TEIL II

IMPLEMENTIERUNG ZWEIER SYSTEMBEISPIELE

6 FALLSTUDIE I: WISSENSELABORATION DURCH PERSPEKTIVEN: DAS SYSTEM DISCBOARD

Im Folgenden wird beschrieben, wie das CardBoard-System (Hoppe, Gaßner et al., 2000; Gaßner, Tewissen et al., 1998) zu einem diskussionsunterstützenden System, dem DiscBoard, erweitert wird. Die Implementierungs- und Entwicklungsarbeiten zu diesem Fallbeispiel decken die folgenden Bereiche ab:

- Erweiterungen des CardBoards, um die Diskussionsunterstützung (DiscBoard) zu realisieren.
- Modellierung der Perspektiven.
- Entwicklung eines Diskussionsmodells, das den Informationsfluss zwischen den Perspektiven definiert.
- Die technische Realisierung der Abbildung für Karten.
- Die technische Realisierung der Abbildung für Strukturen.

6.1 Perspektiven

Bevor auf die technische Realisierung eingegangen wird, wird im Folgenden das Perspektivenkonzept eingeführt.

Perspektiven strukturieren Verstehensprozesse:

> *A perspective structures a person's cognitive process by:*
>
> - *the selection of those properties of the phenomenon that are being considered (and by implication of those that are ignored). In this way the perspective influences the choice of operative cognitions.*
> - *concepts and other cognitions that are being used in the interpretation of the selected properties.*
> - *the figurative and sometimes physical standpoints influencing the choice of position with respect to the situation or phenomenon in question.*
>
> *(Nygaard & Sørgaard, 1987, S.382)*

Der Einfluss, den eine Perspektive ausüben kann, entsteht danach durch die Auswahl von Eigenschaften oder Kriterien, unter denen ein Phänomen betrachtet wird, durch die

Begrifflichkeiten, die darauf angewendet werden sowie durch individuelle Überzeugungen und außerdem durch den situativen Kontext.
Wird ein Sachverhalt bewusst unter einer Perspektive betrachtet, so geschieht das, um ihn unter bestimmten Voraussetzungen oder Zielstellungen darzustellen, in einem Kontext zu interpretieren oder zu analysieren. Wird von einem Problem angenommen, dass es problemimmanent subjektiv und interessegeleitet gelöst wird und dass Lösungen selektiv, unvollständig, ausschnitthaft, fehlerhaft, interpretierend und widersprüchlich sein können, so kann die bewusste Verwendung von Perspektiven vor allem leisten, dass die jeweiligen Kriterien der Problemlösungen offen gelegt werden.
Nygaard und Sørgaard (1987) unterscheiden drei Arten von Perspektiven. Eine Perspektivenkategorie besteht in der bewussten oder unbewussten selektiven Beschreibung oder Wahrnehmung. Eine zweite Kategorie fasst solche Perspektiven zusammen, die die Umwelt oder Inhalte interpretieren. Darunter fallen Ansichten und individuelles Verstehen, denn in beiden Fällen wird subjektives und unvollständiges Wissen als Grundlage herangezogen, also auf Basis eines Vorverständnisses neue Gegebenheiten interpretiert. Als Drittes existiert die Perspektive als Standpunkt, dies auch im Sinne eines physikalischen Standpunktes.
Stahl (2001) prägt den Begriff der „computational perspective", unter der eine Auswahl von Daten aus einer Menge verstanden wird, bei der es sich damit vor allem um eine kriteriengeleitete, filternde Sicht auf Daten handelt. Unter diese Definition fallen beispielsweise Datenbankviews (Dittrich, 1997). Eine Datenbanksicht ist eine abstrakte Relation, die auf den Basisrelationen oder auf anderen Sichten aufbaut. Mit diesen dynamischen „Fenstern" können komplizierte oder sich wiederholende Anfragen vereinfacht werden. Die Sichten wirken nicht interpretierend, sie stellen nur Ausschnitte der in Relationen verbundenen Daten kompakt dar. Auch Stahl (2001) stellt mit dem System WebGuide einen Perspektiven-Ansatz vor. Dort werden Ansichten von Personen oder Gruppen als Perspektiven hierarchisch verwaltet. Perspektiven sind in diesem System Meinungen zu einem Thema. Diese Meinungen können anderen Personen oder Gruppen zugeordnet werden. Mit so einer Zuordnung werden Mengen von Aussagen automatisch in die neue Perspektive übernommen.

6.1.1 „Epistemic forms" als Perspektiven

In dieser Arbeit wird ein Perspektiven-Ansatz gewählt, der die konstruktiven und interpretierenden Aspekte mit einem filternden Herangehen verbindet. Die Grundlage dafür bilden epistemische Formen (Collins & Ferguson, 1993), die im Folgenden erläutert

werden. Nach Collins und Ferguson (1993) sind epistemische Formen Zielstrukturen, die eine wissenschaftliche (Nach-)Forschung bzw. Untersuchung begleiten.

> *We refer to the target structures that guide scientific inquiry as epistemic forms and the set of rules and strategies that guide inquiry as epistemic games.*
> *(Collins & Ferguson, 1993, S.25)*
> *The forms and games we describe are epistemic in that they involve the construction of new knowledge. They are played to make sense of phenomena in the world.*
> *(Collins & Ferguson, 1993, S.26)*

Die epistemischen Spiele werden insofern von Theorien und Modellen abgegrenzt, als dass sie allgemeiner sind. Theorien und Modelle greifen hingegen oft auf epistemische Formen zurück und setzten sie in einer ganz konkreten Weise ein.

> *They* (Anmerkung: epistemic games) *are not simply inquiry strategies or methods; rather they involve a complex of rules, strategies, and moves associated with particular representations (i.e. epistemic forms).*
> *(Collins & Ferguson, 1993, S.26)*

Als ganz grundlegende epistemische Spiele wird das Erstellen von Listen angeführt, Vergleiche, die Ermittlung von Prozessschritten und die Analyse von Trends. Collins und Ferguson (1993) führen u.a. das sehr einfache Spiel des Anlegens von Listen aus: Die Erstellung einer Liste ist immer an eine Frage gebunden, die auf eine Menge von Antworten zielt, wie „Welche Vorteile hat die Französische Revolution für Europa gebracht?“. Für dieses Spiel bestehen typische Constraints auf den Ergebnismengen: Wert wird auf die Abdeckung gelegt, also darauf, dass die Antwort möglichst viele Aspekte enthält, auf die Unterschiedlichkeit, dass sich die Antworten nicht überdecken, die Vielzahl, denn es soll nicht nur eine Antwort in der Liste enthalten sein, auf die Kürze, denn die Antworten sollen möglichst prägnant sein und auf die Gleichartigkeit der Antworten z.B. in Bezug auf den Allgemeinheitsgrad.

Collins und Ferguson (1993) unterscheiden drei Typen von epistemischen Spielen:

- *Strukturelle Analysen.* Darin wird die „Natur“ eines Phänomens hinterfragt. Der zu untersuchende Gegenstand wird in seine Teile oder auch konstituierenden Elemente zerlegt, die beschrieben und ins Verhältnis gesetzt werden. Collins und Ferguson (1993) nennen dafür u.a. folgende Beispiele: Die räumlich Dekomposition, bei der die topologische Relation von Dingen zueinander geklärt wird, Zustandsmodelle, Vergleichen und Gegenüberstellen beispielsweise mit dem Spezialfall der Kosten/Nutzen-Analyse, die Suche nach „Primitiven“ wie den chemischen Elementen oder grundlegenden Beschreibungssprachen in der KI oder auch die einfache Hierarchisierung. Ebenfalls in diese Kategorie werden Axiomensysteme eingeordnet, wie die Peano-Axiome und die Euklid’schen Axiome in der Geometrie.

- *Funktionale Analyse.* Bei der funktionalen Analyse besteht das Ziel darin, die kausale oder funktionale Struktur von Elementen eines Systems zu beschreiben. Ein Beispiel dafür sind „Ursache/Folge"-Analysen, worin kritische Ereignisse herausgegriffen werden, um notwendige Bedingungen oder mögliche Folgen zu spezifizieren (Collins & Ferguson, 1993). Auch UND/ODER-Graphen können für die funktionale Analyse verwendet werden.
- *Prozessanalysen.* Bei der Prozessanalyse soll das dynamische Verhalten von Phänomenen analysiert werden. Beispiele sind „system dynamics"-Modelle und Beschreibungsmodelle über Constraints (Collins & Ferguson, 1993).

Diese drei Kategorien unterscheiden sich in ihrer Zielstellung, jedoch nicht unbedingt in den Mitteln, die für die Analysen eingesetzt werden. Grundlegende Darstellungen mit speziellen Tabellen oder Graphen finden sich in jeder davon.

Collins und Ferguson (1993) vertreten in ihrer Arbeit schließlich u.a. die folgenden Thesen:

> *Epistemic forms are generative frameworks with slots and constraints on filling those slots. In this respect they are like a grammar that can be expanded to fit the degree of complexity of the phenomena being analyzed. The slots can be cells in a matrix, variables in a model ,or types of curves and their parameters.*
>
> *Epistemic forms and games serve to guide inquiry by directing the inquirer as to which slots to fill and which constraints to meet in filling those slots.*
>
> *Scientific and historical theories often reflect the forms that generated them, but they usually combines a number of different forms in complex interrelationships.*
>
> *(Collins & Ferguson, 1993, S.40)*

„Epistemic games" stellen in den meisten Fällen genauso ein analytisches wie ein konstruktives Vorgehen dar. Der Unterschied zwischen dem epistemischen Spiel und der Form ist, dass beim Spiel die Regeln, Züge oder Strategien hinzukommen. Collins und Ferguson (1993) vergleichen den Unterschied mit einem Spielfeld wie bei Tic-Tac-Toe und dem Spiel selbst, durch das bestimmt wird, wie mit den Feldern umgegangen werden soll.

In dieser Arbeit stehen die „epistemic forms" Pate für den Perspektiven-Ansatz. Wie mit den Darstellungsmitteln umgegangen wird, wird dabei weitgehend den Nutzerinnen und Nutzern überlassen. Gerade dadurch soll eine sehr allgemeine Verwendungsmöglichkeit aufrecht erhalten werden.

Allerdings beschränken sich die visuellen Sprachen nicht ausschließlich auf die Formen, in denen etwas dargestellt wird. Durch die Symbole, die die Sprachen anbieten, werden bestimmte Verwendungen nahe gelegt, also wird im Sinne von Collins und Ferguson (1993) bereits ein „epistemic game" angedeutet. Beispielsweise wird eine Sprache angeboten, die später als Brainstormingsprache benannt wird, die sich an der gIBIS-

Darstellung (Conklin & Begemann, 1987) orientiert. Darin können zwar im Prinzip UND/ODER-Graphen erstellt werden, durch die Symbolik wird aber nahe gelegt, dass damit Optionen bzw. Lösungen zu Fragen oder Problemen aufgeschlüsselt werden. Eine Funktionale Analyse, zu der UND/ODER-Graphen bei Collins und Ferguson (1993) zugeordnet werden, wäre in der vorgegebenen Symbolik nicht plausibel.

6.1.2 Modellierung mit „epistemic games“

Perspektiven können für solche Probleme klärend eingesetzt werden, wo Lösungen zu komplex sind, um konsistent und vollständig erfasst zu werden. Insbesondere sind sie sinnvoll, wenn das Ergebnis offen ist, wie in Design-, Modellierungs- und Konstruktionsprozessen.

Seit einigen Jahren haben Perspektiven-Ansätze auch in der Softwareentwicklung Einzug gehalten. Insbesondere im Ansatz partizipativen Software-Engineerings wird von interessegeleiteten Zielen bei der Entwicklung ausgegangen, die schon von Beginn eines Entwicklungsprozesses mit berücksichtigt werden sollen. In diesem Prozess liefert beispielsweise die Anforderungsanalyse einen Erkenntnisgewinn über die Prozesse, die durch die geplante Software unterstützt werden sollen. Als Resultat aus den Analysen wird durch die neue Software oft ein veränderter Arbeitsablauf etabliert. Ungeeignete Anforderungsanalysen führen deshalb immer wieder zum Scheitern von Softwareprojekten. Perspektiven sollen die unterschiedlichen Interessen transparent machen, um so konstruktiv mit ihnen umgehen zu können.

- *Reference Model of Open Distributed Processing (ODP).* Das “Reference Model of Open Distributed Processing (ODP)” ([ODP]) ist ein Standard für die Entwicklung offener, verteilter Systeme, mit dem gerade der geschilderten Problematik Rechnung getragen werden soll. ODP bietet als zentrales Element einen Modellierungsansatz, der fünf verschiedene Perspektiven („Viewpoints”) anbietet: der Enterprise, Information, Computational, Engineering und Technology „Viewpoint“. Jede dieser Sichten bietet eine eigene Menge von Begriffen und graphischer Beschreibungsmöglichkeiten an. Es ergeben sich fünf unterschiedliche Modelle eines Systems, jedes ist ein Teil der Gesamtspezifikation. Die Systembeschreibung soll damit vereinfacht werden, da davon ausgegangen wird, dass eine vollständige Spezifikation nicht möglich ist. Diese wird letztlich erst durch das System selbst realisiert und liegt entsprechend nur implizit vor. Der Zusammenhang zwischen den Perspektiven wird dadurch hergestellt, das die Objekte der einzelnen „Viewpoints“ miteinander korrespondieren.

- *Unified Modelling Language (UML).* Die Unified Modelling Language (UML) (Jacobson, Booch & Rumbaugh, 1999; Quatrani, 2000) bietet nach dem hier vertretenen Verständnis ebenfalls Perspektiven für die Modellierung von Systemen an. Die Perspektiven entsprechen unterschiedlichen Diagrammtypen, die verschiedene graphische Sprachen anbieten, die jeweils für bestimmte Problemdarstellungen geeignet sind, für die sie Mengen von Beschreibungskonzepten anbieten:
  - „Use Case"-Diagramme greifen typische Nutzungsszenarien, Systemverwendungen und Zielstellungen auf. Mit ihnen können vor allem Anforderungen dargestellt werden.
  - Mit „Activity"-Diagrammen können Workflows und parallele Prozesse dargestellt werden. Sie beziehen sich nicht unbedingt auf Objekte, weshalb Aktionen, die auf Objekten durchgeführt werden, nur schlecht damit ausgedrückt werden können. Gut geeignet sind sie wiederum für die Analyse von „Use Cases".
  - UML bietet auch informationsredundante Diagrammtypen an – z.B. die Gruppe der Interaktionsdiagramme - die aufgrund ihrer unterschiedlichen Art der Visualisierung bewusst zu einem anderen Zweck eingesetzt werden können. Typischerweise schlüsselt ein Interaktionsdiagramm einen konkreten „Use Case" auf. Es verbindet ausgesuchte Objekte und Nachrichten, die zwischen diesen Objekten fließen. Beispiele für Interaktionsdiagramme sind „Sequence"- und „Collaboration"-Diagramme. Im „Sequence"-Diagramm ist es leichter, die Reihenfolge von Ereignissen zu erkennen. Sie sind geeignet wenn wenige Objekte mit einem umfangreichen Nachrichtenfluss dargestellt werden. „Collaboration"-Diagramme zeigen eher die statische Verbindung zwischen den Objekten. Sie sind deshalb besser geeignet, wenn viele Objekte relativ wenig interagieren.
  - „State"-Diagramme sollten nur Klassen verwendet werden, bei denen das Verhalten stark zustandsabhängig ist. Oft sind sie für UI-Kontrollobjekte gut einsetzbar.
  - Am bekanntesten sind vermutlich die Klassendiagramme, die schon sehr eng mit der Implementierung verwoben sind. Die Klassendiagramme können direkt zur Spezifikation von Klassen mit Elementen und Methoden sowie für die Darstellung der Vererbungen verwendet werden.

Die Diagramme greifen ineinander, indem gleiche Objekte wieder aufgenommen und in einen anderen Kontext gestellt werden können. Die Idee besteht nicht in der vollständigen Spezifikation eines Systems, sondern in der exemplarischen Darstellung ausgewählter Fälle (Fowler & Scott, 2000). Das gilt auch für die Klassendiagramme. Relevant sind typische Fälle oder besonders schwierige. Dies wird auch als "Use Case driven development" bezeichnet (Jacobson, Booch & Rumbaugh, 1999; Fowler & Scott, 2000; Quatrani, 2000).

UML wird als Modellierungssprache und nicht als Methode gesehen (Fowler & Scott, 2000), d.h. es werden explizit keine konkreten Prozesse definiert. Einen Ansatz für die Prozessgestaltung bietet der „Rational Unified Process (RUP)" (Quatrani, 2000), der zwar die Darstellungsmethoden von UML verwendet aber umgekehrt nicht durch UML präjudiziert wird.

Anzumerken ist, dass diese Diagrammtypen von den UML-Entwicklern nicht als Perspektiven bezeichnet werden. Dort werden die Designebenen als Perspektiven bezeichnet, die die jeweilige Zielstellung einer Modellierung beschreiben: die konzeptuelle Perspektive, die Spezifikationsperspektive oder die Implementierungsperspektive. Im Gegensatz zu Diagrammen als Perspektiven, die die Beschreibungskonzepte in den Vordergrund stellen, stellen die Designebenen die Modellierungsziele vor.

Auch in Diskussionen finden Prozesse der Wissenskonstruktion statt. Dort muss man ganz besonders von einer selektiven Wahrnehmung durch die „Gruppenkultur" ausgehen. Gerade eine unbefriedigende Wissensgrundlage kann Beweggrund für eine Diskussion sein. Generell können Inhalte nur in Ausschnitten während einer Diskussion aufgearbeitet werden. Trotzdem besteht natürlich der Wunsch, möglichst viele relevante Aspekte aufzugreifen. Deshalb kann sich der Einsatz von Perspektiven positiv auswirken, nämlich um Inhalte besser zu verstehen und ihre Vollständigkeit zu hinterfragen.

6.1.3 Perspektivenwechsel

Die Idee der Perspektive legt es nahe, Perspektivenwechsel zielgerichtet einzusetzen. Jonassen und Carr (2000) argumentieren, dass unterschiedliche Perspektiven das Verständnis eines Sachverhaltes verbessern können. Danach wird ein Problem oder ein Gegenstand besser in einen Kontext gestellt, wenn er explizit auf unterschiedliche Weise repräsentiert wird. So wird eher der Problemzusammenhang begriffen und nicht nur einzelne Details. Jonassen und Carr (2000) nennen diesen bewussten Einsatz unterschiedlicher Repräsentationsmittel „active learning strategies". Sie lassen jedoch die

Frage, in welchem Verhältnis diese Darstellungsformen zueinander stehen und wie diese Perspektivenwechsel angeleitet werden können, weitgehend offen. Auch Suthers (1999A, 1999B, 1999C) hebt die notwendige Passung einer Aufgabe zu einer bestimmten Darstellung und Umgangsmethoden hervor, die er als „representational bias“ bzw. „representational guidance“ beschreibt. Er sieht die Notwendigkeit, erst einmal relevante Eigenschaften von Darstellungen zu konkretisieren, um eine solche Zuordnung zu Aufgaben leisten zu können. Collins und Ferguson (1993) erläutern zwar, dass es Übergangsbedingungen zwischen den „epistemic games“ gibt, gehen jedoch nicht darauf ein, wie diese aussehen können.

Allerdings versuchen manche Systemansätze, die epistemische Formen einsetzen, gerade einen Arbeitsablauf dadurch zu unterstützen, dass epistemische Formen oder Spiele als Bestandteile eines Prozesses implementiert werden. Dies soll hier als *epistemischer Prozess* eingeführt werden.

Ein bereits in Kapitel 3.2 beschriebenes Beispiel ist das Sepia-System (Streitz, Haake, et al., 1992; Streitz, Haake, et al., 1998), in dem vier visuelle Sprachen („epistemic games“) Autorinnen und Autoren kognitiv bei der Erstellung von Hypertexten unterstützen sollen. Aus der Untersuchung von kooperativen Schreibprozessen wurden dafür typische Arbeitsschritte abgeleitet, die jeweils durch die visuellen Sprachen angeleitet werden. Die Prozessunterstützung resultiert letztlich aus der Abstimmung der Darstellungsweisen, die mit dem gewohnten Schreibprozess harmonieren sollen.

