Übersetzung der Tabellen *textversions* und *statements* abgeschlossen ist.

4.2.3 Übersetzung von Prämisse und Schlussfolgerung zu S-Node

Aus den Tabellen *premises* und *arguments* werden im nächsten Schritt die S-Nodes erstellt. Für die Erstellung der RA und CA Knoten werden von der D-BAS Datenbank alle Argumente benötigt, welche sowohl eine *premisesgroup_uid* und eine *conclusion_uid* gesetzt haben. Au-

ßerdem muss, je nach Knotentyp, bei dem Feld *is_supportive* der Eintrag *true* für RA Knoten und *false* für CA Knoten gesetzt sein. Danach muss die Tabelle *arguments* mit der Tabelle *premises* über die *premisegroup_uid* vereinigt werden, um Zugriff auf die *statement_uid* zu bekommen. Diese ist wichtig, um die erstellten S-Nodes mit den erstellten I-Nodes zu verbinden, da die I-Nodes als NodeID die *statement_uid* haben. Bei der Erstellung der S-Nodes wird die Liste der Argumente durchgegangen und für jede Prämissengruppe ein S-Node erstellt. Da es aber Prämissengruppen gibt, welche für eine Schlussfolgerung unterstützend sind, für eine andere aber nicht, errechnet sich die *NodeID* aus der ID der Prämissengruppe und einem Multiplikator für den *type* des Knotens, also CA oder RA, was das Problem löst. Außerdem könnte die *premisegroup_uid* mit einer der vergebenen I-Node IDs übereinstimmen, was auch zu falschen Kanten führen würde. Deshalb erhalten alle S-Nodes einen Multiplikator, 10000 für Knoten vom Typ RA und 20000 für Knoten vom Typ CA. Im Eintrag *text* des Knotens steht je nach Typ *supports* oder *attacks*. Der Eintrag *timestamp* wird von dem Argument übernommen. Die Einträge *author_uid* und *is_disabled* können nicht übersetzt werden und fallen weg. Außerdem fällt der Eintrag *is_negated* weg, da das AIF keine negativen Prämissen benutzen kann. Alle zur Übersetzung relevanten Einträge sind nochmal schematisch in Abbildung 4.3 dargestellt. Sind alle S-Nodes erstellt, werden die IDs der erstellten Knoten in einer Liste gespeichert, da sie im nächsten Schritt für die Erstellung der Kanten benötigt werden. Außerdem werden alle erstellten S-Nodes dem *nodes* JSON Array hinzugefügt.

4.2.4 Erstellung der Kanten für S-Nodes aus Prämisse und Schlussfolgerung

Die zu erstellenden Kanten haben eine *edgeID*, eine *fromID* und eine *toID*, zusätzlich noch eine *formEdgeID*, welche aber immer auf *null* gesetzt wird. Die *edgeID* ist beliebig, da sie später im AIF automatisch geändert wird. Die erste Kante verläuft vom I-Node zum S-Node. Als *fromID* wird also die *statement_uid* der Prämisse des aktuellen Arguments gesetzt, als *toID* der zugehörige S-Node, also die *premisegroup_uid* plus dem Multiplikator. Die zweite Kante hat als *fromID* dann die ID des S-Nodes gesetzt und als *toID* die *conlcusion_uid*, welche direkt auf eine *statement_uid* verweist, somit auch direkt auf den erstellten I-Node. Das Ganze wird in Abbildung Abbildung 4.4 noch einmal schematisch dargestellt. Alle erstellten Kanten werden dann dem *edges* JSON Array hinzugefügt, wodurch der erste Teil der

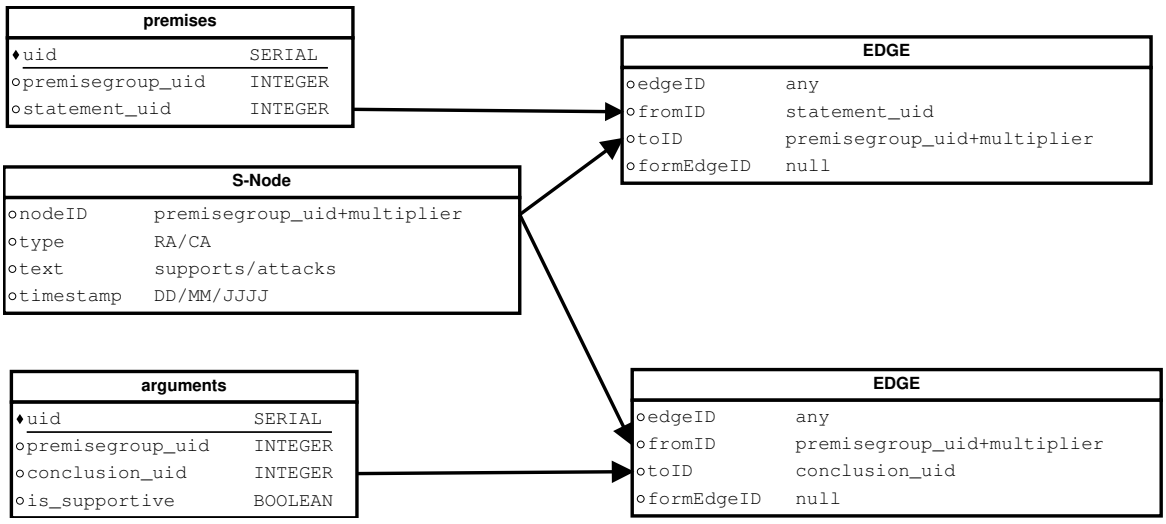

Abbildung 4.4: Erstellung der Kanten aus Tabellen Arguements, Premises und S-Node

Übersetzung in das AIF fertig ist.