Eine Grundlage für die Unterstützung epistemischer Prozesse wird in Scardamalia, Bereiter und Steinbach (1984) gelegt. Deren Ausgangspunkt besteht in der Anleitung von Reflexionsprozessen beim Schreiben. Sie widersprechen der Annahme, dass der „innere Dialog“ mit einem gedachten Leser oder einer Leserin ein geeignetes Modell zur Beschreibung der Reflexion darstellt, die während des Schreibens stattfinden sollte. Stattdessen schlagen sie ein Konzept vor, in dem der Wechsel zwischen zwei Problemräumen, dem „content space“ und dem „rhetorical space“, herangezogen wird, um Phänomene beim Schreiben von Texten zu interpretieren. Meinungen, Fakten, Erklärungen, etc., die im „content space“ vernetzt sind, werden in den „rhetorical space“ übernommen, um sie dort in einen linearen Text zu überführen. Fragen, die sich dort aus der Strukturierung ergeben, werden als Unterziele zurück in den „content space“ übernommen. Scardamalia, Bereiter und Steinbach (1984) behaupten, dass in dieser Teilung der beiden Problemräume die Essenz der Reflexion beim Schreiben liegt.

Carlson (1995) nimmt mit dem Broca-Ansatz, worin allgemeine Schreibprozesse durch die Trennung von inhaltlichen und rhetorischen Aspekten unterstützt werden sollen, auf

den Ansatz von Scardamalia, Bereiter und Steinbach (1984) Bezug. Dafür werden sequentiell verschiedene „facilitator“ (hier interpretiert als „epistemic games“) durchlaufen: Der „Cluster-Browser“ soll das Explorieren von Daten ermöglichen. Jeweils zusammenhängende Graphen bilden die thematischen Cluster. In der „Konzept-Synthese“ sollen die vorher gefundenen Cluster weiter elaboriert werden. Dafür werden Notizkarten angelegt, die Informationsquellen miteinander verbinden und gleichzeitig in eine bestimmte Reihenfolge gebracht werden können. Mit dem „Information Threader“ wird dann versucht, zentrale Konzepte mit semantischen Netzen zueinander ins Verhältnis zu setzten. Die bis hier durchlaufen Phasen behandeln die Daten- bzw. Inhaltsebene des zu erstellenden Textes. Für die Rhetorik werden drei weitere Arbeitsschritte ausgeführt: In einem hierarchischen Planer werden die Textbestandteile von einer Graphstruktur in eine Baumstruktur umgewandelt, um eine lineare Struktur, wie sie für normale Texte verlangt wird, anzustreben. In der nächsten Phase wird die rhetorische Feinstruktur entworfen. In der letzten Phase wird noch eine Revisionsmöglichkeit angeboten.

In den Grundlagen von Broca wird ausführlich für einen Ansatz argumentiert, der verschiedene Arbeitsphasen durch verschiedene Arbeitsmethoden unterstützt. Leider bleibt unklar, inwieweit dieser Ansatz auch als Werkzeug umgesetzt wurde und ob dabei auch schon Erfahrungen dokumentiert wurden. Der technische Ansatz soll auf Objekten basieren, die durch die unterschiedlichen Phasen verändert werden.

Werden in einer Computeranwendung Perspektiven definiert, so beinhaltet das zwingend eine algorithmische Umsetzung davon, wie entweder Daten ausgewählt werden oder in welcher gegenseitigen Abhängigkeit die Perspektiven stehen sowie eine Unterstützung bei der Darstellung der Perspektiven. Die folgende Aufzählung erläutert mögliche Ausprägungen von Abhängigkeiten, die zwischen Perspektiven bestehen können:

- *Filter.* Durch einen Filter kann eine Teilmenge der Inhalte aus einer oder mehreren Perspektiven oder der Datenbasis in einer anderen Perspektive dargestellt werden. Filter sind sinnvoll, wenn auf bestimmte Aspekte fokussiert werden soll. Werden Teile von Inhalten aus mehreren Perspektiven gefiltert, so kann das als Zusammenfassung charakterisiert werden. Damit ist eine Schnittmenge gemäß korrespondierender Eigenschaften gemeint, keine Vereinigungsmenge.
- *Spezialisierung und Abstraktion.* Spezialisierungen können strukturell oder thematisch motiviert sein. Charakteristisch ist, dass ein Objekt, Element oder Teilstruktur aus einer Perspektive herausgenommen wird, um dies in einer anderen Perspektive weiter auszuarbeiten.

Werden Perspektiven durch Vererbungen gewonnen handelt es sich ebenfalls um Spezialisierungen. Die Vererbung basiert allerdings auf internen Hierarchien, die nicht unbedingt aus den externen Darstellungen erkennbar sind.

Eine Abstraktion kann als Gruppierung von Elementen zu einem neuen übergeordneten Objekt verstanden werden. Thematisch motivierte Abstraktionen können z.B. Clusterverfahren sein. Die Bedeutung des Abstraktums bliebe in diesem Fall für das System an die Extension des Clusters gebunden.

- *Sortierungen.* Sortierungen strukturieren ein bestehendes Diagramm um. Sie sind deshalb nicht unbedingt als eigenständige Perspektiven zu verstehen. Trotzdem beeinflussen die Darstellungsformen maßgeblich den Umgang mit ihnen. Sortierungen können sinnvoll mit anderen Perspektivenänderungen kombiniert werden wie z.B. Filter oder Abbildungen. Alle automatisch generierten Perspektiven definieren zwingend auch das Layout der Darstellung und damit die Sortierung.
- *Abbildungen.* Unter Abbildungen kann einerseits die homomorphe oder isomorphe Abbildung von Graphen bzw. Teilgraphen verstanden werden. Abbildungen können andererseits aber auch auf Interpretationsregeln beruhen, die nicht mehr die Struktur erhalten, auch wenn die Basis der Interpretation Strukturmuster sind. Eine solche Interpretation von Strukturen kann auch zur Generierung von Inhalten führen, die in neuen Objekten einer anderen Perspektive visualisiert werden.

Das DiscBoard setzt einen Perspektivenansatz um, der Filter, Abbildungen, Sortierungen und Abstraktionen integriert. Die einzelnen visuellen Sprachen stellen epistemische Formen zur Verfügung, wenn auch durch die Vorgabe von Symbolen bestimmte Verwendungen im Sinn von epistemischen Spielen nahe gelegt werden. Die Kombination verschiedener visueller Sprachen leitet einen epistemischen Prozess an, der aber nicht als Schreibprozess zu verstehen ist, sondern als Prozess der Wissenselaboration, der sich also im „content space“ (Scardamalia, Bereiter & Steinbach, 1984) bewegt.

Durch die Perspektiven soll der Diskussionsgegenstand unter verschiedenen Zielstellungen betrachtet werden können. Es wird deshalb insbesondere keine Konsistenz zwischen den Perspektiven verlangt. Es wird davon ausgegangen, dass sich die jeweiligen Inhalte der Perspektiven zwar aufeinander begründen und auseinander hervorgehen, dass sie sich dann aber auch additiv ergänzen. Das reflektiert die Idee, dass ein Diskussionsgegenstand meistens viel zu umfangreich ist, um ihn vollständig zu erarbeiten. Vielmehr soll er sporadisch für einige relevante Aspekte aufgearbeitet werden.

Zwischen den Perspektiven im DiscBoard wurden Abhängigkeiten implementiert. Durch das Angebot von Perspektiven, die mit einer spezifischen Auswahl bestehender Daten

automatisch gefüllt werden, sollen Diskussionsprozesse angeregt und unterstützt werden. Mit dem DiscBoard wird damit ein pragmatischer Ansatz zur Strukturierung von Gesprächsinhalten implementiert.

6.2 Erweiterungen des CardBoards

6.2.1 „Joint“ Workspaces

Aufgrund des bereits vorgestellten Diskussionsszenarios, dass als „face-to-face“-Szenario eingeführt wurde, gehe ich davon aus, dass in der Gruppe entweder über eine Projektionswand von allen Beteiligten in einer gemeinsamen Applikation gearbeitet wird oder dass „joint“ Workspaces verwendet werden. Diese dienen dann einerseits der komfortablen Eingabe vom Platz, um das Notierte unmittelbar an der Projektionsfläche sichtbar zu machen. Andererseits liegt der Vorteil eines Settings mit „joint“ Workspaces, in dem allen Beteiligten eine eigene Applikation zur Verfügung steht, in der individuellen Kombinationsmöglichkeit privater und öffentlicher Eingaben.

Ob auch „shared“ Workspaces erwünscht sind, hängt nach meiner Überzeugung von der Art des Gespräches und der Aufgabenstellung ab. Stellt man sich eine politische Debatte vor, in der Absprachen zwischen verschiedenen Beteiligten während der Diskussion getroffen werden könnten, wären Zwiegespräche u.U. sinnvoll. In weniger taktischen, sondern mehr konsensorientierten Sitzungen, beispielsweise in Projektgruppensitzungen, sind geheime Bestandteile weniger sinnvoll oder auch unerwünscht. Für die Aufwertung der „joint“ Workspaces gegenüber den „shared“ Workspaces bei der Diskussion spricht auch, dass die Koordination von möglicherweise mehreren schriftlichen Diskussionssträngen sowie der verbalen Kommunikation einschließlich der jeweiligen Interaktion zwischen Personen und mit dem System ausgesprochen kompliziert ist. Dass diese Situation, in der die Interaktion mit dem System mit dem Gespräch koordiniert werden muss, auch ohne parallele schriftliche Gesprächsstränge schwierig ist, konnte schon in der Vorstudie beobachtet werden (siehe Kapitel 4).

Konzeptuell ist für jeden Workspace-Typ, anders ausgedrückt für jede visuelle Sprache, nur ein „joint“ Workspace vorgesehen, um Verwirrungen zu vermeiden. An diesen Workspace kann sich jede Applikation anschließen und diesen auch wieder verlassen. „Joint“ Workspaces erlauben den Zugriff einer beliebigen Anzahl von Diskussionsbeteiligten auf den gleichen Datenraum. Die „joint“ Workspaces präsentieren sich damit den Beteiligten als Arbeitsbereiche, die immer zugreifbar sind und die den aktuellen Diskussionsstand zeigen. In diesem Konzept ist es möglich, dass nur eine Person in einem

„joint“ Workspace arbeitet. Das ist dann die erste Person in diesem Arbeitsbereich, die damit ihre Bereitschaft zur Kommunikation kundtut und dadurch für andere erreichbar wird. Das „Joinen“ stellt sich für die Beteiligten nicht direkt als Aufnahme eines Kommunikationsprozesses dar, sondern eher als Zugriff auf eine virtuelle Tafel auf der auch andere arbeiten könnten.

Dieses Konzept widerspricht in gewisser Weise, der im MatchMaker implementierten Idee der Synchronisation. MatchMaker stellt als Basis für die Synchronisation gerade keine zentralen „Datenobjekte“ zur Verfügung, die einer solchen virtuellen Tafel entsprechen könnten. Synchronisiert werden die Userinterface-Events. Der Zustand eines „joint“ Workspaces wird also nicht direkt im MatchMaker verwaltet, sondern liegt nur in der Applikation selbst als Information vor.

Für das skizzierte „join“-Konzept muss deshalb das Problem gelöst werden, dass eine beliebige Applikation, die ein „join“ ausführt, eine beliebige andere finden muss, von der die relevanten Daten übernommen werden können. Dieser Instanzierungsprozess darf selbst noch nicht im synchronisierten Modus stattfinden.

Jede Applikation verwaltet dafür eine Liste ihrer eigenen „joint“ Workspaces. Löst eine Applikation ein „join“ für einen bestimmten Workspace-Typ aus, so wird eine beliebige andere Applikation gesucht, die bereits einen „joint“ Workspace dieses Typs generiert hat. Mit diesem wird dann die Kopplung durchgeführt. Ein neuer „joint“ Workspace wird erst generiert, wenn in allen anderen Applikationen keine geeignete Partner-Applikation zum Koppeln gefunden wird. Auf diese Weise wird sichergestellt, dass tatsächlich nur ein „joint“ Workspace zu einem Workspace-Typ existiert. Nachdem eine geeignete Partner-Applikation gefunden wurde, wird der neue Workspace mit den bereits vorliegenden Daten initialisiert. Wird ein „joint“ Workspace geschlossen, so besteht die Verbindung für die anderen Beteiligten weiter. Dieser Prozess des „join“ ist in Abbildung 27 dargestellt.

Links sind zwei Applikationen in Form ihrer aktuellen Benutzeroberfläche symbolisiert. In Applikation 1 (oben links) sind zwei „joint“ Workspaces geöffnet, die intern in einer Liste verwaltet werden, die in der Abbildung ebenfalls dargestellt ist. In der unteren Applikation sind drei „joint“ Workspaces geöffnet. Auf der rechten Seite befindet sich eine dritte Applikation in drei Varianten, die sich aus einer zeitlichen Entwicklung ergeben. Diese Applikation hat ein „join“ für den in der Abbildung blau dargestellten Workspace ausgelöst. Dafür wird in Schritt 1) ein geeigneter Partner gesucht, wobei in diesem Fall beide anderen Applikationen in Frage kämen.

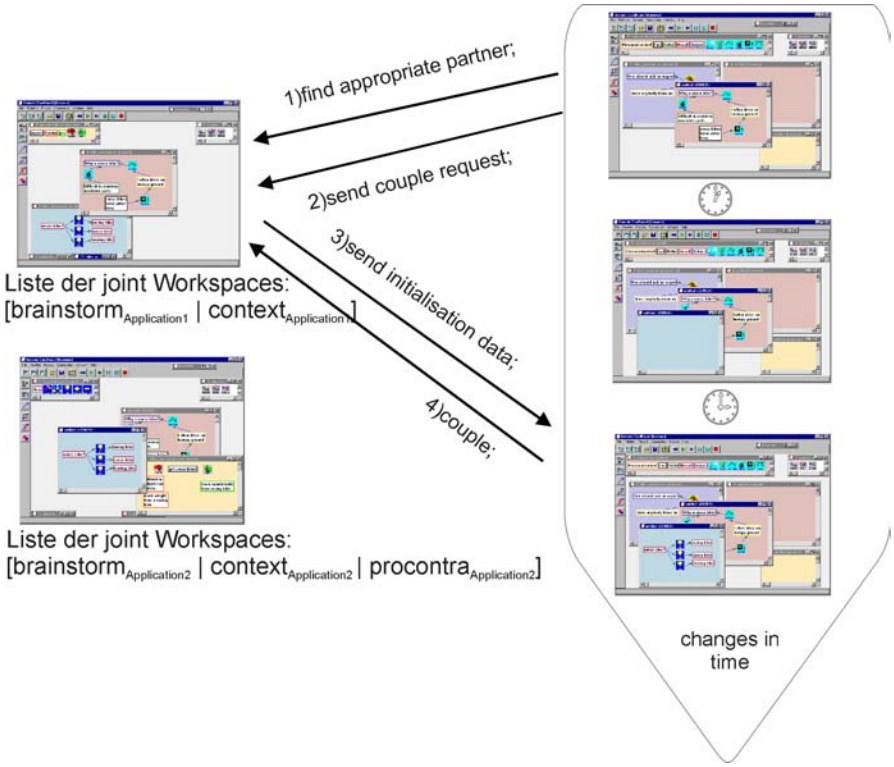


Abbildung 27

Wird von einer Applikation aus ein „join“ durchgeführt, so wird zuerst eine geeignete Partner-Applikation gesucht. Auf die Kopplungsanfrage werden die Initialisierungsdaten zurückgesendet. Erst am Ende wird das eigentliche Koppeln durchgeführt.

Nachdem jedoch in diesem Beispiel die obere ausgewählt wurde, wird an diese ein „couple request“ gesendet (Schritt 2). In Folge davon kann der Workspace selbst bereits aufgebaut werden, hat jedoch noch keine Daten (Bild rechts, mittig). In der nächsten Phase sendet die angefragte Applikation die Initialisierungsdaten (Schritt 3). Erst danach wird die eigentliche Synchronisation vorgenommen (Schritt 4).

Dieses eher umständliche Kopplungskonzept für das „join“ ist als ein Erfahrungshintergrund zu werten, der in die Neukonzeption des MatchMaker TNG (Tewissen, Baloian et al., 2000; Jansen, Pinkwart & Tewissen, 2001; Jansen, 2002) eingeflossen ist. Dort werden gerade Objektmodelle vorgehalten, zusätzlich aber auch Veränderungen in den Modellen an alle betroffenen Applikationen propagiert. Daraus ergibt sich eine Kombination eines zustandsorientierten und eventorientierten Ansatzes.

Trotz der beschriebenen und implementierten Erweiterungen des „join“-Konzeptes, das ohne die Dalis-Anbindung voll funktionsfähig ist, wurden im Verlauf der Entwicklung der Diskussionsunterstützung weitere Adaptionen notwendig, um das „join“ mit den für die Diskussion verwalteten Daten in Dalis kompatibel zu gestalten.

Wegen der in den Folgekapiteln noch eingehend vorgestellten Perspektiven müsste von Dalis eine vollständige persistente Datenhaltung implementiert werden. Jegliches Öffnen eines Workspaces müsste dann auf diese Daten zugreifen. Das „join“ würde dann einen allgemeiner, „öffentlicher“ Zugriff sein, während die „private“ Workspaces an einen konkreten Autor oder Autorin gebunden wären. Das bisher realisierte „join“-Konzept, würde so nicht mehr benötigt werden, da Dalis eine zentrale Verwaltung der Objekt-zustände zur Verfügung stellen würde.
Weder die Autorschaft, noch die vollständige Persistenz der Daten in Dalis sollte aber im Rahmen dieser Arbeit umgesetzt werden, die sich zentral mit den Perspektivenwechseln beschäftigt. Außerdem hätte das weitergehende Konsequenzen nach sich gezogen: Das Interface zwischen den Teilsystemen MatchMaker, CardBoard und Dalis hätte erweitert werden müssen, um diese zusätzliche Information der Autorschaft überhaupt kommunizieren zu können.
Eine vereinfachende Idee bestünde darin, immer alle Personen ausschließlich in „joint“ Workspaces arbeiten zu lassen. Es wären so zwar keine privaten Daten möglich, dafür könnte auf die Autorschaft verzichtet werden.

6.2.2 Paletten

Die Aufgabe der Paletten besteht vor allem darin, den Nutzerinnen und Nutzern die Relationen und Kartentypen einer visuellen Sprache zu präsentieren. Einerseits kann so auf einen Blick das Spektrum der jeweiligen visuellen Sprache erfasst werden und die Kartentypen können leichter gegeneinander abgewogen werden, um einen geeigneten herauszugreifen. Andererseits können so Karten durch einfaches Drag & Drop aus der Palette generiert werden.
Paletten sind selbst Workspaces, jedoch nicht als Arbeitsbereiche vorgesehen. Die CardBoard-Erweiterung besteht darin, dass für die in der Initialisierungsdatei frei anzugebenen Sprachen die Paletten-Definitionen automatisch geladen werden. Abbildung 28 zeigt wie diese Palettendateien referenziert werden.
Außerdem wird durch das System sichergestellt, dass sich immer die Palette im Vordergrund befindet, die zum gerade bearbeiteten Workspace-Typ passt.
Der ebenfalls in Abbildung 28 aufgeführte Eintrag [InitFunctionsPalette] bezieht sich noch auf eine bestimmte Art von Palette, die Funktionspalette. Auf diese wird in Kapitel 6.4 weiter eingegangen. Diese Palette enthält spezielle Karten, die der Steuerung der Interaktion mit Dalis dienen.

```
[Initpaletten]
Init="protocol.ws","protocol"
Init="explore.ws","explore"
Init="brainstorm.ws","brainstorm"
Init="topics.ws","topics"
Init="argument.ws","argument"
Init="procontable.ws","procontable"

[InitFunctionsPalette]
Init="functions.ws","functions"
```

Abbildung 28

Definitionen, welche Paletten geladen werden sollen.

6.3 Die visuellen Sprachen für die Diskussionsunterstützung: Die Perspektiven

Im Folgenden werden die im System umgesetzten visuellen Sprachen vorgestellt, die Perspektiven. Die Grundlage für deren Auswahl sind Vorgängerversionen und einzelne Tests[10], die zu Veränderungen der jeweiligen Objektmengen führten, die jedoch weitergehend evaluiert werden sollten.

Die Sprachen sind so gestaltet, dass aus ihren primitiven Objekten jeweils Graphen oder graphische Darstellungen entwickelt werden können, die auf Basis ihrer „intendierten" symbolischen Bedeutung dafür geeignet sind, bestimmte Strukturen zu entwickeln oder auch ganz frei Notizen anzufertigen. Bei argumentativen Diskursen können beispielsweise Argumentstrukturen aufgebaut werden. Tabellen können für Pro-/Kontradebatten herangezogen werden. Ein UND/ODER-Graph kann für die Diskussion und Abwägung von Alternativen entwickelt werden. Ein Protokollbereich dient der Zusammenfassung und Planung eines Gespräches. Ein weiterer Bereich dient der Exploration von Themen. Damit passen die einzelnen Sprachen zu verschiedenen Phasen der Diskussion.