4.2.5 Übersetzung der Undercuts zu S-Nodes

Im zweiten Teil werden nun die Argumente in das AIF übersetzt, welche statt einer *conclusion_uid* einen Fremdschlüssel auf ein anderes Argument gesetzt haben, also den Eintrag *argument_uid*. Dazu müssen aus der Datenbank alle Argumente mit dieser Eigenschaft extrahiert werden. Diese Argumente greifen dann mit ihrer Prämissengruppe das Argument an, auf das der Fremdschlüssel verweist. Die erstellten S-Nodes werden wieder wie vorher erstellt und erhalten als *nodeID* die *premisegroup_uid* des ersten Arguments, plus wieder einen Multiplikator, diesmal 30000. *Type* und *text* sind wieder CA und *attacks*, der *timestamp* wird von dem ersten Argument übernommen, der Rest wie *author_uid* und *issue_uid* wird wieder verworfen.

4.2.6 Erstellung der Kanten für S-Nodes aus Undercuts

Die Erstellung der Kanten läuft jedoch anders ab als bei den anderen S-Nodes, da die Argumente nicht auf eine *conclusion_uid* verweisen, sondern auf ein anderes Argument. Die

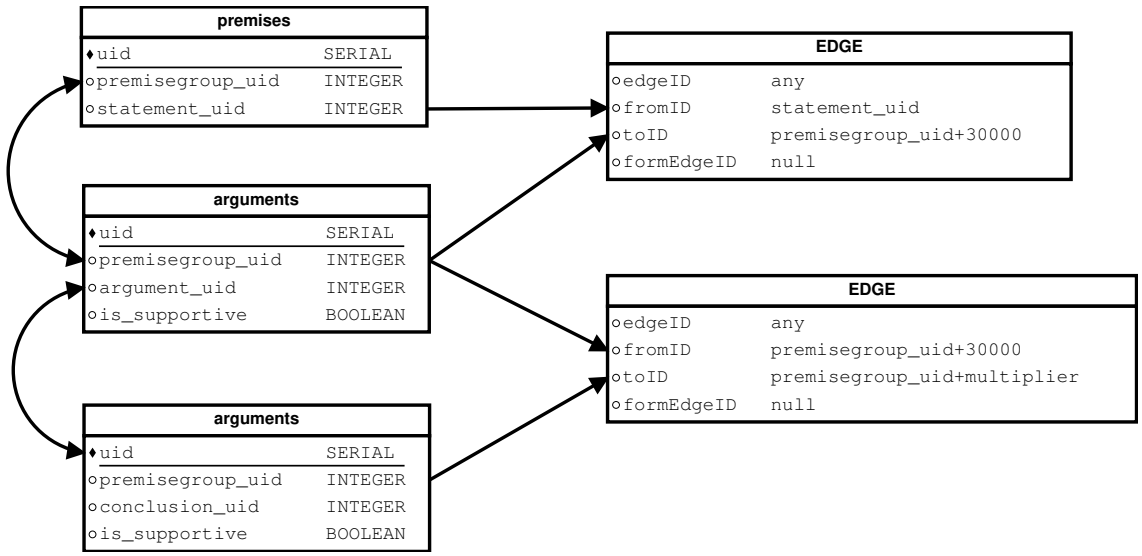

Abbildung 4.5: Erstellung der Kanten aus Undercuts

Kante im AIF verläuft also von einem I-Node zu einem S-Node und dann zu einem S-Node. Um nun die richtigen Knoten miteinander zu verbinden, benötigt der Algorithmus die IDs von allen erstellten S-Nodes. Auf die *premisegroup_uid* des durch den Fremdschlüssel ausgewiesenen Argumentes wird nun jeder Multiplikator, welcher bei der Erstellung der S-Nodes benutzt wurde, einmal addiert. Danach wird überprüft, ob der errechnete Wert einem schon erstellten S-Node entspricht. Ist dies der Fall, stellt der errechnete S-Node die Schlussfolgerung dar, er ist also der angegriffene Knoten und wird in einer Liste zwischengespeichert. Für den Fall einer Verkettung, also das ein Argument, welches ein anderes Argument angreift wieder von einem Argument angegriffen wird, ist der angegriffene S-Node wieder die *premisegroup_uid* des Fremdschlüssel Argument. Nachdem nun die Liste der S-Nodes angelegt wurde, welche die Rolle der Schlussfolgerungen einnehmen, werden die Kanten wie für normale S-Nodes erstellt. Aus der *statement_uid* der Prämissen folgt die erste *fromID*. Die *toID* ist gleich der *nodeID* des S-Nodes, welche gleichzeitig auch die *fromID* der zweiten Kante darstellt. Die *toID* der zweiten Kante ist dann die *nodeID* aus der Liste der S-Nodes welche zuvor angefertigt wurde (Abbildung 4.5). Die *edgeID* ist wieder beliebig, der *timestamp* wird wieder vom Argument übernommen und *formEdgeID* ist wieder *null*.

4.2.7 Hochladen des erstellten JSON Objekts ins AIF

Nach der erfolgreichen Erstellung der Kanten werden sie dem *edges* JSON Array hinzugefügt, aus dem, zusammen mit dem *nodes* JSON Array, eine JSON Datei erstellt wird. Diese JSON Datei wird dann über die Upload Funktion der AIFdb [3] hochgeladen. Unter der zurückgegeben *nodeSetID* befindet sich nun die Diskussion als AIF Graph in der AIFdb.

4.3 Import von AIF zu D-BAS

Der Import aus dem AIF in das D-BAS geschieht in den folgenden sechs Schritten:

1. 4.3.1 Nodeset über AIF Schnittstelle anfordern
2. 4.3.2 S-Nodes zu Prämissen und Schlussfolgerungen umwandeln
3. 4.3.3 Einsortierung der Prämissen in Prämissengruppen
4. 4.3.4 Entfernung doppelter Knoten
5. 4.3.5 Einfügen der Daten in die Tabellen der D-BAS Datenbank
6. 4.3.6 Abschluss des Imports

4.3.1 Nodeset über AIF Schnittstelle anfordern

Für den Import von Diskussionen aus dem AIF in das D-BAS muss zunächst über die Schnittstelle der AIFdb [4] das ausgewählte *nodeSet* heruntergeladen werden. Nach dem Einlesen wird die Datei, welche im JSON Format gespeichert ist, nach *nodes* und *edges* unterteilt. Die Restmenge der *nodes* wird danach in verschiedenen Arrays nach ihrem jeweiligen Typ unterteilt,

[3] http://www.arg.dundee.ac.uk/AIFdb/json/
[4] http://ws.arg.tech/

also I-Nodes, RA-Nodes und CA-Nodes. Um die Laufzeit der Algorithmen zu verbessern werden außerdem alle IDs in den JSON Arrays von *Strings* zu *Integer* Werten umgewandelt, um später das Einfügen in die Datenbank zu erleichtern. Um später den *Boolean is_startpoint* in der Tabelle *statements* festzulegen, werden alle Kanten und I-Nodes betrachtet. Im Fall, dass ein Knoten, beziehungsweise seine *nodeID*, keiner Kante der *fromID* entspricht, handelt es sich um einen Startpunkt. Daraufhin wird diese Information im JSON Objekt gespeichert, welches den I-Node darstellt. Außerdem wird gleichzeitig eine zusätzliche Liste angelegt, in der sich alle Knoten befinden, die nicht I-Nodes sind, welche für das Finden der Prämissen und Prämissengruppe wichtig ist.

4.3.2 S-Nodes zu Prämissen und Schlussfolgerungen umwandeln

Als nächster Schritt gilt es die ID der I-Nodes zu finden, welche durch Kanten mit einem S-Node verbunden sind, um so Prämissen mit ihrer jeweiligen Schlussfolgerung zu verbinden. Dies wird dadurch möglich, dass die ID eines S-Nodes mindestens in zwei Kanten vorkommt, entweder als *fromID* oder als *toID*. Die zugehörige *toID*, beziehungsweise *fromID*, sind dann die IDs der mit dem S-Node verbundenen I-Nodes. Es werden also für jeden RA und CA Knoten alle Kanten zwei Mal untersucht. Stimmt die *nodeID* des Knoten mit der *toID* einer Kante überein, wird zunächst mit der zuvor erstellten Liste der S-Nodes überprüft, ob es sich bei der *fromID* um einen I-Node handelt. Entweder es handelt sich um einen I-Node, dann ist der Knoten die Prämisse oder es handelt sich um einen S-Node, dann wird die nächste Kante untersucht. Handelt es sich um eine Prämisse, wird die *fromID* zwischengespeichert und ein erneuter Schleifendurchlauf über alle Kanten begonnen.

Diesmal wird jedoch die *fromID* mit der *nodeID* verglichen, da nun die Schlussfolgerung gesucht wird. Sobald es eine Übereinstimmung gibt, wird der *toID* Wert wieder mit der Liste der S-Nodes abgeglichen, was zu zwei weiteren Vorgehen führt. Ist die *toID* nicht in der Liste der S-Nodes enthalten, wird ein neues JSON Objekt angelegt, in dem zum einen der zwischengespeicherte Wert der Prämisse unter dem Eintrag *premise* gespeichert wird, zum anderen die *toID* unter dem Eintrag *conclusion*. Außerdem wird im JSON Objekt noch ein Eintrag hinzugefügt, um festzuhalten, dass es sich um ein *premiseConclusion* Objekt handelt, damit später das richtige Argument gebildet wird. Weiterhin wird die S-Node ID in dem JSON Objekt gespeichert, da sie zur Identifikation des Argumentes in der Datenbank dient, und zusätzlich noch einen Eintrag *is_supportive*, welcher je nachdem ob es sich um einen

RA oder CA Knoten handelt, *true* oder *false* ist. Das erstellte Objekt wird dann einem JSON Array hinzugefügt und der Algorithmus betrachtet den nächsten Knoten.

Tritt allerdings der Fall ein, dass die *toID* in der Liste der S-Nodes enthalten ist, wird aus den gefunden IDs ein Argument Objekt erstellt, welches statt einer *conclusion_uid* einen *argument_uid* Fremdschlüssel enthält. Es wird wieder ein neues JSON Objekt angelegt, in welchem sowohl die *fromID* als Prämisse, als auch die *toID*, diesmal allerdings als *argument_uid*, gespeichert wird. Dazu wird jeweils wieder die S-Node ID und der Eintrag *is_supportive* gespeichert. Der Eintrag, dass es sich um ein *premiseConclusion* Objekt handelt, wird auf *false* gesetzt. Das ganze Objekt wird dann dem JSON Array hinzugefügt.