Durch die strukturierte Darstellung können insbesondere die Argumentationen, die Tabellen und die UND/ODER-Graphen zur Wissenselizitation herangezogen werden, da sie Lücken sichtbar machen. Dies ist eine Eigenschaft der visuellen Darstellung (Boxtel & Veerman, 2001). Viele Wissenselizitationsmethoden basieren auf bestimmten Strukturierungsverfahren. Sortierungen, Clustering oder Bewertungen sollen Zugehörigkeiten

[10] Als Test ist die im Kapitel „Diskussionsunterstützung: Eine Vorstudie zur Hypothesenbildung" erläuterte Studie eingeflossen sowie Vorläufer-Versionen zu dieser Studie, eine mehrtägige eher protokoll-orientierte Mitschrift einer Projekt-Sitzung, die mit dem CardBoard durchgeführt wurde und die Verwendung des CardBoards bei der Darstellung konkreter Inhalte.

und Abhängigkeiten verdeutlichen (Cooke, 1994). Tabellen sind besonders gut geeignet, um auf unausgefüllte Felder zu verweisen. Ein Beispiel dafür ist das „repertory grid"-Verfahren nach Kelly (1970), das z.B. von Gaines und Shaw (1993) schon einmal in eine umfangreiche visuelle Sprache umgesetzt wurde.

Suthers (1999B) versucht Vorteile verschiedener diagrammatischer Darstellungsformen herauszuarbeiten. Er beschreibt, dass Tabellen dazu führen, die Relationen zwischen den Objekten vollständig zu definieren. Dagegen verleiten explizite Relationen in Graphen offenbar zu singulären Verbindungen. Der Vorteil besteht darin, dass die Relationen in diesem Fall als eigenes Objekt greifbar sind. Sie stellen deshalb in der Gesprächssituation einen einfachen, optischen Bezugspunkt in der Visualisierung dar, auf den referenziert werden kann. Sowohl die Graph- als auch die Tabellenform kann in unterschiedlichen Kontexten sinnvoll sein. Suthers (1999A, 1999B) benennt diese Passung zwischen Aufgabe und Darstellungsmittel als „salience"[11]. Das bedeutet, dass die hervorspringenden Eigenschaften einer Darstellung mit der Aufgabenstellung übereinstimmen sollten.

In den folgenden visuellen Sprachen wird für die Pro/Kontradarstellung eine tabellarische Darstellung gewählt, die Schwerpunkte und Lücken verdeutlicht. Dagegen werden Argumentationen als Graphstrukturen repräsentiert. Das bietet sich für die Argumentationen in Anlehnung an Toulmin (1958) an, da hier nur spezielle Relationen angeboten werden und eine vollständige Verbindung der Argumentbestandteile gerade nicht erwünscht ist. In der visuellen Sprache zur Exploration sollen die Relationen vorerst nur punktuell aufgeführt werden. Ein tabellarisches Herangehen würde aus meiner Sicht diese Phase zu stark organisieren und den explorativen Charakter aufheben.

6.3.1 Allgemein zur Gestaltung der visuellen Sprachen

Generell muss der Umfang der visuellen Sprachen, also die Anzahl und auch die Art der Relationen und Beitragstypen, abgewogen werden. Eine zu große Ausdrucksmächtigkeit kann schnell unübersichtlich wirken und zu redundanter Ausdrucksweise führen (Suthers, Toth und Weiner, 1997). Das heißt, dass verschiedene Sprachelemente für gleiche Beitragstypen herangezogen werden. Je nachdem, welche Inhalte mit ihnen dargestellt werden sollen, müssen die Beitrags- und Relationstypen an das Problem angepasst werden. Im vorliegenden Ansatz zur Diskussionsunterstützung kann durch die Vielfalt an Perspektiven der Umfang der einzelnen Sprachen reduziert werden, um so eine Anpassung an eine ganz konkrete Arbeitsphase bzw. Anwendungsmethode zu erreichen.

[11] Übersetzung: Hevorspringen

Die Gestaltung der Relationen stand im Spannungsfeld zwischen der technischen Umsetzbarkeit und Ausdrucksmächtigkeit einerseits und der benutzungsfreundlichen Darstellung andererseits.

In Bezug auf die Ausdrucksmächtigkeit stellen die verwendeten visuellen Sprachen umfangreiche Mittel zur Verfügung, da wie bereits erwähnt, n-stellige Relationen darstellbar sind, die zur Entwicklung von Hypergraphen führen. Es zeigte sich jedoch in der Versuchsphase, dass mehrstellige Relationen nur schwer lesbar waren. Zugunsten der Übersichtlichkeit ist deshalb darauf verzichtet worden. Eine n-stellige Relation würde durch n Links visualisiert, die eine Konnektorkarte mit einer entsprechenden Anzahl von Inhaltskarten verbindet.

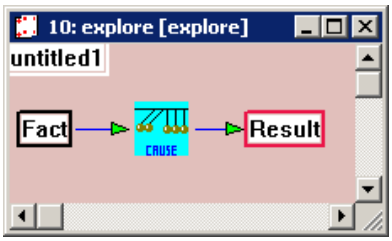


Abbildung 29

Eine Relation bestehen aus zwei Inhaltskarten („Fact“ und „Result“) und einer Karte, der Konnektorkarte, die den Typ der Relation symbolisiert, die über zwei Links mit den Inhaltskarten verbunden ist.

Weiterhin bestand eine Frage darin, wie die Konnektorkarten gestaltet werden sollten. Für die Diskussionsunterstützung wurden die Konnektorkarten letztendlich durchgängig mit festen Symbolen ausgestattet, wie es in Abbildung 29 zu sehen ist. Es bestand zusätzlich die Option, solche Konnektorkarten zu verwenden, die wie die Inhaltskarten, frei beschriftet werden können. Außerdem hätte es den Nutzerinnen und Nutzern überlassen werden können, selbst ausgewählte Symbole zu verwenden. In dem hier gewählten Vorgehen sollten jedoch die Symbole jeweils einen Hinweis auf den Bedeutungszusammenhang in dem sie verwendet werden geben („intendierte Semantik“). Es soll gerade ausgeschlossen werden, dass immer neue Symbole ein „gemeinsame Sprache“ in der Gruppe verhindern.

Auch die Art der Symbole hat sich als relevant herausgestellt. In der Versuchsphase beinhaltete die dort verwendete Sprache Symbole, die optisch gerichtet waren, wie z.B. einen Folgerungspfeil, der von links nach rechts zeigte. Diese Richtung wirkte verwirrend, da sie teilweise entgegengesetzt zu den interaktiv hergestellten Links war. Deshalb ist auf solche gerichteten Symbole für die Relationen verzichtet worden.

In den ersten Entwicklungen des CardBoards wurde die Entscheidung getroffen, die Typen der Inhaltskarten durch Farben zu kennzeichnen. Diese Entscheidung ist

rückblickend zu kritisieren. Es zeigten sich zwei Probleme: Das erste bestand darin, dass anhand der Farben nur relativ wenige trennende Kategorien definiert werden können. Kategorien wären rot, blau, gelb, grau... Die Farben sind jedoch schwierig zu unterscheiden, wenn sie dichter beisammen liegen. Außerdem zeigte sich, dass die Bedeutung der Farben nur sehr schwer zu erlernen ist. Dagegen können Formen sehr vielfältig und auch bedeutungstragend definiert werden. Die Form selbst ist leicht als Kategorie zu verstehen wie eine Wolkenform, ein Warndreieck, etc. Diese Verbesserung ist in die zeitlich nachfolgende Entwicklung des FreeStylers eingeflossen.

6.3.2 Die visuelle Sprache zur Diskussionsstrukturierung (Protokoll)

In der Gruppe der visuellen Sprachen für die Diskussionsunterstützung ist eine Sprache für die Diskussionsstrukturierung zuständig. Diese Sprache wird als Protokollsprache bezeichnet. Sie ist jedoch gerade nicht dazu gedacht, die Diskussion im Sinne eines klassischen Protokolls zu begleiten. Sie soll vor allem die Planung und Strukturierung des Gesprächs unterstützen und ist deshalb im Wesentlichen für die Anfangsphase einer Diskussion gedacht. Darin werden Besprechungspunkte, erste Zusammenhänge und Bemerkungen festgehalten. Es werden Beiträge wie Fragen, Aspekte und Ideen erst einmal gesammelt. Das Notieren der Beiträge soll schnell gehen und die eigentliche Diskussion dazu später geführt werden. Deshalb stellt diese Sprache nur eine kleine Auswahl an Beitragskategorien und Relationstypen zur Verfügung, denn sowohl die Klassifikation der Beiträge als auch deren Strukturierung mit Relationen ist im Allgemeinen anspruchsvoll und zeitaufwendig. Eine einzige allgemeine Relation kann dazu verwendet werden, um die Beiträge in Verbindung zu setzen.

Natürlich kann die Liste der zu behandelnden Besprechungspunkte auch während der Diskussion leicht ergänzt oder Punkte wieder gestrichen werden. Zu diesen Ergänzungen gehören später die Diskussionsergebnisse, die mit in das Protokoll integriert werden.

Die Tabelle 10 stellt in der linken Spalte die visuellen Objekte der Protokollsprache dar. In der rechten Spalte werden die Kategorien erläutert.

In Abbildung 30 ist links ein Schema für die Verwendung der Protokollsprache angedeutet. In ihr können Listen von „Tasks“ erstellt werden, die selbst schon auf Teilprobleme verweisen können. Dies könnten selbst wieder „Tasks“ sein oder auch „Questions“. Ein empfohlenes Strukturierungsmittel ist hier die Einrückstruktur. Außerdem kann die Relation verwendet werden, um Zusammenhänge aufzuzeigen. Auf der rechten Seite der Abbildung wird beispielsweise ein Zusammenhang „Task“-übergreifend markiert. Die

„Contribution“- und „Question“-Karten stellen im Wesentlichen ad hoc-Beiträge zur ersten „Task“ dar.

Task	Die Kategorie „Task“ ist zum Festhalten von Problemen oder Arbeitspaketen und grundlegenden Fragestellungen gedacht.
Question	Konkrete, detailliertere Fragen können mit der „Question“-Kategorie von den Tasks unterschieden werden.
Contribution	Auf Karten der „Contribution“-Kategorie können beliebige andere Beiträge notiert werden.
Idea	Mit Karten der „Idea“-Kategorie können Ideen notiert werden. Diese können natürlich auch außerhalb des gerade diskutierten Kontextes liegen.
RELATED	Mit dieser Relation können Beiträge assoziiert werden.

Tabelle 10

Menge und Bedeutung der Beitrags- und Relations-Typen der Protokollsprache

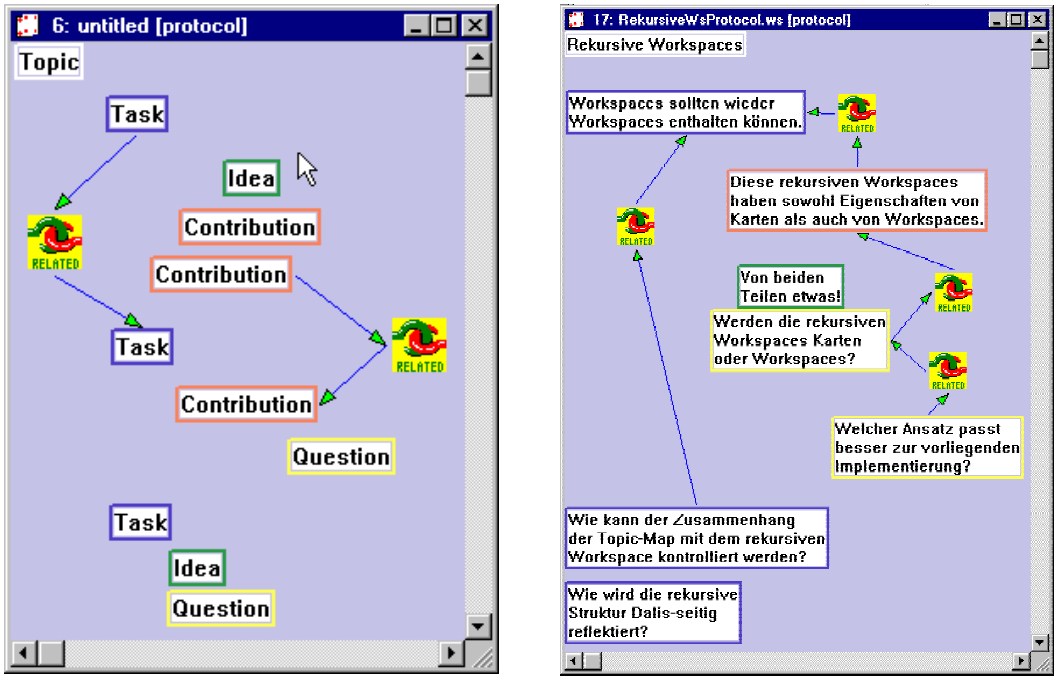


Abbildung 30

Schema (links) und Beispiel (rechts) für die Verwendung der Protokollsprache.

6.3.3 Die visuelle Sprache zum Explorieren

Die Explorationssprache dient der Vor- wie Nachbereitung anderer Arbeitsphasen. Aus der Diskussionsanfangsphase werde Beiträge aufgegriffen und exploriert. Einzelne Aspekte sollen dann später in spezifischeren Darstellungen, wie Tabellen, weiter diskutiert werden. Mit der Explorationssprache sollen Argumente darstellbar sein, sowie ein Themenfeld exploriert werden können. Zu einem Thema werden in einem

Arbeitsbereich Fakten gesammelt, Einschränkungen und Bedingungen aufgeführt sowie Hintergründe, Zusammenhänge und Meinungen dargestellt.

Je nach Bedarf können hier sehr detailliert Themengebiete entwickelt werden. Es ist davon auszugehen, dass die Strukturierung von Inhalten mit dieser visuellen Sprache arbeits- bzw. zeitaufwendig ist. Durch die größere Menge an Relationen können umfangreiche Strukturen aufgebaut werden, die der Reflexion des Gegenstandes dienen.

Verzichtet wurde allerdings – wie vorher in dieser Arbeit schon diskutiert - auf die Darstellung von „moves“ wie sie in linguistischen Analysen meist verwendet werden. Dort interpretieren „moves“ die linguistische Bedeutung zwischen Beiträgen in Gesprächen, meistens in Dialogen. Beispiele dazu wären „Re-initiative“, „Gegen-Initiative“ oder „Reaktion“ (Franke, 1990). Auch die „Answer“-Relation im Sepia-System wäre darunter einzuordnen. Hier sollen die Beiträge im Wesentlichen auf inhaltlicher Ebenen miteinander in Verbindung gebracht werden.

Die Suche nach geeigneten Primitiven gestaltete sich jedoch schwierig. Gruppen von Primitiven werden auch in anderen Ansätzen verwendet, wie zum kooperativen Schreiben (Sepia), zum wissenschaftlichen Argumentieren (Belvedere) oder auch für didaktische Zusammenhänge (Baloian Tataryan, 1997); aus dem linguistischen Umfeld stammen Diskursanalysen (Levin & Moore, 1977; Pilkington & Mallen, 1996; Baker, 1995).

Mit der Explorationssprache sollten möglichst viele Arten von Zusammenhängen ausgedrückt werden. Versucht wurde, die in anderen Systemen verwendeten Primitiven aufzugreifen, zu gruppieren und allgemeinere Bezeichnungen dafür zu finden. Tabelle 11 stellt diese Sammlung von Primitiven dar, die in den genannten Systemen verwendet werden sowie eigene Ergänzungen. In der ersten Zeile werden diese Primitive allgemeiner kategorisiert, um daraus wiederum Beitragkategorien für die Explorations-sprache abzuleiten.

Allgemeine Beschreibung	Ausgangspunkt	Gegebenheiten	Vages/ Abstimmbares	Persönliche Wertigkeiten/ Vorlieben/ Umstände	Ergebnisse
Mögliche Primitive	Problems Tasks Questions	Constraints Facts Option Thesis Premise	Hypothesis Belief Opinion Advantage Disadvantage	Interest Desire Preference Advantage Disadvantage	Thesis Solution Compromise Suggestion Result Hypothesis
Abgeleitete Beitrags-Kategorie	Subject	Fact	Estimation	Personalcontext	Result

Tabelle 11

Beitragskategorien für die Explorationssprache

Neben den Beitragskategorien in Tabelle 11 werden in Tabelle 12 die verwendeten Relationen erläutert.

	Ein Beitrag stützt einen anderen.
	Ein Beitrag stellt eine Lösung für ein Problem dar.
	Ein Beitrag beschreibt, warum etwas nur unter bestimmten Bedingungen eintritt, zutrifft, etc.
	Folgern; Eine Begebenheit führt zu einer weiteren.
	Erklären-Warum, Begründen, Rechtfertigen
	Beiträge die andere in Frage stellen aufgrund von Indizien, Annahmen, die eine Gegenposition darstellen oder einen logischen Widerspruch aufzeigen.

Tabelle 12

Bedeutung der Relations-Typen der Explorationssprache.

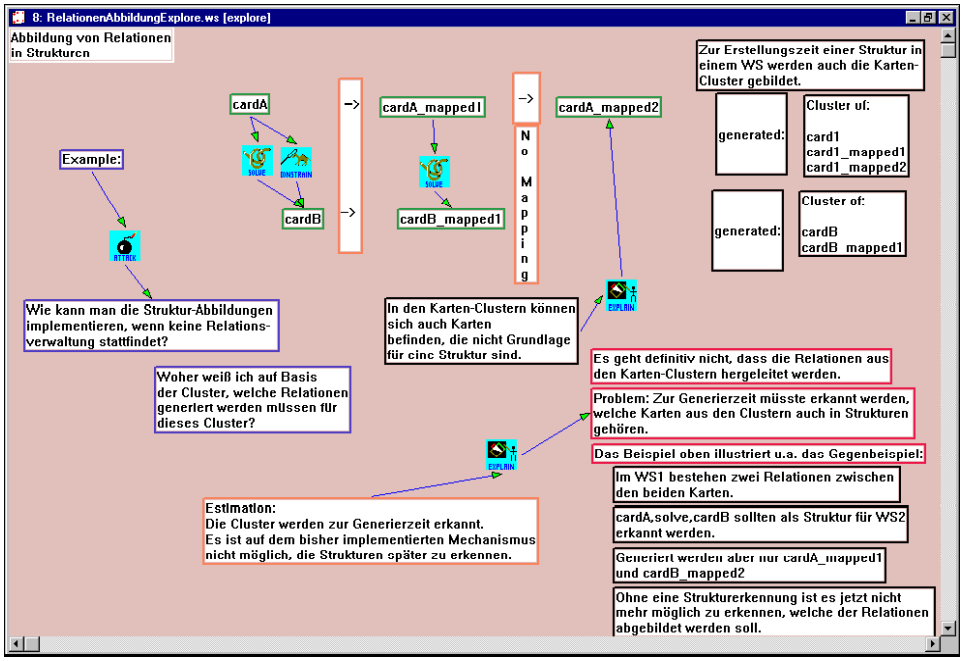


Abbildung 31

Beispiel 1 für die Verwendung der Explorationssprache.

Abbildung 31 und Abbildung 32 zeigen Beispiele für die Arbeitsweise in einem Explorations-Workspace. Inhaltlich befassen sie sich mit Designfragestellungen zum Entwurf der Diskussionsunterstützung. Im ersten Beispiel (Abbildung 31) werden Karten u.a. als Auflistungen (rechts unten) und als Container für selbstdefinierte Symbole bzw. optisch strukturierende Elemente verwendet. Gemeint sind damit z.B. die Pfeile, die ohne

weiteren Text in einer Karte enthalten sind und in diesem Beispiel die Abbildung von Karten zwischen Workspaces andeuten.

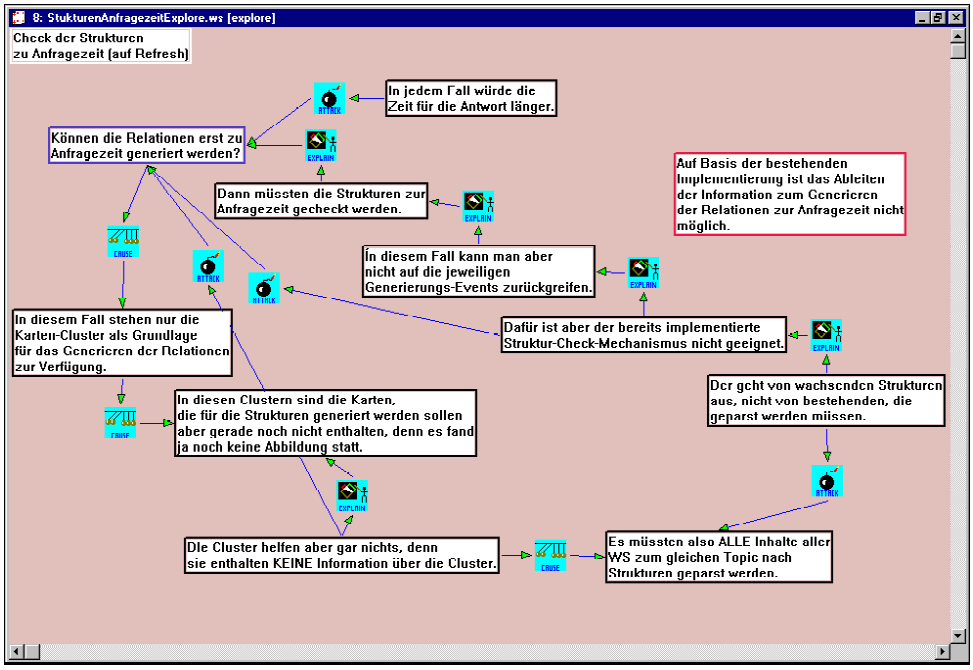


Abbildung 32

Beispiel 2 für die Verwendung der Explorationssprache.