4.3.3 Einsortierung der Prämissen in Prämissengruppen

Nachdem alle Knoten betrachtet sind, müssen nun die Prämissen in Prämissengruppen gegliedert werden. Dabei stellt jeder S-Node eine Prämissengruppe dar, welche aus einer oder mehreren Prämissen besteht. Dazu werden diesmal alle Kanten betrachtet und dann für jede Kante wieder alle Knoten. Das Iterieren über die Kanten stellt sicher, dass auch alle Prämissen einer Prämissengruppe zugeordnet werden. Stimmt die *nodeID* eines Knoten mit der *toID* einer Kante überein, wird wieder überprüft, ob die *fromID* die ID eines I-Nodes und nicht eines S-Nodes ist, da im letzterem Fall keine Prämissengruppe angelegt wird. In einem neu angelegten JSON Objekt werden nun die *fromID* als *premise_uid* und die *nodeID* als *premisegroup_uid* gespeichert. Außerdem wird noch der *timestamp* des S-Nodes in das Objekt übernommen.

4.3.4 Entfernung doppelter Knoten

Problematisch wird es wenn man aus dem AIF eine Diskussion importieren möchte, welche von einem I-Node mehrere Kanten zu verschiedenen S-Nodes hat, welche aber wieder alle zu einem einzigen I-Node führen. Da dies logisch keinen Sinn macht und außerdem zu einer falschen Anzahl an Prämissengruppen führen würde, gibt es in der Schnittstelle die Option der Formatierung. Dabei werden bei der Erstellung der Prämissen Listen angelegt, in denen die gefundene *premisegroup_uid* und *premise_uid* festgehalten werden. Vor der Erstellung

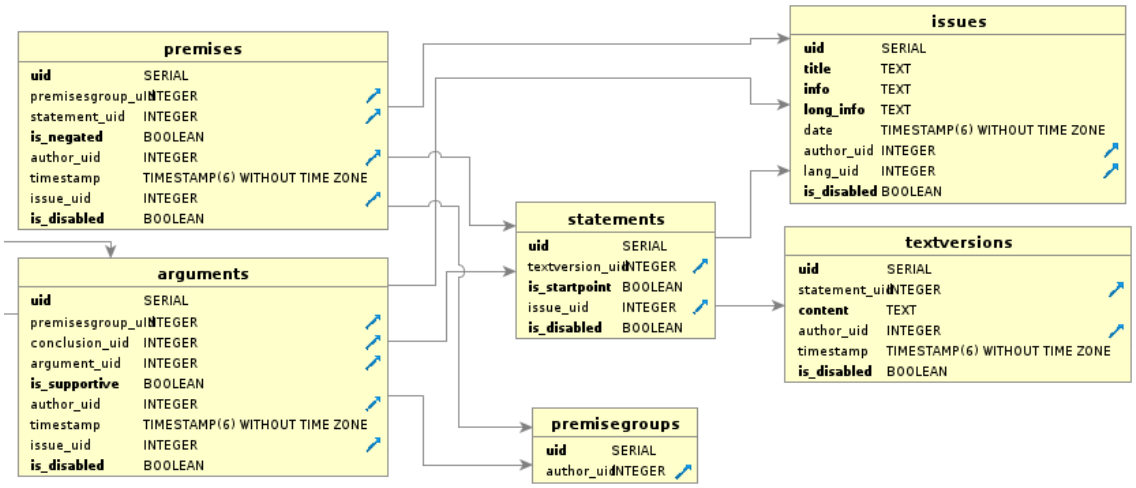

Abbildung 4.6: Für den import relevanten Datenbank Tabellen

einer neuen Prämissengruppe überprüft der Algorithmus zur Erstellung der Prämissengruppen, ob für die Prämisse schon eine Prämissengruppe existiert. Ist das der Fall, geht der Algorithmus einen Schritt weiter und betrachtet den nächsten Knoten. Andernfalls erstellt der Algorithmus eine neue Gruppe.

4.3.5 Einfügen der Daten in Tabellen der D-BAS Datenbank

Nachdem ein Array für die Verbindungen von Prämissen und Schlussfolgerung und ein weiteres Array für die Einsortierung der Prämissen in Prämissengruppen erstellt wurde, müssen die Daten in die Datenbank des D-BAS eingefügt werden. Damit die Diskussion im D-BAS korrekt angezeigt wird und auch funktionsfähig ist, müssen für die Tabellen *issues*, *textversions*, *statements*, *premises*, *premisegroups* und *arguments* Einträge erstellt werden. Dies wird in Abbildung 4.6 veranschaulicht.

Tabelle issues Zuerst wird der Eintrag für die Tabelle *issues* vorbereitet. Dabei wird die *uid* von der Datenbank automatisch erzeugt. Hierbei ist wichtig, dass die erzeugte *uid* gespeichert wird, da sie als Fremdschlüssel in anderen Tabellen noch benötigt wird. Alle anderen Einträge werden als Liste zwischengespeichert, welche sobald alle Einträge gesetzt sind, in die Datenbank eingefügt wird. Als *title* wird der Eintrag *text* des ersten I-node gewählt, da

es bei den Diskussionen im AIF kein wirkliches Thema gibt. Da jedoch meist nur ein Startpunkt existiert, beschreibt dieser in vielen Fällen angemessen das Thema der Diskussion. Als *info* und *long_info* wird der Text „imported from AIF“ als Platzhalter eingefügt. Der Eintrag *date* wird aus dem Eintrag *timestamp* des ersten I-node übernommen. Die *author_uid* wird auf 1 gesetzt, da hiermit auf den anonymen Benutzer in der Tabelle *users* verwiesen wird. Der Eintrag *lang_uid* wird auf 1 gesetzt, womit die Sprache des Themas auf Englisch gesetzt wird, und *is_disabled* wird auf false gesetzt. Danach wird aus der erstellten Liste ein neuer Eintrag in die Tabelle *issues* erzeugt.

Tabelle textversions Als nächstes werden aus den I-Nodes die Einträge für die Tabelle *textversions* erstellt. Dazu werden die einzelnen, zu erstellenden Einträge, in einer Liste zusammen gespeichert, welche dann, sobald der letzte Eintrag gesetzt ist, nach und nach in die Datenbank geschrieben werden. Die *uid* wird wieder automatisch generiert, muss jedoch auch wieder zwischengespeichert werden, da sie als Fremdschlüssel in der Tabelle *statements* noch benutzt wird. Der Eintrag *statement_uid* wird erstmal auf *null* initialisiert und wird nach der Erstellung der Einträge in die Tabelle *statements* eingesetzt. In *content* wird nun der Eintrag *text* der I-Nodes gespeichert. Die *author_uid* wird wieder auf 1 gesetzt, *timestamp* wird von den I-Nodes übernommen und *is_disabled* wird auf *false* gesetzt.

Tabelle statements Die Einträge in der Tabelle *statements* werden auch wieder über die I-Nodes erzeugt. Die erzeugten *uids* der Einträge müssen zwischengespeichert werden, da die Tabellen *premises*, *arguments* und *textversions* einen Fremdschlüssel auf diese haben. Dazu wird die von der Datenbank vergeben *uid* direkt in dem entsprechenden I-Node gespeichert. In *textversions_uid* werden nun die *uid*s der zuvor erstellten Einträge der Tabelle *textversions* gespeichert. Ob der Eintrag *is_startpoint true* oder *false* ist, hängt von dem entsprechenden Wert im I-Node ab, welcher zu Anfang gesetzt wurde. *Issue_uid* ist die *uid* des Themas und *is_disabled* wird wieder auf *false* initialisiert.

Tabelle premisegroups Nach der Erstellung der Einträge für die Tabelle *statements* werden die Einträge für die Tabelle *premisegroups* erstellt. Dazu wird die Liste der Verknüpfungen von Prämissen mit ihrer jeweiligen Prämissengruppe benötigt. Vorher muss sichergestellt werden, dass jede Prämissengruppe nur einmal eingefügt wird, da durch die automatische

Erzeugung der *uid* in der Datenbank sonst für jede Prämisse eine eigene Prämissengruppe erstellt würde. Dazu werden von allen betrachteten Objekten die Prämissengruppen vor dem Einfügen in die Datenbank gespeichert. Ist die Prämissengruppe vom nächsten Objekt noch nicht in der Liste vorhanden, wird ein neuer Eintrag erstellt, bei dem die *author_uid* wieder auf 1 gesetzt wird. Bei der Erstellung wird dann die ID der Prämissengruppe im Objekt durch die von der Datenbank vergebene ID ersetzt. Falls nun schon eine Prämissengruppe für ein Objekt erstellt wurde, wird nur der Wert der Prämissengruppe im Objekt mit dem entsprechenden Wert in der Datenbank ausgetauscht.

Tabelle premises Nachdem die Prämissengruppen erstellt sind, können nun die Einträge für die Tabelle *premises* erstellt werden. Dazu wird über das JSON Array iteriert, in dem die Prämissen mit ihrer jeweiligen Premissengruppen ID in der Datenbank stehen, welches zuvor bearbeitet wurde. Die *uid* wird wieder automatisch generiert und der Fremdschlüssel *premisesgroup_uid* wurde bei der Prämissengruppenerstellung direkt in jedem Objekt aus dem JSON Array gespeichert und kann übernommen werden. Um den Fremdschlüssel *statement_uid* zu setzten, wird das I-Node Array mit der ID der aktuellen Prämisse verglichen, da die IDs der Prämissen direkt von der *nodeID* der I-Nodes gebildet wurden. In dem passenden Knoten wurden bei der Erstellung der Einträge für die Tabelle *statements* die erzeugte *uid* gespeichert, welche nun den Eintrag *statement_uid* bildet. *Is_negated* und *is_disabled* bekommen beide den Wert *false* zugewiesen. *Issue_uid* wird auf die dem Thema entsprechende *uid* gesetzt, *author_uid* wieder auf den anonymen Benutzer mit dem Wert 1 und der *timestamp* wird aus dem Prämissenobjekt übernommen.

Tabelle arguments Als letzter Schritt müssen die Argumente erstellt werden. Dazu wird das Array, in dem die Prämissen mit ihren jeweiligen Schlussfolgerungen stehen, in die Tabelle *arguments* eingefügt. Der erste relevante Eintrag, nach der *uid*, ist die *premisesgroup_uid*. Um die *premisesgroup_uid* passend zu dem Eintrag in der Datenbank zu setzten, wird die aktuelle Prämisse wieder mit dem Array, in dem die Prämissen mit ihren jeweiligen Prämissengruppen stehen, verarbeitet. Nachdem die passende Prämissengruppe gesetzt ist, wird nun im weiteren Vorgehen unterschieden, ob aus dem Objekt ein Argument wird, welches eine Schlussfolgerung gesetzt hat, oder ein Argument, welches einen Fremdschlüssel auf ein anderes Argument hat.