Auch bei solchen optisch oder inhaltlich strukturierenden Symbolen können die Kartenkategorien sinnvoll verwendet werden. In diesem Fall wurden die Pfeile in „Estimation"-Karten eingetragen. So kann eine Unsicherheit in dieser Einschätzung ausgedrückt werden.

Das andere Beispiel (Abbildung 32) zeigt zwei Begründungsketten, die sich in der unteren rechten Ecke mit einem gemeinsamen Bezug schließen, der so auf ein ganz zentrales Problem bei dem dort erörterten Design-Vorschlag verweist.

6.3.4 Die visuelle Sprache für das Brainstorming

Die Brainstormingsprache lehnt sich an den IBIS-Ansatz (Conklin & Begemann, 1987) an. Sie stellt an UND/ODER-Graphen angelehnte Hilfsmittel zur Verfügung, Ideen und Alternativen festzuhalten. Mit ihr können Optionen zu Fragestellungen in mehreren Baumebenen aufgeführt werden. Diese können dann positiv oder negativ bewertet und mit Kriterien versehen werden. Durch Markierungskarten können Entscheidungen und Präferenzen deutlich gemacht werden.

Für das eigentliche Brainstorming werden die vorgegebenen Relationen jedoch erst einmal gar nicht verwendet. Hier werden die jeweiligen Ideen der beteiligten Gruppenmitglieder einfach auf Karten notiert. Dazu wird möglichst immer die gleiche Karten-

kategorie verwendet, um während des Brainstorms nicht über die Kategorie reflektieren zu müssen. Je nachdem worüber das Brainstorming durchgeführt wird, kann dies entweder ganz allgemein die „Idea"-Karte sein oder bereits die „Option"-Karte. Im Grunde könnte sogar über Begründungen ein Brainstorm durchgeführt werden. Dem eigentlichen Brainstorming kann dann die Strukturierung der Vorschläge in dem UND/ODER-Graph folgen.

Die Brainstormingsprache ist deshalb weniger als Hilfe für das Brainstorming selbst zu sehen, als dass es eine Möglichkeit bietet, die Ergebnisse nachfolgend zu strukturieren und zu bewerten. Die Brainstorming-Methode könnte prinzipiell mit allen visuellen Sprachen des DiscBoard-Systems durchgeführt werden.

Task	Die Ausgangsfragestellung.
Idea	Allgemeine Ideen.
Option	Zum Festhalten von alternativen Vorschlägen zu einer übergeordneten Frage.
Reason	Ein Grund kann sowohl eine Bewertung als auch Kriterien erläutern.
	Die „Idea"-Relation stellt eine ODER-Verbindung zu anderen Ideen auf der gleichen Ebene dar.
	Die „Ideagroup"-Relation stellt eine UND-Verbindung zu anderen Ideen auf der gleichen Ebene dar. Die Zugehörigkeit mehrerer Optionen oder Ideen kann momentan jedoch nur durch Nähe der verwendeten Relationskarten visualisiert werden.
	Diese Relation signalisiert eine positive Einstufung einer Idee.
	Diese Relation signalisiert eine negative Einstufung einer Idee.
	Mit dieser Relation können Kriterien dargestellt werden, unter denen Ideen verwirklicht werden können, Voraussetzungen, etc.
	Zwei Ideen können gegeneinander bewertet werden.
	Endgültige Entscheidungen werden markiert.

Tabelle 13

Bedeutung der Beitrags- und Relationstypen der Brainstormingsprache.

In Tabelle 13 werden die Kategorien und Relationen der Brainstormingsprache erläutert. Abbildung 33 zeigt ein Schema der Brainstormingstrukturen. Von der Baumwurzel, der „Task", verzweigen sich die Optionen. Kriterien und Gewichte beziehen sich auf die einzelnen Optionen. Falls eine Auswahl getroffen wurde, kann diese durch die „Select"-Karte hervorgehoben werden. In Abbildung 34 wird die Bewertung der drei Optionen dargestellt. Dabei gewichten die Begründungen teilweise mehrere Optionen. Die mittlere

Option verweist auf eine weitere „Task“, die dafür gelöst werden müsste, was die übergeordnete Option in diesem Fall negativ gewichtet.

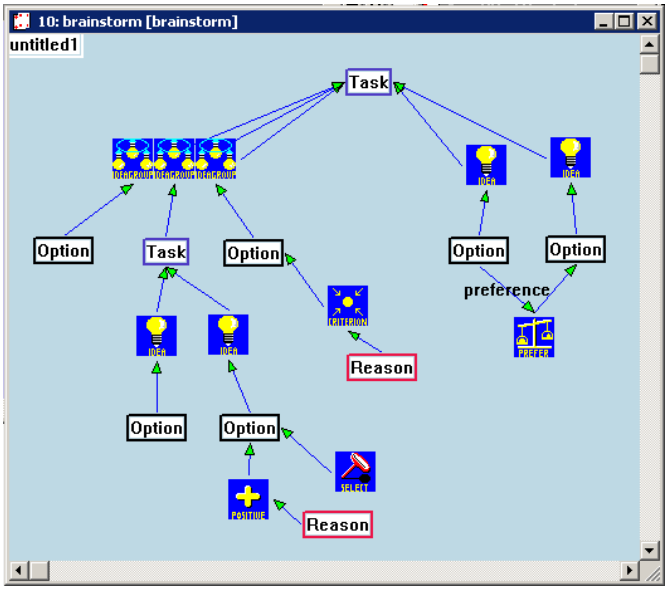


Abbildung 33

Schema für die Verwendung der Brainstormingsprache.

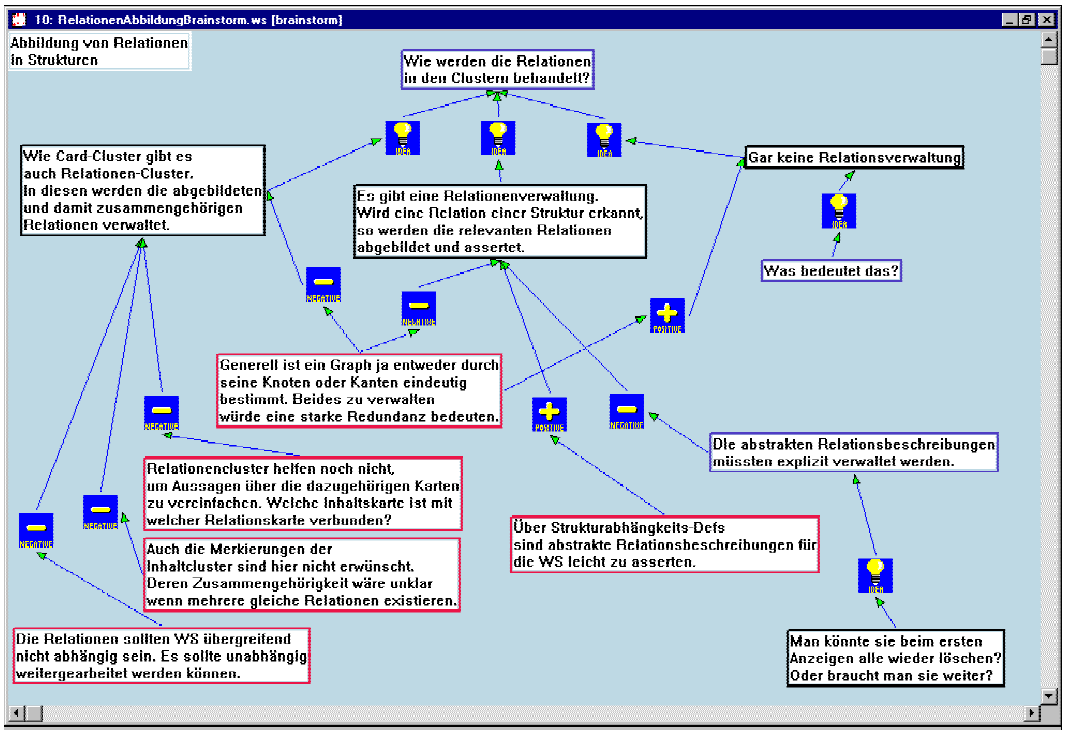


Abbildung 34

Beispiel für die Verwendung der Brainstormingsprache.

6.3.5 Die „Topic Map“

Die Diskussionsunterstützung soll es ermöglichen, punktuell Themen aufzuarbeiten, diese später weiter zu vertiefen und sie in ein Verhältnis zu setzen. Die Zuordnung von Themen zu Diskussionssträngen ist in zweierlei Hinsicht sinnvoll. Erstens können während der Diskussion verschiedene Aspekte jeweils Themen („Topics“) zugeordnet werden und so die Diskussion und die Inhalte strukturiert werden. Zweitens kann so ein Themengraph aufgebaut werden, die „Topic Map“[12], der sowohl die erarbeitete Wissensstruktur darstellt als auch einen Zugriff auf die jeweiligen Diskussionsstränge erlaubt.

Durch die Untergliederung von Problemen in Teilprobleme (Dekomposition) kann ein Problemlösungsprozess positiv beeinflusst werden. Im Allgemeinen dient sie dem Verständnis der Problemzusammenhänge und reduziert die Komplexität.

Im Diskussionsunterstützungssystem wird jeder Workspace mittels einer „Topic“-Karte einem Thema zugeordnet. In den vorangegangenen Abbildungen war diese jeweils oben links in den Workspaces zu erkennen (siehe auch Abbildung 35 und Abbildung 36). Konzeptuell muss jeder Arbeitsbereich genau einem Topic zugeordnet sein. Das heißt, dass jede visuelle Sprache auch die „Topic“-Karte als Kartentyp enthalten muss.

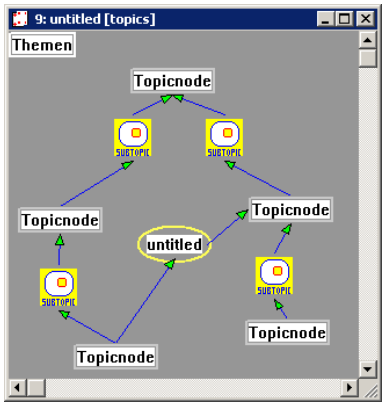


Abbildung 35

Schema für eine „Topic Map“.

Wegen der geforderten Zuordnung von einem „Topic“ zu einem Workspace unterliegt die „Topic“-Karte einer besonderen Kontrolle. Sie wird nicht frei zur Verfügung gestellt, sondern automatisch generiert. Nur der Inhalt muss von den Nutzerinnen und Nutzern hinzugefügt werden. Jeder generierte „Topic“ wird automatisch in die „Topic Map“ übernommen. Diese Übernahme basiert auf einem Abbildungsmechanismus, der in

[12] Diese entstehenden „Topic Maps“ sind nicht mit dem 1999 verabschiedeten ISO/IEC Standard 13250 über Topic Maps zu verwechseln. Diese können als Ontologien verstanden werden, die auf Ressourcen im Web verweisen, die entsprechend durch Topics klassifiziert worden sind.

Kapitel 6.6 vertieft vorgestellt wird. Zu jedem „Topic“ können die verschiedenen Perspektiven herangezogen werden. Der „Topic“ wird jedoch nur ein Mal in der „Topic Map“ erwähnt.

Die „Topic Map“ ist damit eine abstrahierte Sicht auf die Themen. In sie werden alle Themen, die in der Diskussion behandelt werden, automatisch integriert. Dementsprechend gibt es nur eine „Topic Map“ in der Diskussionsunterstützung.

Begrifflich ist es für die „Topic Map“ schwierig, die normalen Beitragskarten von dem „Topic“ zu unterscheiden, dem die „Topic Map“ selbst zugeordnet ist. Da es nur eine „Topic Map“ geben soll, ist deren „Topic“ einem Defaultwert zugeordnet („Themen“), wie es in Abbildung 35 und Abbildung 36 zu sehen ist.

Die restliche visuelle Sprache für die „Topic Map“ ist nur aus wenigen Kartentypen zusammengesetzt, mit denen Themen zueinander ins Verhältnis gesetzt werden können. Sie enthält ausschließlich einen Beitragskartentyp, den „Topicnode“, eine frei editierbare Relation und die Teilthema-Relation. Zwischen den Topics dürfen beliebig viele Relationen hergestellt werden (siehe Tabelle 14 und Abbildung 35, die die „Topic Map“ schematisch darstellt). Mit Hilfe der „Topicnode“-Karte können nicht nur solche Themen aufgeführt werden, die bereits in den anderen Perspektiven behandelt wurden, sondern auch neue eingebunden werden.

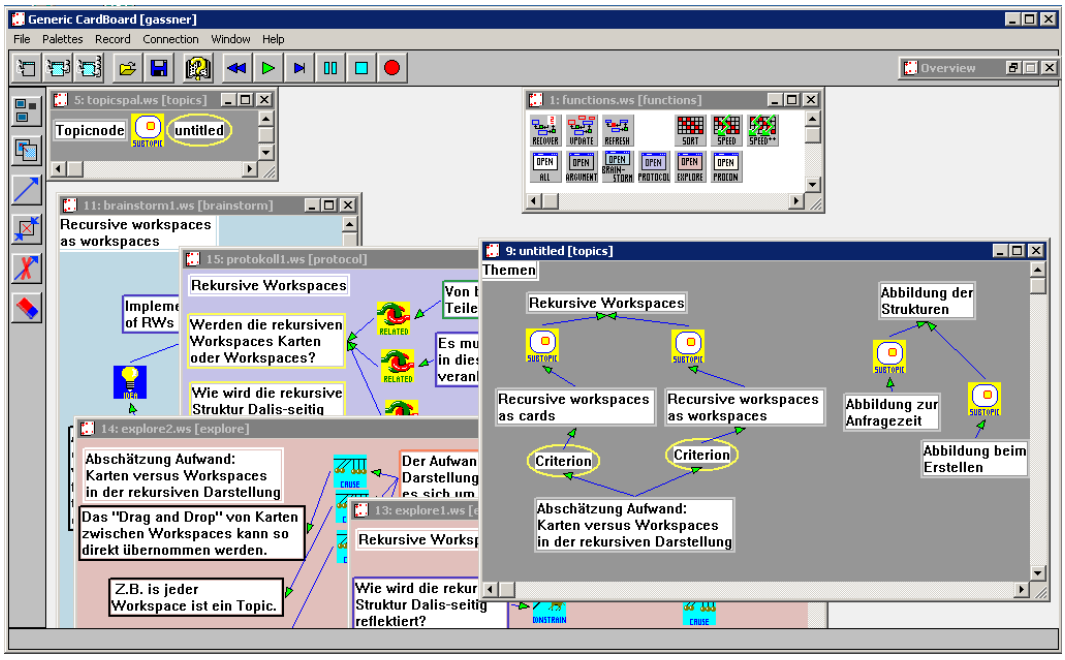


Abbildung 36

Beispiel einer „Topic Map“. Es werden existierende Topics mit neuen vermischt.

Die „Topic Map“ dient nicht nur der Übersicht über die behandelten Themen, sondern wird auch zur Arbeit mit den Themen herangezogen. Über die „Topic Map“ können

Workspaces zu einem bestimmten Thema und unter einer bestimmten Perspektive geöffnet werden. Dazu werden Funktionskarten verwendet, die im Kapitel 6.4 erläutert werden. Die Interaktion mit den „Topic“-Karten der verschiedenen Perspektiven und der „Topicnode“-Karten der „Topic Map“ unterliegt intern verschiedenen zusätzlichen Kontrollen.

	Im einzigen Beitragskartentyp werden die Themen festgehalten.
	Mit dieser Karte können beliebige Relationen zwischen Themen definiert werden.
	Es ist davon auszugehen, das die Teilthema-Relation besonders häufig genutzt wird. Dafür wird eine eigene Karte zur Verfügung gestellt.

Tabelle 14

Bedeutung der Beitrags- und Relations-Typen der Brainstormingsprache.

6.3.6 Die visuelle Sprache zum Argumentieren

In Diskussionen sind Argumente wichtige Bestandteile. Werden sie klar herausgearbeitet, so steuern sie nicht nur den Diskussionsverlauf, sondern sind gleichzeitig auch ein wichtiges Resultat einer Diskussion. Für die Diskussionsunterstützung wird deshalb davon ausgegangen, dass es sinnvoll ist, die entstandenen Argumente gesondert zu sammeln. Dadurch, dass sie als Teilstrukturen automatisch aus anderen Darstellungen herausgegriffen werden, werden sie u.U. überhaupt erst sichtbar. Zusammengefasst werden jeweils die Argumente zu einem „Topic“, die dafür aus anderen Perspektiven gefiltert werden müssen. Der Vorgang des Filterns wird ausführlich in den Kapiteln 6.7 und 6.8 beschrieben. Hier soll zur Erläuterung nur erwähnt werden, dass die im Folgenden definierten Argumentstrukturen automatisch vom System in verschiedenen Perspektiven erkannt, herausgegriffen und in der Argumentperspektive gesammelt werden. Zwischen den Workspaces wird also eine Relation definiert, auf deren Grundlage zu einem benutzerdefinierten Zeitpunkt die Argumente gefiltert werden. Die gefilterten Argumente können dann wieder frei strukturiert werden.

Argumentstrukturen werden typischerweise aus linguistischer Sicht definiert. Es gibt jedoch keine allgemein gültige Definition. Meißner (1994) weist auf die unterschiedliche Verwendung des Terminus „Argument“ hin. Einerseits stehe es für eine „Schlussfigur“, andererseits für „Sprechhandlungssequenzen [...], die als wesentliches Merkmal ein strittiges Moment aufweisen“ (Meißner, 1994, S. 25). Die Schlussfigur verweist auf die inhaltliche Struktur eines Argumentes. Es wird davon ausgegangen, dass ein Argument auf eine bestimmte Art und Weise strukturiert sein muss, z.B. muss verdeutlicht werden, wovon ausgegangen wird, und was man daraus ableitet. Stellvertretend für diesen Ansatz

ist die Arbeit von Toulmin (1958), die sich außerdem stark an logischen Aussagestrukturen orientiert.

Der Ansatz, das Argument als Sprechhandlungssequenz zu sehen, geht davon aus, dass bei einem Argument bestimmte Aussagen aufeinander folgen. Klein, mit der Theorie der konklusiven Sprechhandlungen (zitiert durch Meißner, 1994), vertritt diese zweite Auslegung. Die möglichen Sequenzen sind dabei Erklären-Warum, Begründen, Rechtfertigen, Folgern, welche sich wiederum aus Sprechakten zusammensetzen.

Für die angestrebte Diskussionsunterstützung soll die Toulmin'sche Schlussfigur Pate stehen. Dafür sprechen zwei Gründe: Erstens ist sie strukturorientiert und passt sich deshalb besser in den strukturorientierten Ansatz der Diskussionsunterstützung ein. Zweitens kann diese Struktur im Wesentlichen auch ohne Sprechakte nachgebildet werden. Das heißt, dass die Bestandteile der Struktur nicht als Sprechakte begriffen werden müssen.

Trotzdem sollen die von Klein beschriebenen Sequenzen Erklären-Warum, Begründen, Rechtfertigen und Folgern nicht völlig vernachlässigt werden. Auf der rechten Seite von Abbildung 37 ist jeweils ein Beispiel für diese vier Sequenzen aufgeführt, wie sie zumindest in einem Explorations-Workspace dargestellt werden können, wenn sie auch bisher nicht für den Argumentations-Workspace gefiltert werden.