Im ersterem Fall wird die in dem Objekt gespeicherte *conclusion* ID mit der Liste der I-Nodes verglichen, um so an die *statement_uid* in der Datenbank zu kommen. Diese ID wird dann als *conclusion_uid* gesetzt. Dazu wird noch der Eintrag *is_supportive* aus dem Objekt übertragen, welcher schon vorher gesetzt wurde. Die *author_uid* wird wieder auf den Wert 1 gesetzt und die *issue_uid* wieder auf die *uid* des Themas gesetzt. Der *timestamp* wird aus dem Prämissenobjekt übernommen und der Eintrag *is_disabled* wird auf *false* gesetzt. Die von der Datenbank automatisch generierte *uid* wird zusammen mit der gespeicherten S-Node ID, in einer separaten Liste gespeichert.

Dies ist nötig, um im Falle eines Objekts, bei dem statt der *conlusion* ID eine *argument* ID gesetzt ist, den Fremdschlüssel *argument_uid* passend zu dem entsprechendem Datenbankeintrag zu setzten. Auch hier wird wieder die S-Node ID und die generierte *uid* des neu eingefügten Argumentes in die Liste gespeichert, da dieses Argument auch wieder der Fremdschlüssel von einem anderen Argument sein kann.

4.3.6 Abschluss des Imports

Nachdem die erläuterten Schritte erfolgreich abgeschlossen sind, ist die Übersetzung vom AIF in das D-BAS vollständig abgeschlossen. Somit kann die Diskussion sowohl in der Datenbank bearbeitet, als auch im D-BAS benutzt werden.

4.4 Im- und Export Beispiel

Im folgenden wurde eine Diskussion aus dem D-BAS, zur Übersicht dargestellt mit der Graph-Ansicht Funktion des D-BAS in Abbildung 4.7, mit der Schnittstelle in die AIFdb übertragen, zu sehen in Abbildung 4.8. Danach wurde der Graph wieder in das D-BAS Format übersetzt und in die Datenbank des D-BAS eingefügt, zur Veranschaulichung wieder als Graph in Abbildung 4.9.

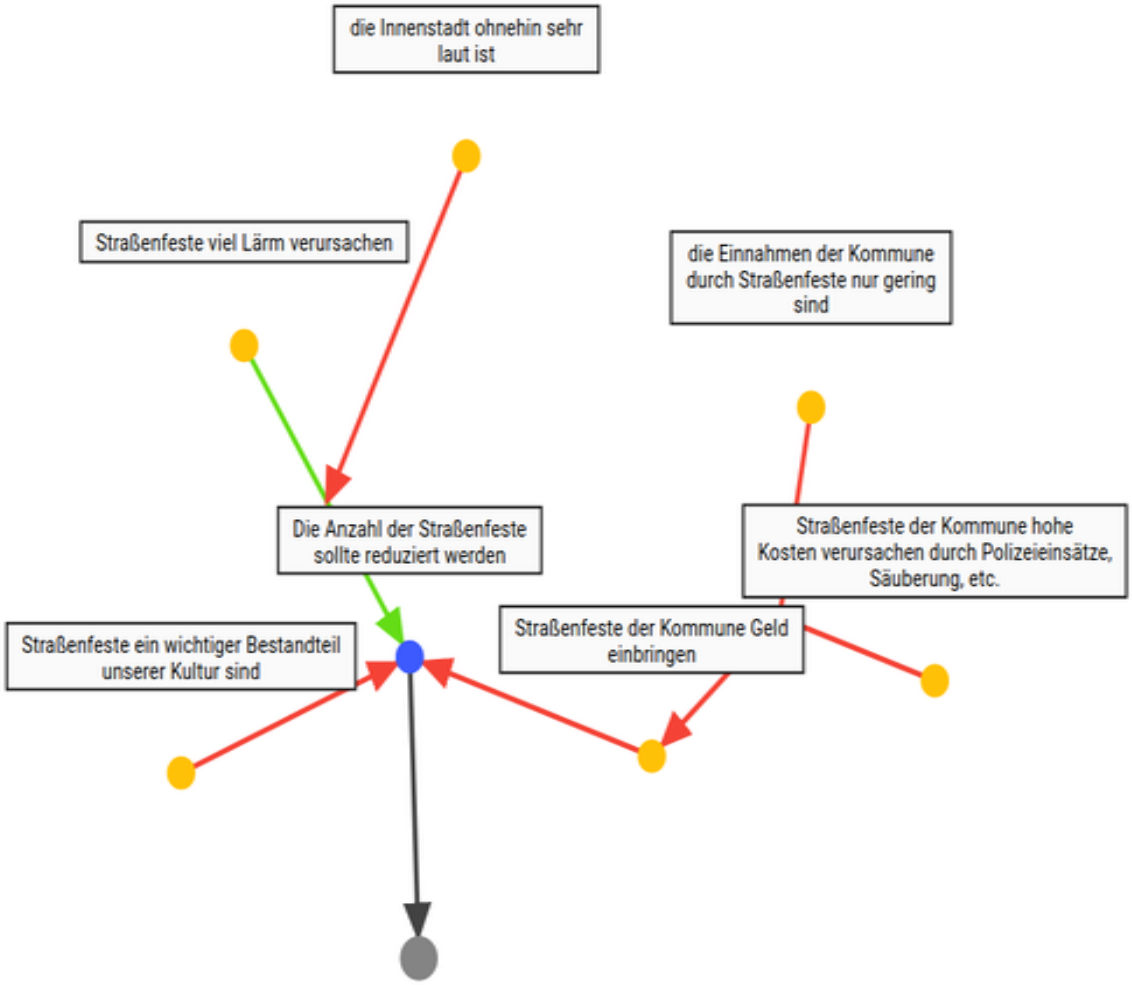

Abbildung 4.7: Diskussion im D-BAS dargestellt als Graph

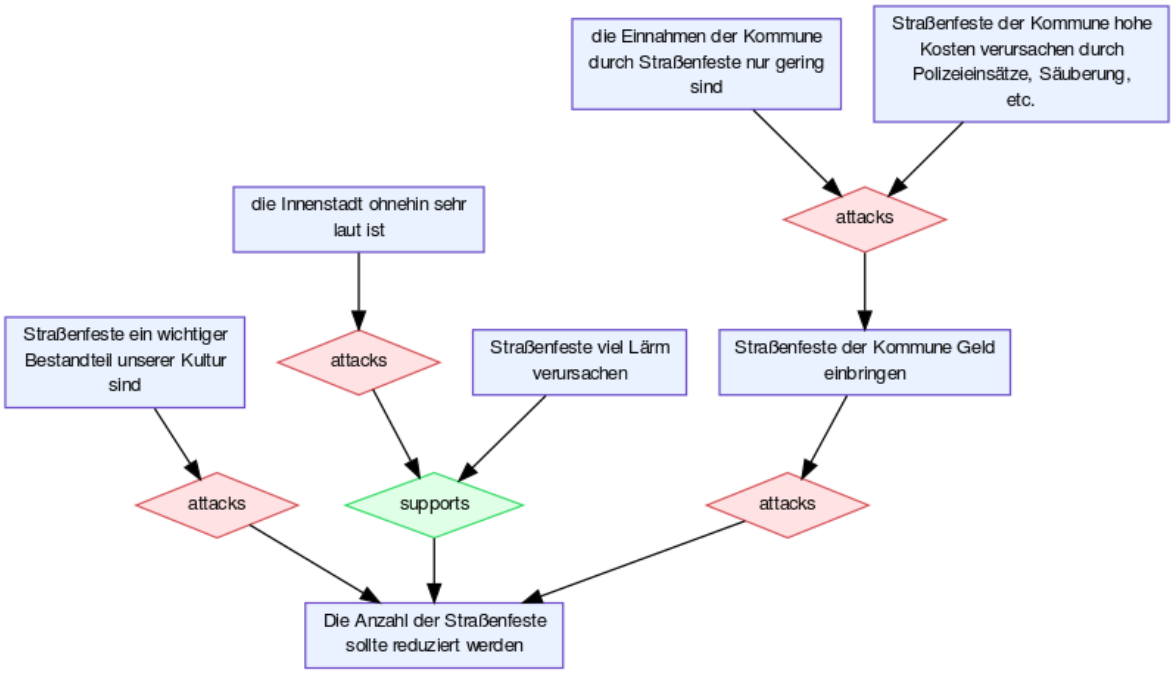

Abbildung 4.8: Mit der Schnittstelle aus dem D-BAS exportierte Diskussion

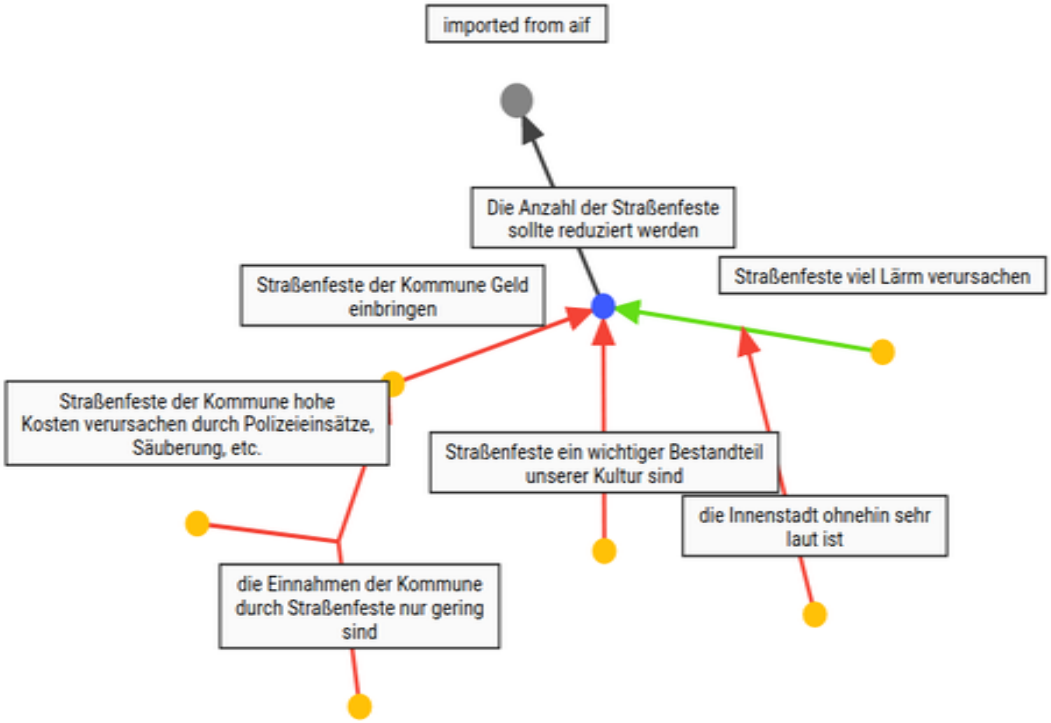

Abbildung 4.9: Aus dem AIF importierte Diskussion

Kapitel 5

Ausblick und Fazit

In der vorausgegangen Arbeit wurden die erforderlichen Schritte vorgestellt, mit deren Hilfe sich die Übertragung der wichtigsten Bestandteile einer Argumentation, sowohl aus dem D-BAS in das AIF, als auch aus dem AIF in das D-BAS realisieren lässt.