Erklären-Warum ist „das Explizieren des Zustandekommens eines Sachverhalts" (Klein zitiert durch Meißner, 1994, S. 25). Dafür steht das erste Beispiel in der Abbildung rechts oben. Die Sequenz wird durch eine „Explain"-Karte ausgedrückt, die in diesem Fall ein Faktum als Voraussetzung angibt und auf ein Resultat verweist. „Rechtfertigen ist das Stützen des Anspruchs auf nichtnegative oder positive Bewertung ... eines Subjektes, ... eines Sachverhaltes, ... oder des Zustandekommens eines Sachverhaltes ..." (Klein zitiert durch Meißner, 1994, S. 26). Dafür steht das zweite Beispiel, das ein Resultat, das nicht erwünscht ist, durch einen persönlichen Umstand erklärt. Folgern „... soll das Ziehen einer Konsequenz aus Gegebenem bezeichnen ..." (Klein zitiert durch Meißner, 1994, S. 26). Hierfür steht das dritte Beispiel, bei dem ein Faktum die Voraussetzung für eine Einschätzung ist. „Begründen ist das Stützen von ... Ansprüchen in der Wahrheitsdimension ..." (Klein zitiert durch Meißner, 1994, S. 26). Diese Kategorie wird hier so interpretiert, dass ein gegebener Sachverhalt durch persönliche Einschätzungen begründet wird.

Da es sich um Darstellungen im Explorations-Workspace handelt, sind diese Beispiele als kurzer Exkurs anzusehen, der im Kontext der Argumentdarstellungen als Erläuterung

dienen soll. Sicherlich wünschenswert ist es, den Argumentfilter später um diese Argumentarten zu erweitern.

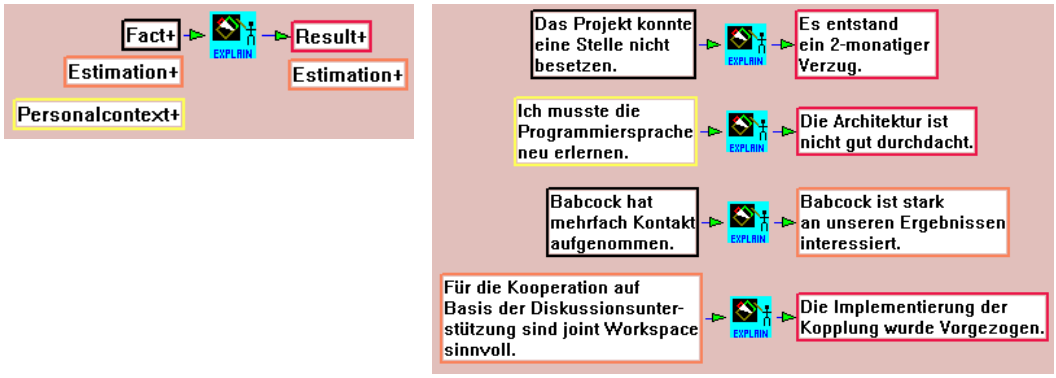


Abbildung 37

Die Argument-Sequenzen nach Klein werden im Explorations-Workspace alleine mit der „Explain"-Karte ausgedrückt (links). Auf der rechten Seite sind Beispiele dafür (folgern, rechtfertigen, begründen, erklären).

In Abbildung 38 wird die Argumentstruktur nach Toulmin dargestellt. Das Prinzip des Toulmin'schen Arguments ist eine Schlussfolgerung aufgrund von Daten, die in das Argument einbezogen werden. Darin besteht ein deutlicher Unterschied zu der Sicht nach Klein, der nicht nur Daten bzw. Fakten als Prämisse zulässt. In diesem Sinne kann die Toulmin'sche Schlussfigur auch als strenge Grundlage für wissenschaftliches Schließen interpretiert werden, wie es beispielsweise im Belvedere-System (siehe Kapitel 3.2) umgesetzt ist. Schlüsse aus nicht sicheren Voraussetzungen sind hingegen im Alltag üblich, wie es auch im Klein'schen Ansatz betont wird.

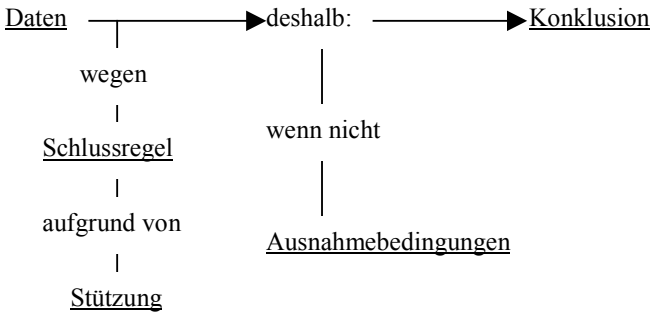


Abbildung 38

Die Argumentstruktur nach Toulmin (1958)

Neben den einfachen Daten, die nach Toulmin in ein Argument einfließen, können noch zusätzliche Prämissen eingebracht werden, die selbst wieder Daten sein können oder auch in weiteren Schlussregeln aufgeschlüsselt werden. Ein dritter Bestandteil sind die Ausnahmenbedingungen, unter denen das Argument nicht zutreffen würde. Toulmin problematisiert, dass die stützenden Daten sowie die Ausnahmebedingungen in den

meisten Argumentationen nicht explizit genannt werden. Das kann daran liegen, dass es schlicht überflüssig ist, sie zu nennen, weil diese Bestandteile entweder durch den Kontext oder das Vorwissen der Diskussionsteilnehmerinnen und -teilnehmer bekannt sind. Oft liegen hierin aber zentrale Ursachen für Missverständnisse oder falsche Argumentationen. Diese Tatsache kann auch als Hintergrund dienen, dass Visualisierungen zu sachlicheren Argumentationen führen können, wenn fehlende Bestandteile deutlich werden und fehlerhafte Aussagen zur Diskussion stehen.

Für die Diskussionsunterstützung wurde diese Toulmin'sche Struktur in eine Darstellung mit einer visuellen Sprache übertragen. Abbildung 39 (links) stellt die allgemeine Struktur eines Arguments in einem Argument-Workspace vor. Dabei sind die Kartentypen und deren Position in der Struktur relevant. Die Darstellung ist so zu lesen, dass bei nebeneinander oder untereinander stehenden Karten immer die eine oder die andere Karte verwendet werden kann. Das „+"-Zeichen in den Karten besagt, dass genau eine Karte davon in einer zulässigen Struktur an dieser Stelle vorhanden sein muss. Das „*"-Zeichen besagt, dass beliebig viele Karten dieser Art mit der entsprechenden Relation an der genannten Position in dem Argument auftreten können.

Abbildung 39

Allgemeine Argumentstruktur nach Toulmin in einem Argument-Workspace (links). Beispiel für eine Folgerung nach Toulmin (rechts).

Als Voraussetzung des Arguments wird die „Fact"-Karte verwendet, die durch die „Cause"-Relation auf die Konklusion verweist. Diese kann dann alternativ als Resultat oder Einschätzung kategorisiert sein. Die „Cause"-Relation macht den Kern des Arguments aus, der jeweils nur aus einer Voraussetzungs- und eine Konklusionskarte besteht. Diese Struktur kann insofern als Einschränkung bzgl. der Toulmin'schen gesehen werden, als dass sie jeweils nur eine Voraussetzungs- und eine Konklusionskarte vorsieht. Damit soll erst einmal eine überschaubare Grundlage für das Filtern der Argumente geschaffen werden. Um diese Einschränkung zu umgehen, könnten natürlich umfangreichere Inhalte in die Karten eingefügt werden. Der unterstützende Teil wird durch die „Support"- oder „Cause"-Karte in Verbindung mit weiteren Fakten oder

Einschätzungen aufgeführt. Einschränkungen werden mit der „Constrain“-Karte beigefügt. Eigentlich bezieht sich die Einschränkung des Arguments, die durch die „Constrain“-Karte ausgedrückt wird, auf den Schluss selbst, müsste also auf die „Cause“-Karte verweisen. Dies lässt sich aber weit schwieriger formalisieren, weshalb ich mich für diese Vereinfachung entschieden habe.
Das Beispiel auf der rechten Seite der gleichen Abbildung stellt ein mögliches Argument dar. Thematisch bezieht es sich auf das Systemdesign für rekursive Workspaces mit der Frage, ob diese eher als Karten oder eher als Workspaces entwickelt werden sollten. Ausgehend von der Annahme, diese auch als Workspaces zu modellieren, wird gefolgert, dass dann genau eine zentrale Eigenschaft der Karten verloren geht, nämlich sichtbar über Kanten miteinander in Relation gesetzt werden zu können, was aber auch für rekursive Workspaces wünschenswert wäre. Diese Folgerung wird relativiert, indem die Option aufgezeigt wird, dass die Workspaces sowohl von der Klasse Karte als auch von der Klasse Workspace sein könnten.
Die visuelle Sprache zur Darstellung der Argumente ist eine Teilmenge aus der Explorationssprache (Fact, Result, Estimation, Support, Cause, Constrain). Das liegt daran, dass in der Explorationssprache Argumente bereits darstellbar sein sollen. Das bedeutet aber nicht, dass sich Argumente in anderen Perspektiven nicht anders präsentieren. Z.B. werden auch bestimmte Strukturen der Brainstormingperspektive als Argumente erkannt, die dort sehr unterschiedlich aussehen und erst interpretiert werden müssen, um sie in die Argumentperspektive aufnehmen zu können. Auch handelt es sich im Falle des Filterns von Argumenten aus einem Explorations-Workspace nicht um eine triviale Übernahme. Für eine weiterführende Diskussion wird jedoch auf das Kapitel 6.7 verwiesen.

6.3.7 Die visuelle Sprache für die Pro-/Kontratabellen

Tabellen können oft gut dazu genutzt werden, um Lücken sichtbar zu machen. In der Pro-/Kontraperspektive können Tabellen dargestellt werden in denen Pro- und Kontra-argumente gegenübergestellt werden. Jede der Tabellen soll einem strittigen Aspekt gewidmet werden.
Wie für die Argumente wird auch für diese visuelle Sprache davon ausgegangen, dass Inhalte für die Tabellen aus den Explorations-Workspaces automatisiert übernommen werden können und eine Grundlage für weitere Diskussionen bieten. Auch hier kann nach dem Filtern frei mit den Tabellen gearbeitet werden. Wie dieses Filtern aussieht wird erst später, als Beispiel der Strukturabbildung erläutert (Kapitel 6.7 und Kapitel 6.8).

Abbildung 40 zeigt das Schema einer Pro-/Kontratabelle. Intendiert sind zweispaltige Tabellen, die durch die jeweilige „Pro“- bzw. „Kontra“-Konnektorkarte übertitelt sind. Für die Darstellung werden einfache Formatierungskarten in Form von vertikalen und horizontalen Linien angeboten. Über der Tabelle kann jeweils der strittige Aspekt angegeben werden.

Aspect	Überschrift für eine Tabelle: Der Aspekt, der durch Pro- und Kontraargumente erörtert werden soll.
Contra	Container für Kontrabeiträge.
pro	Container für stützende Beiträge.
	Formatierungskarten.
PRO	Spaltenmarkierung der stützenden Argumente.
CONTRA	Spaltenmarkierung der Kontraargumente.

Tabelle 15

Die Elemente der visuellen Sprache für die Pro-/Kontraperspektive.

Bei der freien Arbeit ist die Platzierung ausreichend, um optisch eine Zusammengehörigkeit der Argumente zur Tabelle bzw. zum behandelten Aspekt zu verdeutlichen. Die automatisch generierten Tabellen sehen jedoch jeweils Relationen zwischen Argumenten und Aspekten vor, um die Zusammengehörigkeit auch systemseitig explizit zu gestalten.

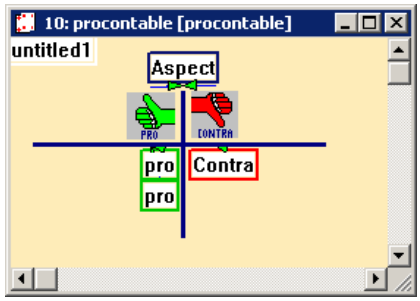


Abbildung 40

Schema der Tabellendarstellung.

6.4 Die Funktionspalette

In Kapitel 3.1 sind die drei Basis-Systeme für die Diskussionsunterstützung vorgestellt worden. Insbesondere das Kommunikationsinterface zwischen Dalis und dem CardBoard – vermittelt durch MatchMaker – wird für die Diskussionsunterstützung intensiv genutzt.

Im gleichen Kapitel wurde auch erläutert, dass das Interface im Wesentlichen auf den drei Nachrichten „create“, „delete“ und „modify“ basiert, über die mit zusätzlichen Parametern Informationen zwischen den Teilsystemen kommuniziert werden.
Diese Schnittstelle führt insofern zu Problemen, als dass sie sich ausschließlich auf Interaktionen auf den Objekten der visuellen Sprachen bezieht (Ausnahme bilden Daten über die Workspace selbst). Im Falle der für die Diskussionsunterstützung verwendeten Architektur wird ein Diskussionsmodell (Kapitel 6.8) in Dalis verwaltet. Dieses Modell steuert mehr als nur unmittelbare Reaktionen auf Karteninteraktionen in den Workspaces. Es hält eine Workspace-übergreifende Datenbasis vor, auf die auch beim Öffnen und schließen von Workspaces sowie für Sortierungen zugegriffen werden muss. Dies sollte an sich durch einen Dialog gesteuert werden. Auf Basis des vorliegenden Interfaces müssen solche Interaktionen jedoch auf Kartenoperationen abgebildet werden. Das bedeutet, dass Interaktionen, die auf die Daten in Dalis rekurrieren, immer in Form von Aktionen auf Karten ausgedrückt werden müssen, damit dies an Dalis übermittelt werden kann.
Um das zu erreichen, wurde eine Funktionspalette eingeführt. Die Funktionspalette ist selbst ein Workspace-Typ, für den eine eigene visuelle Sprache vorliegt. Die Karten werden jedoch nicht zum Aufbau von Graphen verwendet. Wird eine Karte dieser Palette in einen Workspace gezogen, so wird an Dalis eine entsprechende „create“-Mitteilung gesendet. Auf Basis des Kartentyps kann in Dalis eine vordefinierte Reaktion ausgelöst werden. Eine Rückmeldung von Dalis kann auf Kartenzustände und deren Vorhandensein Einfluss nehmen sowie auf die Workspaces selbst.
Abbildung 41 zeigt diese Palette. Die Karten darin bilden funktional drei Gruppen (Tabelle 16)

1. Verwaltung von Zustandsmarkierungen
2. Sortierungen
3. Funktionen zum Öffnen von Workspaces

Auf die Verwaltung der Zustandsmarkierungen wird vertieft in Kapitel 6.6 eingegangen und den Sortierungen widmet sich Kapitel 6.9, die Funktionen zum Öffnen werden im Folgenden erläutert:
Die Idee der Diskussionsunterstützung führt dazu, dass die Daten zu den Karten in den Workspaces persistent sein sollten, auch wenn diese während der Diskussion geschlossen werden. In dieser Situation liegen die wirklichen Objekte nicht mehr im DiscBoard vor und die Daten werden nur noch über Dalis vorgehalten. Es muss also ermöglicht werden,

die Daten in Dalis wieder anzufordern, wenn ein bestimmter Workspace erneut geöffnet werden soll. Dazu müssen zwei Aspekte gelöst werden:

- Der Workspace muss ausgewählt werden können und
- diese Auswahl muss, wie oben erläutert, über Aktivitäten auf Karten implementiert werden.

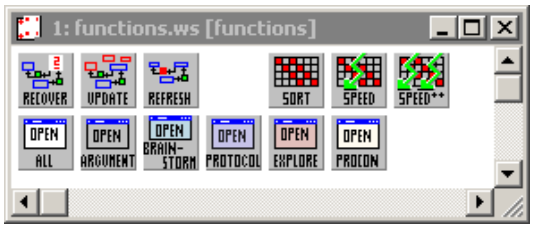


Abbildung 41

Die Funktionspalette.

RECOVER UPDATE REFRESH	Verwaltung von Zustandsmarkierungen
SORT SPEED SPEED**	Sortierungen
OPEN ALL, OPEN ARGUMENT, OPEN BRAIN-STORM, OPEN EXPLORE, OPEN PROTOCOL, OPEN PROCON	Funktionen zum Öffnen von Workspaces

Tabelle 16

Die Elemente der visuellen Sprache für die Funktionspalette.

Die Auswahl des Workspaces wird in Zusammenhang mit der „Topic Map“ realisiert (Kapitel 6.3.5). Die „Topic Map“ zeigt alle „Topics“ an, zu denen Workspaces bearbeitet wurden. Die Auswahl eines Workspaces erfolgt durch die Wahl des gewünschten „Topics“ in der „Topic Map“. Dazu wird eine Funktionskarte auf diesen „Topic“ gezogen. Um die Perspektive bestimmen zu können, unter der das Thema bearbeitet werden soll, steht für jede Perspektive eine eigene Funktionskarte zur Verfügung. Wird eine Funktionskarte zum Öffnen auf den entsprechenden „Topic“ gelegt, so bewirkt das, dass die Daten dazu aus Dalis angefordert werden. Auf Basis der Daten in Dalis wird dann ermittelt, welche Karte überdeckt wurde. Der Inhalt dieser Karte bestimmt dann das gewünschte Thema. Aus dem Typ der „Open“-Karte wird der Typ des angeforderten Workspaces erkannt, also die Perspektive. Daraufhin werden die angeforderten Daten übermittelt und der Workspace in der entsprechenden Sicht erzeugt.

Für dieses Vorgehen muss garantiert werden, dass jedes bis dahin bearbeitete Thema in der „Topic Map“ enthalten ist. Um das zu erreichen wird jedem Workspace erst einmal ein generierter (bedeutungsleerer) „Topic“ zugeordnet, der später angepasst werden kann. Diese „Topics“ werden automatisch in die „Topic Map“ übernommen, worin zusätzlich auch direkt neue Themen angelegt werden können.

6.5 Implementierungskonzept der multi-Perspektivik

6.5.1 Anforderungen und Ziele

Das DiscBoard implementiert einen Perspektivenansatz, der den Übergang zwischen den Perspektiven und den damit einher gehenden Arbeitsphasen unterstützt. Die eigentliche Perspektivik ergibt sich aus den verwendeten visuellen Sprachen und den darin darstellbaren Zusammenhängen und Beitragsarten, also den Darstellungsmitteln und nicht den konkreten Inhalten. Jeder Workspace der DiscBoard-Applikation ist einer visuellen Sprache zugeordnet und bietet damit eine bestimmte Perspektive an.

Die einzelnen Perspektiven stellen epistemische Formen zur Verfügung im Sinne von Collins und Ferguson (1993). Sie begleiten Diskussionsphasen gleichermaßen wie Diskussionsstränge, da den einzelnen Arbeitsbereichen Themen zugeordnet werden. Aus der Verwendung unterschiedlicher Perspektiven ergeben sich Perspektivenwechsel. Dieser Ansatz folgt damit den Forderungen aus Suthers und Xu (2002), dass es möglich sein muss, Artefakte – hier die einzelnen Diagramme – auf natürliche Art und Weise zu wechseln. Auch dort wird davon ausgegangen, dass dieser Wechsel des Artefaktes zu einem neuen Diskussionsstrang führt und dies explizit kenntlich gemacht werden sollte.

Mit dem hier vorgestellten Perspektivenansatz soll vor allem der Übergang zwischen den Perspektiven unterstützt werden. Die Übernahme, die hier nicht nur einzelne Beiträge, sondern auch zusammenhängende Strukturen betreffen kann, wird im weiteren als Abbildung bezeichnet. Geleistet wird dabei nicht nur die Duplizierung von Elementen, sondern auch eine Generierung von Knoten sowie Relationen.

Der Implementierung der Abbildungen liegen im Wesentlichen die folgenden Annahmen und Voraussetzungen zugrunde:

- Das systemgesteuerte Generieren von Knoten und Strukturen wirkt motivierend und koordinierend auf die Diskussion.
  - Die generierten Objekte bieten einen Ausgangspunkt für die weiterführende Elaboration. Sie dienen als Einstiegspunkt insbesondere dann, wenn durch besondere Strukturierungen Lücken in den Inhalten deutlich werden.
  - Automatisch übernommene Beiträge oder Strukturen haben eine Koordinationsfunktion bei der Diskussion, da durch sie deutlich werden kann, an welchen Punkten zusätzliche Diskussionen sinnvoll wären.
  - Eine Grundlage für weiterführende Diskussionen entsteht insbesondere dadurch, dass Strukturen, die aus anderen Perspektiven extrahiert wurden, automatisch

angeordnet werden. Es werden dadurch Muster angeboten, die strukturelle Aspekte hervorheben.

- Durch die Vorgaben soll verhindert werden, dass relevante Aspekte übersehen werden.
- Durch die Abbildung der Karten soll Zeit und unnötiger Aufwand gespart werden, denn es ist davon auszugehen, dass sonst mühsam die Objekte herausgesucht werden müssen, die noch nicht diskutiert wurden. Gerade in größeren Diskussionen, in denen gar nicht alle Arbeitsbereiche zu überblicken sind, ist das ohne Systemunterstützung beinahe unmöglich. Durch das Löschen der Systemvorschläge können diese übrigens leicht ignoriert werden. Das entspricht der Designidee, nämlich den Prozess nicht vorzuschreiben sondern flexibel anzuregen.

- Die Inhalte der Perspektiven entwickeln sich unterschiedlich, sie divergieren.
  - Im Sinne einer „representational guidance“ wird davon ausgegangen, dass sich die unterschiedlichen Perspektiven dazu eignen, verschiedene Inhalte hervorzulocken und zu entsprechend spezifischen Ergebnissen führen. Da die Perspektiven so gestaltet sind, dass es nicht sinnvoll ist, jeden Aspekt unter allen Perspektiven zu besprechen, ergänzen sich die Inhalte der Perspektiven eher, als dass sie sich überlagern. Entsprechend gibt es keine „Integrationsperspektive“, die alle Inhalte aus allen anderen Perspektiven konsistent zusammenfasst.
  - Die Übernahme von Inhalten aus anderen Perspektiven, stellt eine Diskussions*grundlage* dar. Es kommt zwar zu einer vorübergehenden Übereinstimmung von Teilen der inhaltlichen Beiträge zwischen den Perspektiven aber im Anschluss können sich die Inhalte unterschiedlich entwickeln. Im Verlauf der weiteren Diskussion divergieren die Inhalte in den Perspektiven.
  - Solange Inhalte von Knoten, die abgebildet wurden, noch übereinstimmen, wird diese Zusammengehörigkeit vom System verwaltet. Diese Zusammengehörigkeit muss bei der Nutzung einfach aufgehoben werden können.
- Es wird davon ausgegangen, das nach Themen diskutiert wird.
  - Die Zuordnung erörterter Inhalte zu Themen soll die inhaltliche Strukturierung der Diskussion forcieren.
  - Das teilweise Generieren einer „Topic Map“ aus den Themen, die einzelnen Workspaces und damit auch Gesprächsphasen zugeordnet werden, soll die Übersicht über die Diskussion verbessern.

  - Zu einem „Topic“ unter einer Perspektive kann nur ein Workspace geöffnet werden. Alle Beiträge dazu sollen in diesem einen Workspace platziert werden. Werden mehrere Workspaces mit der gleichen visuellen Sprache geöffnet, sind diese zwingend unterschiedlichen Themen zuzuordnen.
- Um einen Arbeitsprozess anregen und unterstützen zu können, muss Wissen über die Diskussionsabläufe im System repräsentiert werden. Das Diskussionsmodell definiert eine konkrete Annahme über Übergänge zwischen Diskussionsphasen.
  - Für die Diskussion wird davon ausgegangen, dass es zwar unterschiedliche Diskussionsphasen gibt, diese aber keiner definierten Reihenfolge unterliegen.
  - Das Diskussionsmodell definiert, welche Kartentypen oder Strukturen in andere Perspektiven abgebildet werden sollen.
  - Die Abbildungen der Karten und Strukturen beziehen sich auf die Typen der Karten und Relationen, nicht auf deren Inhalt. Das Diskussionsmodell definiert die Abbildungen in allgemeinen Regelschemata, die auf die „semantische Idee“, den Typ der Sprachprimitive, Bezug nehmen.

6.5.2 Konzept der Implementierung

Bereits vorgestellt wurden die visuellen Sprachen, die die Perspektiven bilden. Das Konzept zur Verwendung multipler Perspektiven sieht vor, dass diese nicht einfach nur abwechselnd verwendet werden können, sondern dass teilweise Informationen zwischen ihnen transferiert werden, die dabei potentiell strukturellen (topologischen) Veränderungen und geänderten Ordnungskriterien unterliegen. Abbildung 42 stellt diesen Vorgang schematisch dar: Für die Daten des grafischen Editors wird eine jeweils aktuelle Zustandsbeschreibung protokolliert. Auf Basis dieser Daten arbeiten die Produktionsregeln im Diskussionsmodell, die daraus Informationen ableiten und diese wieder an den Editor übermitteln.

Der Editor einschließlich der konkreten visuellen Sprachen ist Bestandteil des DiscBoards. Die Zustandsbeschreibung und das Diskussionsmodell werden in Dalis verwaltet. Das Diskussionsmodell besteht aus den Definitionen für die Abbildungen von Strukturen und Beiträgen zwischen den Perspektiven. Die Zustandsbeschreibung bildet die Datenbasis, die den Diskussionsstand vorhält. Dieser bezieht sich auf alle erarbeiteten Inhalte einer Diskussion, also auch die Daten für Karten und Strukturen, die gerade nicht angezeigt werden aber bereits erarbeitet wurden.

Das Zustandsmonitoring der aktuell angezeigten Daten ist eine Grundfunktion von Dalis und in Mühlenbrock (2001) ausführlich beschrieben.

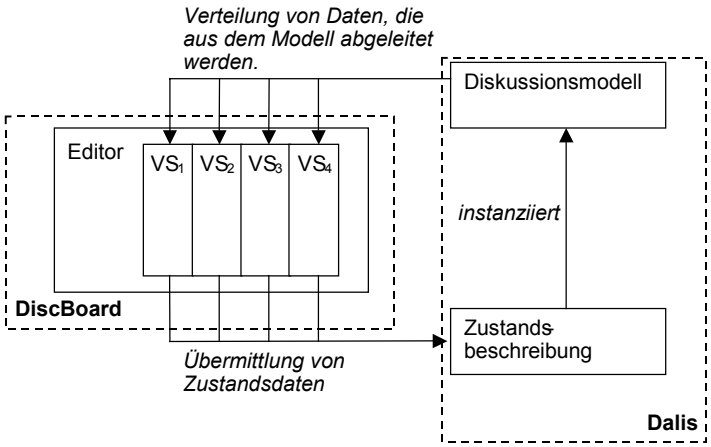


Abbildung 42

Datentransfer zwischen verschiedenen Systemkomponenten.

Struktur- und Knotenmodelle

Für die Diskussionsunterstützung müssen Informationen über die Übereinstimmungen zwischen den Perspektiven verwaltet werden. Diese Übereinstimmungen entstehen durch definierte Abbildungen zwischen den Perspektiven, die auf Systeminitiative hin vorgenommen werden. Abgebildet werden einzelne Beiträge und Teile der Diagramme, im Weiteren als Strukturen bezeichnet, die durch die Nutzerinnen und Nutzer im Verlauf einer Diskussion erstellt werden.

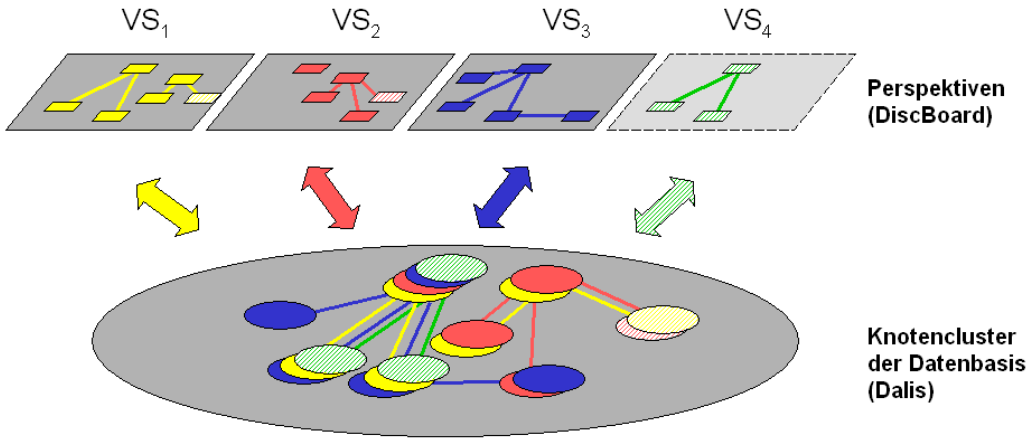


Abbildung 43

Über die Perspektiven wird die Datenbasis aus der DiscBoard-Applikation heraus modifiziert.

Abbildung 43 stellt schematisch die Trennung zwischen der Datenbasis auf Seite von Dalis und den Perspektiven im DiscBoard dar. Vier Workspaces, die unterschiedliche Perspektiven VS_1-VS_4 symbolisieren, sind dort in der oberen Ebene rautenförmig

dargestellt. Durch die Farben wird angedeutet, dass die Perspektiven jeweils Ausschnitte aus der Datenbasis zeigen.
Bei der Abbildung einzelner Beitragskarten werden die Inhalte dieser Karten mit in die neue Perspektive übernommen. Solange diese Inhalte nicht geändert werden, wird zwischen diesen Karten eine Abhängigkeit verwaltet: Dafür werden als Knotencluster bezeichnete Datensätze erstellt und gewartet.

> **Knotencluster** sind Datensätze der Datenbasis in Dalis mit denen die Zusammengehörigkeit von Knoten verwaltet wird. Alle Knoten eines Knotenclusters haben den gleichen Inhalt aber im Allgemeinen einen anderen Typ. Zu jeder Perspektive darf sich nur ein Knoten in einem Knotencluster befinden (siehe auch Folgekapitel 6.6).

Abbildung 43 symbolisiert auf der unteren Ebene solche Knotencluster für die Perspektiven. Jeweils die übereinander gelegten kleinen Ovale sind zusammen als Cluster zu verstehen. Farblich gekennzeichnet ist, aus welchen der vier Perspektiven Karten zu diesem Cluster gehören. Die durchbrochen umrandete Perspektive VS_4 mit den gemusterten Objekten deutet an, dass eine Perspektive nicht unbedingt angezeigt wird, trotzdem aber die Daten für die Knotencluster in der Datenbasis vorgehalten werden.
Die Abbildungen beziehen sich nicht nur auf nur einzelne Beitragskarten, sondern auch auf Strukturen. Beim Abbildungsvorgang werden dazu Strukturmodelle erstellt und in die Datenbasis aufgenommen.

> **Strukturmodelle** verwalten die Informationen zu erkannten Teilstrukturen. Sie setzten sich zusammen aus Kantentypen und Knotenclustern, die durch diese Kantentypen verbunden werden und der jeweiligen Perspektive, für die diese Information zutreffen soll.

Die Informationen, die in Dalis über die Abhängigkeiten von Karten und Strukturen verwaltet werden, müssen unabhängig von den konkreten visuellen Objekten sein, da Workspaces geschlossen werden können und so diese konkreten Objekte nicht mehr existieren. Deshalb wird hier auch von Strukturmodellen bzw. Knotenmodellen gesprochen. Über sie muss eindeutig definiert sein, welche Karten und Relationen zu erstellen sind, wenn ein bestimmter Workspace angefordert wird.
Die Datensätze können grob in drei Gruppen unterteilt werden (Abbildung 44): diejenigen, die die Modellinformationen, also die Abhängigkeiten verwalten (innerhalb der Raute in der Datenbasis), die, die die Referenz zu Oberflächenobjekten beifügen

sobald diese existieren (hinterlegt durch den gelben Pfeil) und drittens allgemeine Information, die beispielsweise Zugriffe auf Daten beschleunigen.

Abbildung 44 stellt die konkret verwendeten Prädikate vor. Zentral sind die Prädikate *cluster* und *structure* für die Modellinformationen der Strukturen und Knotencluster. Die Verbindung zur Visualisierung wird durch die Prädikate *in_structure*, *map* und *thema* geleistet. Sie verwalten für jede Karte, ob sie zu einer Struktur oder zu einem Cluster gehört und zu welchen. Den existierenden Workspaces werden Themen zugeordnet.

Die Prädikate *hash_topics* und *topic2clusters* sind für die Verwaltung von „Topic"-Informationen zuständig. Die „Topics" gehen nur in Form eines Identifikators in andere Fakten der Datenbasis ein, um so deren Änderung auf eine Stelle zu begrenzen (*hash_topics*). Nur aus Effizienzgründen existiert das Prädikat *topic2clusters*, dass für jedes Thema verwaltet, in welchen Knotenclustern es erwähnt wird. So müssen beim Generieren von Karten zu einem Thema nicht alle Knotencluster durchsucht werden, sondern es können die relevanten zielgerichtet angesprochen werden. Prinzipiell ist es möglich, dass sich ein Knotencluster über unterschiedliche Themen erstreckt. Insbesondere wird diese Option für die Übernahme der „Topics" in die „Topic Map" verwendet.

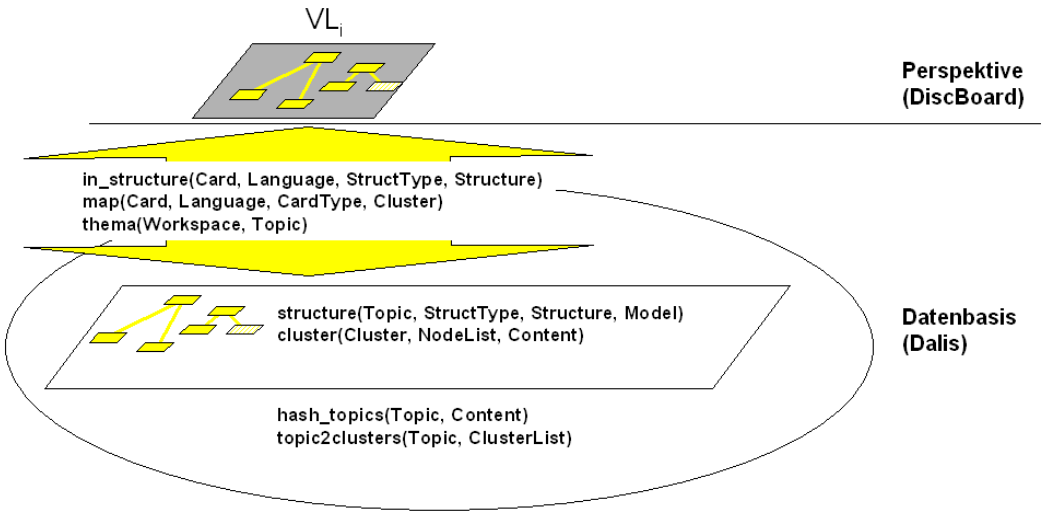


Abbildung 44

Einträge der Datenbasis.

User-gesteuerte Inhaltspropagierung („delayed changes")

Werden Karten abgebildet, so besteht eine Abhängigkeit der Inhalte zwischen den Karten, die auseinander hervorgegangen sind. Wenn nun der Inhalt einer Karte, für die ein solcher Zusammenhang zu anderen besteht, unter einer Perspektive geändert wird, so bestehen

zwei Möglichkeiten, damit umzugehen. Entweder werden die Inhalte sofort angeglichen oder später, unter der Entscheidung der Nutzerinnen und Nutzer. Für das DiscBoard wurde entschieden, dass diese Änderungen nicht sofort in die dazugehörigen Karten propagiert werden. Ein Grund dafür ist die Annahme, dass trotz der Abhängigkeit die Karten in den verschiedenen Perspektiven in unterschiedliche Kontexte eingebettet sind. Eine Inhaltsänderung kann also wegen der Perspektivänderung zustande kommen. Dies bedeutet dann nicht unbedingt, dass die vorangegangenen Inhalte selbst nicht mehr relevant sind. Deshalb wurde die Entscheidung getroffen, den Nutzerinnen und Nutzern die Änderungen offen zu lassen. Dafür werden die Änderungen in Form von Zustandsmarkierungen angezeigt. Dieses Vorgehen wird als „delayed changes" bezeichnet.
Es sind aber nicht nur die Änderungen von Karten durch diese Design-Entscheidung betroffen, sondern auch das initiale Generieren von Karten. Würden diese neuen Karten gemäß der Abbildungsvorschriften sofort auch unter den anderen Perspektiven angezeigt werden, so würde dies dort relativ unmotiviert das „optische Erscheinungsbild" maßgeblich beeinflussen. Es ist davon auszugehen, dass das zur Verwirrung der Nutzerinnen und Nutzer führt. Auch hier wird deshalb nach dem Prinzip der „delayed changes" vorgegangen, das heißt, dass zusätzliche Karten explizit angefordert werden müssen.
Dieses Prinzip wird als besonders wichtig für den Fall von Kartenlöschungen angesehen. Automatisches Löschen führt zum Verdacht, dass Inhalte verloren gegangen sind, was beim automatischen Aktualisieren auch wirklich geschehen kann. Löschungen von Workspace-Inhalten finden deshalb ebenfalls zu einem von Usern definierten Zeitpunkt statt und werden ebenfalls durch „Marker" angezeigt.
Das Konzept des „delayed deletion" wird auch in Stefik et al. (1987) eingesetzt. Dort wird argumentiert, dass widersprüchliche Inhalte gerade auch eine Diskussion anregen können und erst einmal nicht gelöscht werden sollten.

Verwendung struktureller statt räumlicher Eigenschaften

In Ansätzen zum formalen Schließen auf visuellen Darstellungen wird oft davon ausgegangen, dass der Ort der Objekte und deren räumliche Relation zueinander (z.B. neben, über, berührt, etc.) eine Rolle spielen und dass deshalb solche Informationen mit in die formale Repräsentation des Zustandes aufgenommen werden müssen.
Wang und Lee (1993) und Wang, Lee und Zeevat (1995) argumentieren, dass diese räumlichen Informationen nur dann sinnvoll verwendet werden können, wenn in diese bereits Wissen über die Domäne einfließt, in der Schlussfolgerungen angestellt werden sollen. Das liegt insbesondere daran, dass bestimmte Relationen nur für bestimmte

Domänen gelten. Beispielsweise kann ein Flaschenzugproblem visualisiert werden und die Objekte, aus denen die konkrete Kombination aus Flaschenzügen zusammengesetzt werden, können als Seile mit den jeweiligen Verbindung, Rollen und Gewichte repräsentiert werden (Beispiel aus Larkin und Simon, 1987). Eine anderes Beispiel besteht in der diagrammatischen Darstellung von Syllogismen (Wang, Lee & Zeevat, 1995; Stenning & Oberlander, 1995). Aber auch allgemeinere Eigenschaften wie „Nähe" oder „Farbe" mit potentiellen Schlussfolgerungen müssen im Allgemeinen anwendungsabhängig interpretiert werden.

Insofern spielen beim visuellen Schließen nicht unbedingt allgemeine räumliche Parameter eine Rolle, sondern domänenspezifische Aspekte einer zweidimensionalen Darstellung.

Im Zusammenspiel von Dalis und DiscBoard werden die Zustandsdaten an Dalis übermittelt und dort überwacht. Diese Zustandsdaten enthalten auch die Objektkoordinaten und –dimensionen (Abbildung 45). Auf dieser Basis können dort auch Schlüsse implementiert werden, die diese räumlichen Daten ausnutzen. Im Rahmen der Diskussionsunterstützung wird aus diesen Informationen bisher jedoch noch keine Bedeutung abgeleitet. Ausgenutzt werden sie nur für die automatische Anordnung der Objekte. Insofern können die Abbildungen als topologische bzw. strukturelle Schlüsse charakterisiert werden!

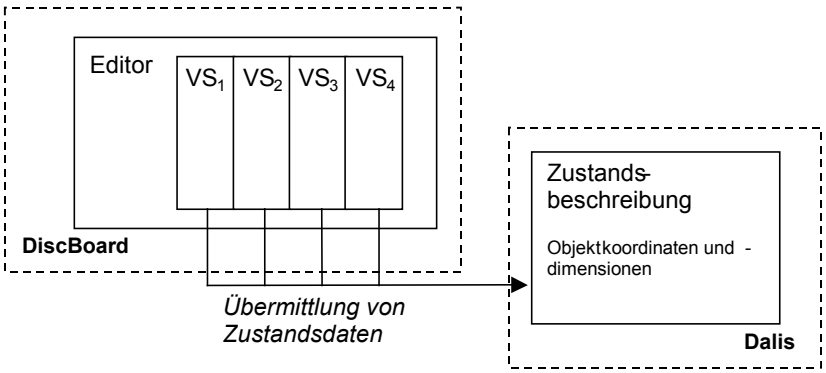


Abbildung 45

In den Zustandsdaten in Dalis werden auch Objektkoordinaten und –dimensionen vorgehalten.