Für den Export wurden die Daten, die im D-BAS ein Argument bilden, in I-Nodes und S-Nodes übersetzt. Daraufhin wurden Kanten erstellt, um die erstellten I-Nodes, welche die Prämissen und Schlussfolgerungen darstellen, über einen S-Node zu verbinden. Außerdem wurden noch die Undercut Argumente übersetzt, wobei sich die Erstellung der Kanten unterscheidet. Nach der erfolgreichen Übersetzung wurde die Diskussion als JSON Objekt in die AIFdb hochgeladen und ist dort gespeichert und einsehbar.

Für den Import einer Argumentation aus dem AIF in das D-BAS wurden zunächst über die Schnittstelle der AIFdb ein JSON Objekt angefordert, welches die Argumentation enthält. Danach wurde das JSON Objekt, in dem die Knoten und Kanten des AIF Graphen enthalten sind, in S-Nodes, I-Nodes und Kanten unterteilt. Aus den I-Nodes wurden die Aussagen in der D-BAS Datenbank erstellt. Mit Hilfe der S-Nodes und Kanten wurden die verschieden I-Nodes in Prämissen, Schlussfolgerungen und schließlich in Argumente übersetzt. Dabei wurden die Daten nach ihrer Erstellung in die verschiedenen Tabellen der Datenbank des D-BAS eingefügt.

Die Schnittstelle führt zur einer Erweiterung der Reichweite des AIF. Alle Argumentationen, sowohl aus der AIFdb, als auch aus Carneades, Rationale oder anderen Formaten, die einen

Export in das AIF anbieten, können nun in das D-BAS Format transferiert werden. Durch die Einbindung des Dialog Systems des D-BAS, sind die Argumentationen leicht erweiterbar und einer breiten Masse an Benutzern zugänglich. Durch die Schnittstelle steigt aber auch die Reichweite der D-BAS Diskussionen. Nach dem Import in das AIF können die Diskussionen weiter exportiert werden, um beispielsweise in Rationale von Hand aus erweitert zu werden. Auch eine Analyse der Diskussion, um zum Beispiel die durchschnittliche Wort Anzahl pro Argument oder die Anzahl der unterstützenden Prämissen zu ermitteln, ist durch den Export in das AIF möglich.

In Zukunft kann die Schnittstelle noch als Ganzes in das D-BAS integriert werden, um Diskussionen während des Dialoges zu im- und exportieren. Für zukünftige Erweiterungen gilt es außerdem noch herauszufinden, wie man die Autoren einer Aussage aus dem D-BAS in das AIF übertragen kann und umgekehrt. Zwar ist es möglich einen neuen Benutzer im AIF zu registrieren, jedoch existiert keine Möglichkeit, die vom AIF generierte ID des neu erstellten Benutzers zurück zu bekommen. Es muss entweder von der AIFdb eine Möglichkeit geschaffen werden um die ID abzufragen oder man überlegt sich einen neuen Knotentyp, welcher mit den Aussagen verbunden ist und nur den Autor speichert.

Literaturverzeichnis

[BGLR12] BEX, Floris; GORDON, Thomas F.; LAWRENCE, John; REED, Chris: Interchanging arguments between Carneades and AIF. In: *COMMA*, 2012, S. 390–397.

[CMR+06] CHESÑEVAR, Carlos; MODGIL, Sanjay; RAHWAN, Iyad; REED, Chris; SIMARI, Guillermo; SOUTH, Matthew; VREESWIJK, Gerard; WILLMOTT, Steven u. a.: Towards an argument interchange format. In: *The Knowledge Engineering Review* 21 (2006), Nr. 4, S. 293–316.

[GPW07] GORDON, Thomas F.; PRAKKEN, Henry; WALTON, Douglas: The Carneades model of argument and burden of proof. In: *Artificial Intelligence* 171 (2007), Nr. 10, S. 875–896.

[KBBM16] KRAUTHOFF, Tobias; BAURMANN, Michael; BETZ, Gregor; MAUVE, Martin: Dialog-Based Online Argumentation. In: *Proceedings of the 2016 conference on Computational Models of Argument (COMMA 2016)* (2016). http://dx.doi.org/10.3233/978-1-61499-686-6-33. DOI 10.3233/978–1–61499–686–6–33.

[LBRS12] LAWRENCE, John; BEX, Floris; REED, Chris; SNAITH, Mark: AIFdb: Infrastructure for the Argument Web. In: *COMMA*, 2012, S. 515–516.

[VG07] VAN GELDER, Tim: The rationale for Rationale™. In: *Law, Probability & Risk* 6 (2007), Nr. 1-4, S. 23–42.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 23.August 2017 Emil Warkentin

Diese CD enthält:

- eine *pdf*-Version der vorliegenden Bachelorarbeit
- die LaTeX- und Grafik-Quelldateien der vorliegenden Bachelorarbeit samt aller verwendeten Skripte
- die Quelldateien der im Rahmen der Bachelorarbeit erstellten Software XYZ
- die Websites der verwendeten Internetquellen