Lokales Parsen: Vermeidung globalen Parsens

Unter globalem Parsen ist hier zu verstehen, dass die gesamte Darstellung überprüft bzw. geparst wird. Die Ziele eines solchen globalen Parsens können beispielsweise darin bestehen, Schlussfolgerungen aus dieser Darstellung zu ziehen oder sie in eine andere

Darstellung zu transformieren. Das globale Parsen ist im Normalfall allerdings sehr zeit- und rechenaufwändig und soll für die Diskussionsunterstützung vermieden werden.
Ein pragmatischer Grund dafür liegt auch in der Art, wie diese Strukturen entstehen. Sie werden allmählich, während der Arbeit, entwickelt. Beispielsweise kann ein Argument mehrere Voraussetzungen haben, die erst nach und nach hinzugefügt werden. Das heißt, dass nach sehr vielen Interaktionen im Workspace eine Strukturüberprüfung durchgeführt werden muss, die deshalb nicht übermäßig aufwändig sein sollte.
Der zentrale Grund für das lokale Parsen liegt jedoch darin, dass die bedeutungstragenden Strukturen gerade als Teile der Gesamtdarstellung definiert sind. Da es sich bei den bedeutungstragenden Strukturen, um verbundene Graphen handelt, kann eine neue Struktur oder eine Erweiterung einer bestehenden immer nur entstehen, wenn eine Relation eingefügt wird. Wird ein solcher Teil identifiziert, ausgehend von einer Benutzerinteraktion, braucht die restliche Darstellung nicht mehr geprüft werden. Das lokale Parsen geht deshalb von einem Objekt aus, konkret einem Link als Bestandteil einer Relation, um ein verändertes Diagramm auf die Entstehung einer Struktur zu untersuchen.
Die Ausdrücke der visuellen Sprache brauchen nicht als Ganzes geparst werden, da sie in der Diskussionsunterstützung im Prinzip immer als korrekt anzusehen sind, denn alle Objekte der visuellen Sprache dürfen ohne Einschränkungen verwendet werden.

Klasse der erkennbaren Strukturen

Für die zu erkennenden Strukturen können in Bezug auf die Anzahl von Karten und Relationen keine eindeutigen Definitionen angegeben werden, wohl aber für die Art und Weise, wie diese Strukturen gebildet werden. Im Rahmen dieser Arbeit wurde ein Algorithmus zum lokalen Parsen entwickelt, der Strukturen erkennen kann, die auf die folgende Art und Weise aufgebaut sind: Diese Strukturen teilen sich auf in einen Strukturkern, das ist die minimale Struktur, und in Erweiterungen. Die Art des Strukturkerns sowie die Art der einzelnen Erweiterungen wird jeweils als Einheit beschrieben. Der Aufbau der gesamten Struktur findet dann nach und nach statt. Darauf basiert ein inkrementelles Vorgehen bei der Erkennung und Wartung von Strukturen.
Die Strukturarten, die damit erkannt werden können, werden im Folgenden erst einmal unabhängig von einer konkreten visuellen Sprache erläutert. Die Beschreibung der konkreten Strukturen folgt in den nächsten Kapiteln.

> **Ausdruck in einer visuellen Sprachen.** Sei VS eine visuelle Sprache mit einer Menge $Label_{N\_VS}$ von Knotenarten und einer Menge $Label_{E\_VS}$ von

Kantenarten. Die Funktionen φ_{N_VS} und φ_{E_VS} liefern zu jedem Knoten bzw. jeder Kante das entsprechende Label. Sei weiterhin N_{VS} die Menge der Knoten, E_{VS} die Menge von Kanten eines Graphs mit $E_{VS} \subseteq N_{VS} \times N_{VS}$. Ein Graph ist definiert als

$$G_{VS} = (N_{VS}, E_{VS}, Label_{N\_VS}, Label_{E\_VS}, \varphi_{N\_VS}, \varphi_{E\_VS})$$

und wird als Ausdruck in einer visuellen Sprache bezeichnet.

Die Definition für einen Ausdruck in einer visuellen Sprache soll verdeutlichen, dass im hier entwickelten Ansatz alle Ausdrücke in allen visuellen Sprachen als Graphen repräsentiert sind. Die Bezeichnung „Label“ wird hier neu eingeführt und entspricht dem Typ oder der Art von Kanten und Knoten, wie sie im Vorfeld verwendet wurden. Während „Typ“ eher auf eine Bedeutung verweist, verdeutlich „Label“, dass in dieser und den folgenden Definitionen vor allem auf die Syntax Bezug genommen wird und den Knoten und Kanten ein Name zugeordnet wird.

Erkennbare Struktur. Eine Instanz einer erkennbaren Struktur besteht aus genau einem Strukturkern und beliebig vielen Extensionen. Der Strukturkern sowie alle Arten von Extensionen müssen jeweils für sich als Einheit definiert werden. Eine erkennbare Struktur ist immer ein echter oder unechter Teilgraph eines Ausdruckes einer visuellen Sprache, kann sich also nicht über mehrere visuelle Sprachen erstrecken. Es gilt immer:

$N_{VS\_Struktur} \subseteq N_{VS}$, $E_{VS\_Struktur} \subseteq E_{VS}$,
$Label_{N\_VS\_Struktur} \subseteq Label_{N\_VS}$, $Label_{E\_VS\_Struktur} \subseteq Label_{E\_VS}$.

Im Folgenden werden die Bildungsvorschriften für Strukturkerne und Extensionen sowie deren Kombination anhand von kurzen Graphgrammatiken erläutert.
Abbildung 46 stellt die Grammatik für Strukturkerne vor, Abbildung 48 die Grammatik für die Extensionen sowie deren Kombination mit den Kernen.
Die in Abbildung 46 dargestellten Produktionen 1-3 für Strukturkerne verdeutlichen, dass diese kaum beschränkt sind, sondern beinahe beliebige Graphen zu einer visuellen Sprache sein können. Die einzige Einschränkung besteht darin, dass der „kleinste Strukturkern“ aus mindestens zwei Knoten und einer Kante bestehen muss. Regel 1 stellt diese Beschränkung dar, indem aus dem Startsymbol G_{Kern} gleich ein solcher Kern hervorgeht. Regel 2 ermöglicht die Erweiterung der Strukturen um weitere Knoten, die entweder über eine ausgehende oder eine eingehende Kante mit dem Ausgangsgraphen

verbunden sein müssen. Regel 3 ermöglicht, die aus Regel 1 und 2 entstehenden Ketten aus Knoten und Kanten, auch untereinander zu verbinden.

Als Einbettungsregel für die hier eingeführte Graphgrammatik gilt, dass alle Kantenverbindungen, die vorher zu einem Knoten bestanden haben, nach der Ersetzung wieder hergestellt werden.

Auch wenn für jeden Strukturkern eine Grammatik angegeben werden kann, so werden die Kerne sowie die Extensionen später im Diskussionsmodell nicht über eine solche Grammatik definiert, sondern als Menge von Relationen- und Knotenarten. Der Grund besteht vor allem darin, dass die tatsächlich anfallenden Strukturen, denen eine Bedeutung zuerkannt wird, sehr einfach sind. Beispielsweise kann die Grammatik für den Strukturkern einer Argumentstruktur wie in Abbildung 47 angegeben werden.

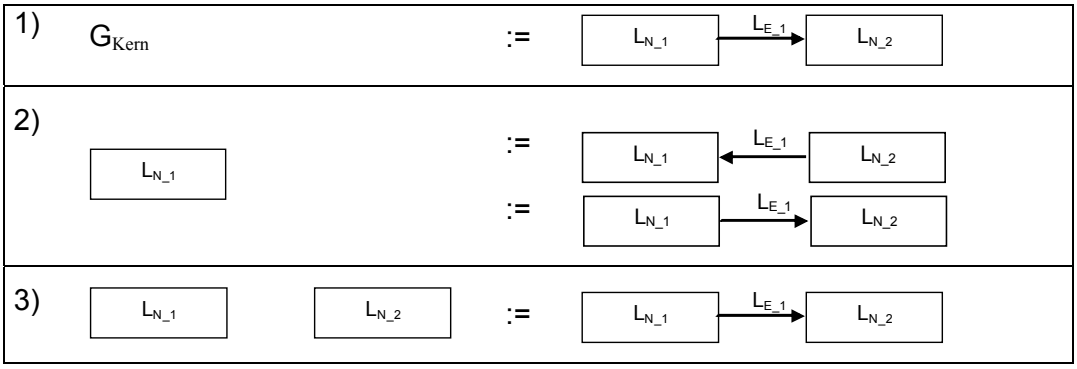


Abbildung 46

Eine Graphgrammatik für Strukturkerne, in der zwischen erweiterbaren und nicht erweiterbaren Knoten unterschieden wird. Ansonsten ist jeder gerichtete Graph ein zulässiger Strukturkern.

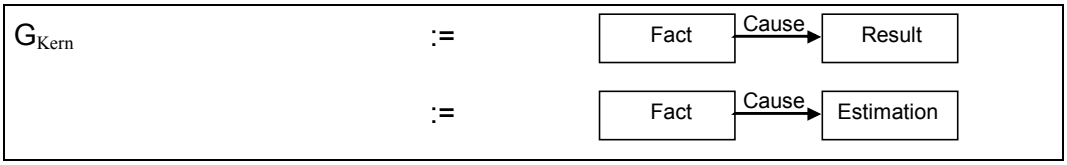


Abbildung 47

Grammatik für die Bildung zweier Ausprägungen des Strukturkernes für eine Argumentstruktur.

Abbildung 48 beinhaltet die Erweiterungsregeln des Kerns. Relevant ist darin die Unterscheidung in die rechteckigen und ovalen Knoten. Die rechteckigen Knoten sind noch Bestandteil des Strukturkerns, während die ovalen Bestandteil der Extension sind. Die durchbrochen gezeichneten Knoten des Kerns sind explizit ausgezeichnete Knoten, an die die Extensionen angebunden werden können. Somit beschreiben die Regeln 4 und 5, wie Extensionen an den Kern angebunden werden können und die Regeln 6 und 7 beschreiben den Aufbau der Extensionen selbst. Regeln 6 und 7 entsprechen den Regeln

2 und 3, über die wiederum allgemein ein gerichteter Graph aufgebaut werden kann. Sie sind hier hauptsächlich aus Gründen der Vollständigkeit noch einmal aufgeführt und zeigen, dass auch die Extensionen in sich keinen Beschränkungen unterliegen.

Im Prinzip kann jeder Knoten eines Strukturkernes erweitert werden. Welche konkret erweitert werden dürfen, wird über die Regeln für die Extensionen definiert (Regel 4). Deshalb befindet sich diese Regel in der Grammatik für die Extensionen und nicht in dem Teil für die Kerne. Die Erweiterbarkeit bezieht sich ausschließlich auf den Typ des zu erweiternden Knotens, spezifiziert durch das jeweilige Label. Für eine konkrete Struktur müssen die jeweiligen Label konkretisiert werden.

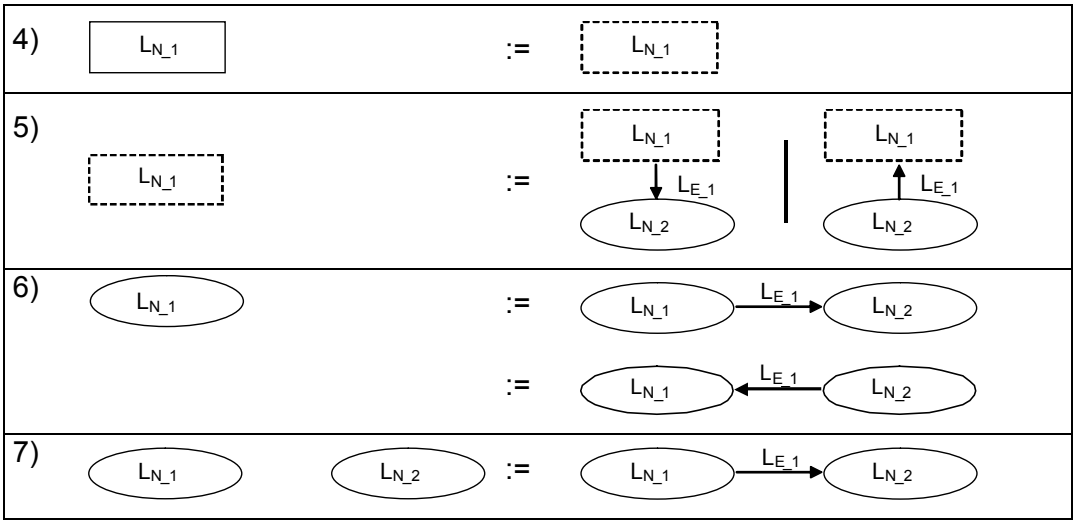


Abbildung 48

Die Grammatik für die Extensionen mit der Anbindung an die Strukturkerne.

Zu bemerken ist, dass Regel 5 mehrfach angewendet werden kann. Es können also sowohl mehrere Extensionen mit einem Knoten des Kerns verbunden werden, als auch eine Extension über mehrere Relationen.

Dadurch, dass nur die Strukturkerne erweitert werden können, denn in der Grammatik besteht keine Erweiterungsregel für die Extensionen, werden keine rekursiven Erweiterungsmöglichkeiten erlaubt. Das heißt, dass keine Extensionen von Extensionen definiert werden können. Die Extensionen können zwar als Klasse einen beliebigen Graph bilden, werden dann aber als „Ganzes“ betrachtet. Das heißt auch, dass der Parser keine rekursiven Extensionen erkennt. Der Grund dafür besteht in der konkreten Nutzung für die Diskussionsunterstützung, die keine solchen komplexen Strukturen verwendet.

Abbildung 49 zeigt ein Beispiel für Grammatikregeln für die Argumentstruktur. Darin wird das Beispiel aus Abbildung 47 aufgegriffen und der„Result“-Knoten des Argumentkerns erweitert.

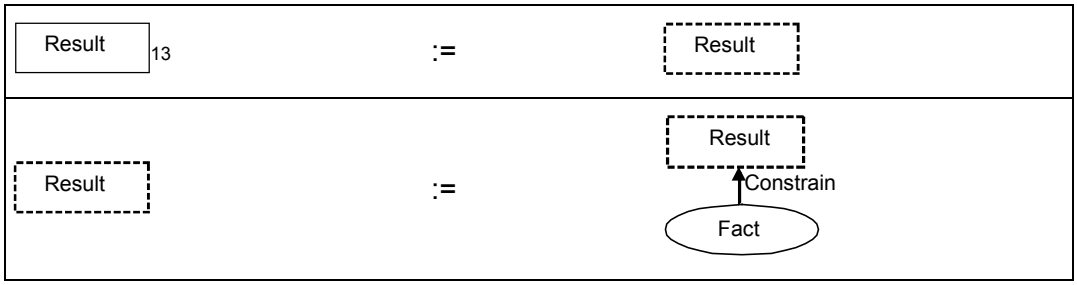


Abbildung 49

Extension des Strukturkerns der Argumentstruktur über einen Knoten vom Typ „Result“.

Abbildung 50 veranschaulicht mögliche Topologien, die mit der Implementierung zum lokalen Parsen erkannt werden könnten, hier auch komplexere als solche, die für die Diskussionsunterstützung definiert sind. Die Strukturkerne sind jeweils rot hervorgehoben. In den Beispielen a)-c) sind nur solche Extensionen enthalten, die jeweils aus einer Relation bestehen. Der aus zwei Knoten bestehende Strukturkern in Beispiel a) wurde mit 4 Extensionen erweitert.

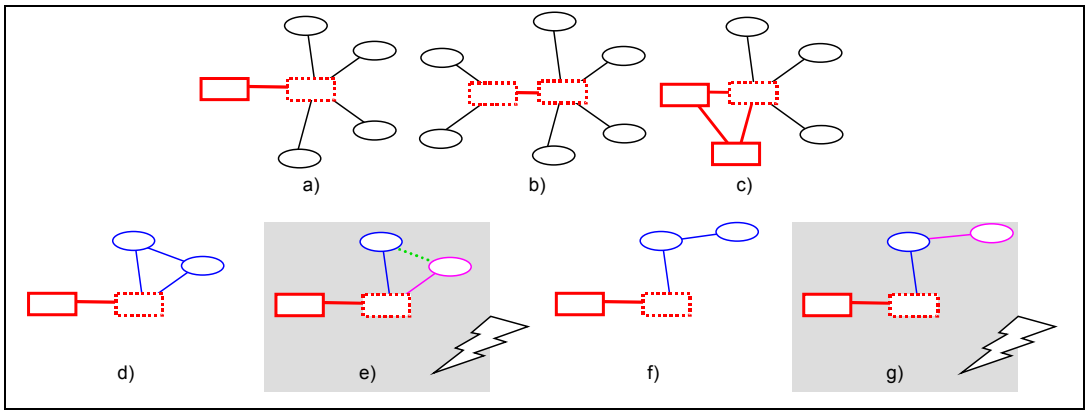


Abbildung 50

Mögliche Topologien für Strukturen, die mit der Implementierung zum lokalen Parsen erkannt werden könnten. Die roten Bestandteile symbolisieren jeweils den Strukturkern, die anderen die Extensionen.

Beispiel c) beinhaltet einen Kern, der aus drei Knoten und drei Relationen besteht. Beispiel d) zeigt eine Extension (blau), die mit dem Kern über zwei Relationen verbunden ist. Nicht definierbar ist die Struktur aus Beispiel e)! Darin ist ein Knoten des Kerns durch zwei Extensionen erweitert worden (angedeutet durch unterschiedliche Farben). Eine Verbindung dieser Extensionen (grün, durchbrochen) kann im vorliegenden Ansatz nicht definiert und nicht erkannt werden. Beispiel f) zeigt ein Beispiel mit einer Extension, die aus zwei Knoten und zwei Relationen besteht (blau). Im Gegensatz kann

[13] Werden im Strukturkern mehrere Knoten gleichen Typs eingebunden und ist außerdem die Position des Knotens in der Topologie relevant ist, so müssen die linken Seiten dieser Regeln den Strukturkern darstellen.

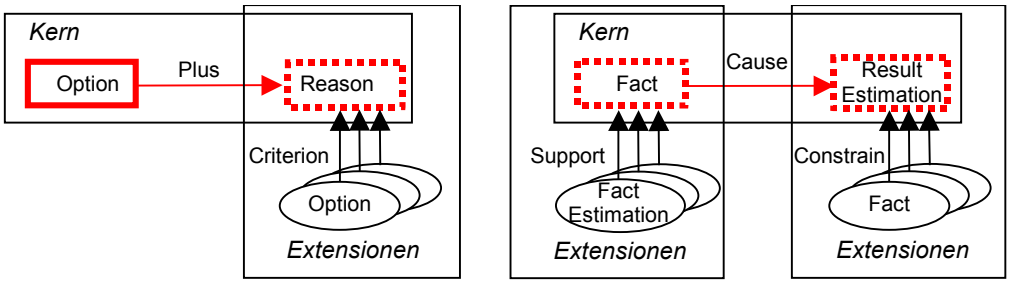
das Beispiel g) nicht definiert und erkannt werden. Der Graph ist zwar topologisch identisch zu f) aber es werden, angedeutet wieder durch die unterschiedlichen Farben, zwei Ausdrücke aneinander gereiht.

Multiple Perspektiven: Die Abbildungen

Die Diskussionsunterstützung operiert auf den oben beschriebenen lokalen Strukturen, indem diese zwischen den Perspektiven abgebildet werden. Im vorangegangenen Abschnitt wurde der Aufbau der erkennbaren Strukturen erläutert. Bei einer Abbildung von Strukturen ist nun außerdem der Typ dieser Strukturen relevant, da pro visueller Sprache mehrere Strukturtypen spezifiziert werden können.

> Ein **Strukturtyp** T ist durch eine bestimmte Topologie charakterisiert, die in einer Strukturgrammatik ausgedrückt werden kann. Für eine visuelle Sprache VS darf ein Strukturtyp T nur über genau eine Bildungsvorschrift definiert sein. Ist ein Strukturtyp T für mehrere visuelle Sprachen VS_i definiert, so sind die Topologien bzw. deren Bildungsvorschriften unabhängig voneinander.

Die Argumentstruktur ist ein Beispiel für einen Strukturtyp. Sie ist für die Explorationssprache und die Brainstormingsprache definiert, in denen sich das Argument jeweils in einer anderen Topologie darstellt. Abbildung 51 stellt diesen Strukturtyp dar und greift dazu die Darstellungsform aus Abbildung 50 auf. Die Form und Farbe der Knoten bezieht sich entsprechend nicht auf die Typen der Objekte der visuellen Sprache, sondern kennzeichnet ihre Zugehörigkeit zur Extension oder zum Kern.



Der Strukturtyp „Argument“ in der Brainstormingsprache

Der Strukturtyp „Argument“ in der Explorationssprache

Abbildung 51

Der Strukturtyp „Argument“.

Die Abbildung einer Struktur unterteilt sich in zwei Phasen:

- Erkennen einer Struktur und deren Typs.

- Abbildung der Struktur auf Ebene der internen Modelldarstellungen.

Die spätere Darstellung der abgebildeten Strukturen ist nicht mehr Bestandteil der eigentlichen Abbildung. Ein Strukturtyp muss explizit nur für die visuellen Sprachen definiert sein, aus denen Strukturen diesen Typs abgebildet werden sollen. Nur dort müssen die Strukturen identifiziert werden. In der Zielsprache einer Abbildung sind die Strukturtypen nur implizit definiert, indem die Modelle einem Typ zugeordnet werden. Bei einer konkreten Abbildung wird die neu generierte Struktur dem gleichen Strukturtyp zugeordnet, der bereits für das Urbild zutraf. Der Strukturtyp wird also bei einer Abbildung beibehalten.

Abbildungen von Strukturen werden inkrementell durchgeführt. Das bedeutet einerseits, dass sie schrittweise durchgeführt werden, beginnend mit dem Strukturkern. Werden dann Extensionen erkannt werden diese jeweils in einem weiteren Schritt zusätzlich abgebildet. Andererseits bedeutet inkrementelles Abbilden hier auch, dass das schrittweise Erkennen von Bestandteilen einer Struktur zur deren Ergänzung führt und nicht die gesamte Struktur neu generiert wird. Die neuen Bestandteile werden dem bestehenden Modell hinzugefügt.

> **Abbildung eines Strukturkerns.** Sei S_{VSx_T} ein Strukturkern in einer visuellen Sprache VS_x zu einem Strukturtyp T, dann führt eine Abbildung dieser Struktur in eine visuelle Sprache VS_y zur Generierung, das heißt zum Anlegen eines Modells einer Struktur gleichen Typs T in der visuellen Sprache VS_y, wobei $x \neq y$ gelten muss:
>
> $$a: S_{VSx\_T} \rightarrow S_{VSy\_T}$$

> **Modell einer Struktur.** Eine Relation r ist in einem Strukturmodell beschrieben durch den Typ zweier Knoten, $Label_{N1\_VS}$ und $Label_{N2\_VS}$, eine Funktion c: $Label_{N\_VS} \rightarrow$ Content, die den Inhalt der Knoten liefert und den Typ der Relation: $r_{VS} = (Label_{N1\_VS}, Label_{N2\_VS}, c_{N\_VS}, Label_{E\_VS})$. Sei T ein Strukturtyp sowie VS_x und VS_y die visuellen Sprachen der Ausgangs- und Zielsprache einer Abbildung, dann besteht ein Strukturmodell M aus den Teilmodellen $M_{VSx\_T} = \{r_{1\_VSx}, ..., r_{n\_VSx}\}$ und $M_{VSy\_T} = \{r_{1\_VSy} ..., r_{m\_VSy}\}$.

> **Abbildung einer Extension.** Die Abbildung einer Extension führt zur Erweiterung der Menge der Relationen, die im Modell M einer Struktur enthalten sind: Sei M_{VSx_T} das Teilmodell einer Struktur S_{VSx_T} in der Sprache VS_x und M_{VSy_T} das Teilmodell der selben Struktur in der Sprache

VS_y, dann führt die Abbildung der Extension a: $S_{VSx\_T} \rightarrow S_{VSy\_T}$ zu einer Erweiterung der Modelle $M_{VSx\_T} \cup \{r_{j\_VSx}, ..., r_{k\_VSx}\}$ und $M_{VSy\_T} \cup \{r_{s\_VSy}, ..., r_{t\_VSy}\}$.

Es ist möglich, dass die Abbildungen zu nicht strukturgleichen Bildern führen.
Ob eine Struktur neu entstanden ist, wird immer bei einer Erstellung eines Links untersucht, der potentiell eine Relation vervollständigt. Dabei ist es im Prinzip möglich, dass eine entstandene Relation eine Grundlage für mehrere Strukturen darstellt oder auch, dass sie gleichzeitig einen neuen Strukturkern darstellt und für eine bestehende Struktur eine Extension bildet oder Extension für unterschiedliche Strukturen ist. Es müssen in diesem Fall mehrere Abbildungen durchgeführt werden.

Mehrfachabbildungen. Wird ein Teilgraph eines Ausdruckes einer visuellen Sprache VS_x als Kern oder Extension unterschiedlicher Strukturtypen $T_{1\text{-}m}$ erkannt, so werden alle Abbildungen a_i: $S_{VSx\_Ti} \rightarrow S_{VSy\_Ti}$ durchgeführt, für alle i mit $1 \leq i \leq m$.

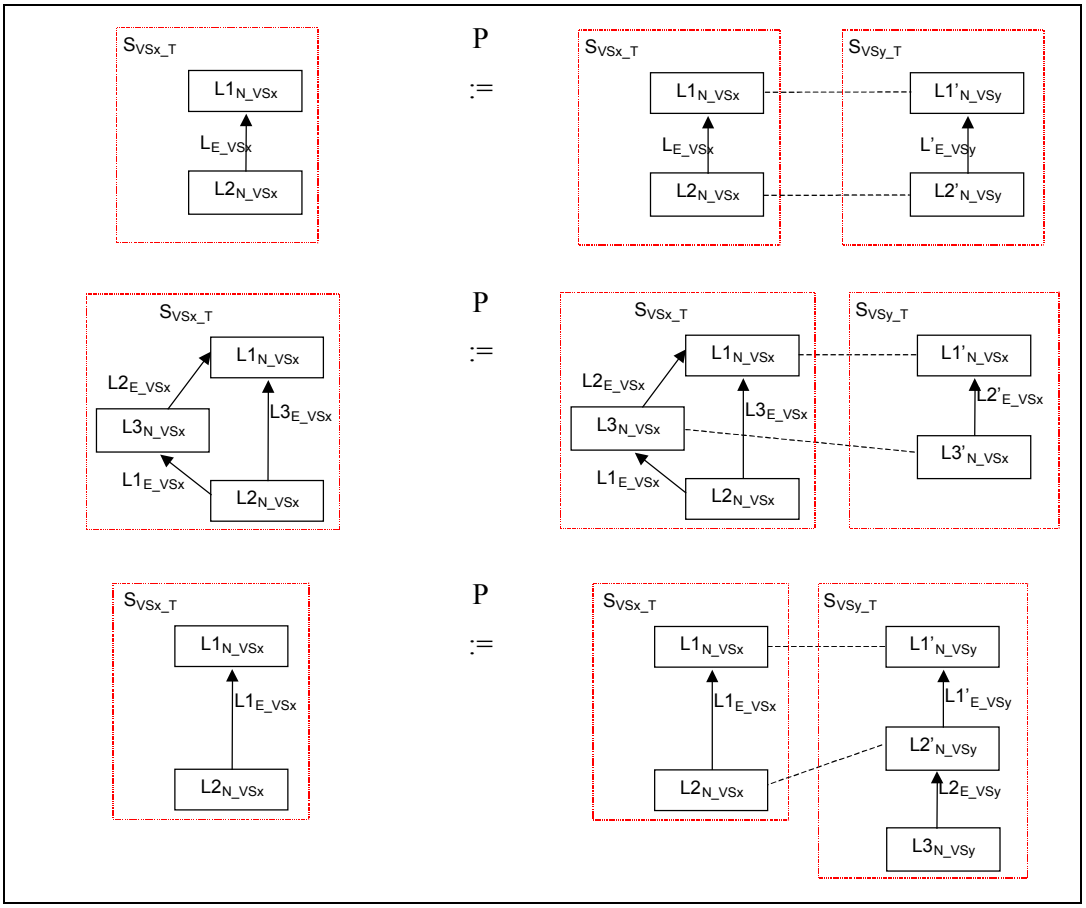


Abbildung 52

Beispiele für Produktionen, die eine Abbildung von Strukturen zwischen visuellen Sprachen darstellen.

Im Folgenden wird diese Abbildungen ebenfalls abstrahiert mit Hilfe von Graphgrammatiken beschreiben. Wie auch bei Rekers (1994) führt das zu verschiedenen Problemen. Graphgrammatiken eignen sich eigentlich nicht dazu, um multiple Perspektiven zu beschreiben, denn sie sagen nur etwas über Transformationen aus. Der Ansatz der Abbildungen führt aber zum Generieren neuer Graphen in anderen Perspektiven, die an sich nicht miteinander in Verbindung stehen. Dieses Generieren könnte zwar auch als graphgrammatische Produktionsregel notiert werden, ohne Einbettungsregeln anzugeben. Das würde aber eine Erweiterung bereits generierter Graphen ausschließen, da die Zusammengehörigkeit der Knoten oder auch Kanten verloren ginge.
Die gewählte Darstellung der Abbildungen lehnt sich an die von Rekers (1994) an. Darin wird das Generieren von Graphen auf die Transformation zurückgeführt. Dafür werden ausgezeichnete Kanten mit erstellt, die letztlich die Verbindung der Graphen verdeutlichen.
Sei $VS_{1\text{-}m}$ eine Menge von visuellen Sprachen und sei S_{VSx_T} eine Struktur zum Strukturtyp T in der Sprache VS_x aus dieser Menge $VS_{1\text{-}m}$. Dann definiert eine Produktion P die Erweiterung des Graphen um eine Struktur S_{VSy_T}. In Abbildung 52 sind Beispiele für solche Produktionen dargestellt, wobei explizit Kanten generiert werden, wenn Knoten auseinander hervorgehen (durchbrochene Kanten). Hier ist diese Verbindung als Zugehörigkeit zu einem Knotencluster zu interpretieren. Die rot und durchbrochen gezeichneten Rechtecke markieren die Teilgraphen, sind aber kein Bestandteil der Graphgrammatik.

6.6 Abbildung von Karten: Knotencluster

Um den Diskussionsprozess zu unterstützen, werden Karten automatisch von einem Workspace in einen anderen abgebildet. Als Ergebnis dieser Abbildung wird ein Knotencluster generiert, das das Modell der nun abhängigen Knoten bildet. Dafür wird in einem ersten Schritt auf Basis des Diskussionsmodells festgestellt, ob überhaupt eine Abbildung stattfinden soll. Dann wird das Diskussionsmodell nach allen zutreffenden Abbildungsvorschriften durchsucht, in denen auch definiert wird, in welche Perspektiven abgebildet werden soll. Darauf folgt die eigentliche Generierung eines Knotenclusters.
Abbildung 53 macht die Zusammengehörigkeit von Knoten zwischen den Perspektiven deutlich. Dargestellt sind die Perspektiven VS_1, VS_2 und VS_3. Unter jeder Perspektive können unterschiedliche Diagramme entstehen. Nur wenn eine Abbildung ausgeführt wird, führt das dazu, dass ein Knotencluster angelegt wird. Diese Abbildung muss nicht

von einer Perspektive in alle anderen geschehen. So sind in Knotencluster A und B beispielsweise drei Karten aus drei Perspektiven referenziert, während Knotencluster C und D nur zwei Karten referenzieren. In den Knotenclustern ist ausschließlich die Information zu Karten enthalten, keine Information zu Relationen. Die Abbildung führt nicht unmittelbar auch zum Anzeigen der Oberflächenobjekte. Erst durch ein „Refresh“ unter einer Perspektive werden auf Grundlage des Knotenclusters die erforderlichen Karten generiert.

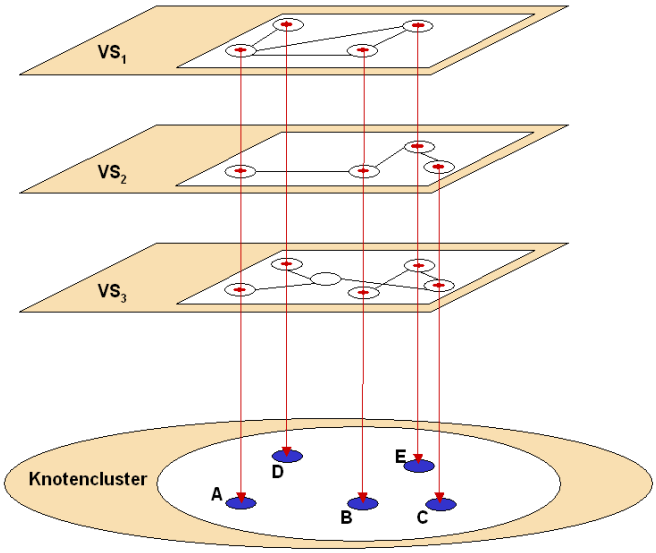


Abbildung 53

Abhängige Knoten werden intern in Knotenclustern verwaltet.

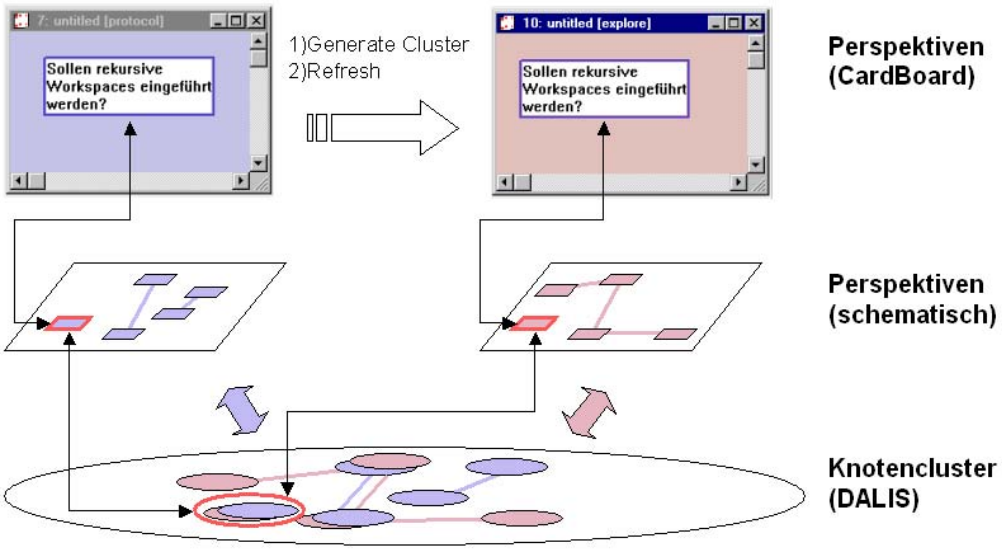


Abbildung 54

Beispiel für die Generierung neuer Karten.

Abbildung 54 zeigt dazu ein Beispiel: Symbolisiert werden dort wieder die Knotencluster in Dalis und die Perspektiven. Darüber ist jeweils ein konkreter Workspace zu erkennen wie er im DiscBoard erscheint. In dem Beispiel wird versucht, einen Ablauf zu verdeutlichen. Ein Beitrag, der links oben in der Protokollperspektive erstellt worden ist, führt zu einer Abbildung dieser Karte in die Explorationsperspektive oben auf der rechten Seite.

Aus der Protokollperspektive werden Beiträge, die als „Task“ ausgewiesen sind, automatisch in eine Karte vom Typ „Subject“ der Explorationsperspektive übernommen. Dafür wird das hervorgehobene Knotencluster erstellt und in die Datenbasis von Dalis mit aufgenommen.

In die nach dem „Refresh“ generierte Karte wird der Inhalt der Karte übernommen, die als Urbild der Abbildung diente. Dabei handelt es sich um eine zentrale Designentscheidung, diese Übereinstimmung explizit zu verwalten. Dadurch soll erreicht werden, dass Änderungen, die in einem Workspace vorgenommen werden, auch in die dazugehörigen Perspektiven propagiert werden können. In so einer Verwaltung von Inhalten und Zusammenhängen besteht ein Vorteil der Computernutzung zu nicht-digitalisierten Mitschriften, in denen dies kaum leisten in.

Die Cluster werden als Fakten in der Prolog-Datenbasis abgelegt[14]:

```
cluster(? Cluster, + Nodes, ? Content).
```

Die Cluster-Information beinhaltet die eindeutige Id des Clusters (*Cluster*), eine Liste dazugehöriger Knotenmodelle (*Nodes*) und den Inhalt, den diese Karten besitzen sollen (*Content*). Da davon ausgegangen wird, dass Knoten nur so lange in Abhängigkeit stehen, wie sie den gleichen Inhalt haben, steht der Inhalt *Content* stellvertretend für alle Karten eines Clusters. Die Cluster verwalten prinzipiell keine direkte Referenz zu Oberflächenobjekten. D.h. es wird in den Clustern weder eine Referenz zu Karten-Ids verwaltet, noch zu Workspace-Ids.

Bekannt sein muss die visuelle Sprache, der Kartentyp und das Thema. Aus diesen Informationen kann ein Oberflächenobjekt generiert werden. Auch ein Zustand *State* einer Karte wird verwaltet, der aber erst weiter hinten in diesem Kapitel erläutert wird. Entsprechend setzt sich die Knotenliste *Nodes* eines Clusters zusammen:

[14] In der Implementierung taucht das Prädikat *cluster* nicht direkt auf, sondern wird nur als Term in *db(cluster)* verwendet. Aus Gründen der Übersichtlichkeit wird im weiteren auf die Darstellung des Prädikates *db* verzichtet. Dies gilt entsprechend für alle Fakten, die in die Datenbasis eingetragen werden. Formal ergeben sich daraus formal kaum Unterschied. Für den Zugriff auf die Fakten der Datenbasis bot sich in der Implementierung jedoch eine Vereinheitlichung der Prädikate an.

cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, State$_1$) | Nodes], Content).

Dadurch, dass der „Topic“ für jede Karte im Cluster mit aufgenommen wird, können auch themenübergreifende Cluster definiert werden. Ausgenutzt wird das für das Generieren der „Topic Map“.

6.6.1 Das Generieren der Knotencluster

Wird eine Karte in einem Workspace angelegt, so wird automatisch geprüft, ob dafür ein Cluster angelegt werden muss. Das geschieht auf Grundlage des Diskussionsmodells, in dem die Abbildungsdefinitionen zwischen den Kartentypen enthalten sind. Diese Abbildungen werden mit dem Prolog-Prädikat *translate* definiert.

translate (**–** Language$_1$, **–** Type$_1$, **–** Topic$_1$, **+** Language$_2$, **+** Type$_2$, **+** Topic$_2$).

Dieses Prädikat bildet die Vorschrift, eine Karte vom Typ *Type*$_1$, mit der visuellen Sprache *Language*$_1$ und dem Thema *Topic*$_1$ in eine Sprache *Language*$_2$, mit dem Kartentyp *Type*$_2$, in das Thema *Topic*$_2$ abzubilden. Die Abbildungsvorschriften beziehen sich jeweils auf einen einzelnen Kartentyp mit Bezug auf eine visuelle Sprache. Soll ein Cluster erstellt werden, müssen deshalb alle zutreffenden Abbildungsvorschriften ermittelt werden, da für jede Abbildung in eine Perspektive eine eigene Vorschrift definiert ist. *get_translation_list* erstellt eine Liste dieser gültigen Abbildungen und erstellt daraus Informationen, die später in das Cluster übernommen werden.

```
get_translation_list(– Language1, – Type1, – Topic1, + Nodes):-
        setof((   Language2, Type2, Topic2, default),
                  translate(Language1, Type1, Topic1, Language2, Type2, Topic2),
                  + List),
        append([ (Language1, Type1, Topic1, default) ], List, + Nodes), !.
```

Mit dem Prädikat *generate_cluster* wird für die resultierende Liste (*List*) das Cluster erstellt und in der Datenbasis abgelegt. Existiert ein Oberflächenobjekt, also eine Karte, zu der Information im Cluster, so wird eine Referenz (*map*) zwischen der visualisierten Karte und dem in der Datenbasis repräsentierte Cluster erstellt. Zum Zeitpunkt der Cluster-Erstellung wird diese Referenz deshalb nur für die die Abbildung auslösende Karte angelegt, da die entsprechenden Oberflächenobjekte nicht sofort mit generiert werden.

```
generate_cluster(– Language, – Type, – Card, – Topic, – Content):-
        get_translation_list(Language, Type, Topic, + Nodes),
        unique_db_id(+ Cluster),
        …
        assert( map(Card, Language, Type, Cluster) ),
        assert( cluster(Cluster, Nodes, Content) ), !.
```

Als Gegenpol zu der Prozedur der Erstellung der Cluster muss auch das Generieren von Karten aus den Clustern implementiert werden. Um die generierten Cluster anzuzeigen steht den Nutzerinnen und Nutzern die „Refresh“-Karte aus der Funktionspalette zur Verfügung. Wird diese Karte in einen Workspace gelegt, so wird dessen Inhalt auf Grundlage der Datenbasis erweitert. Tatsächlich handelt es sich dabei um eine monotone Operation, denn durch sie kann die Anzahl der Karten in einem Workspace nur erweitert werden oder gleich bleiben. Bestehende Karten fallen durch diese Operation in keinem Fall weg und werden auch nicht verändert.
Für das Generieren der Karten werden auf Basis eines gewünschten Themas und einer Perspektive die relevanten Cluster ermittelt. Dabei handelt es sich um einen einfachen Zugriff auf eine Information der Datenbasis, da aus Effizienzgründen für jedes Thema verwaltet wird, in welchen Clustern es vorkommt.

```
refresh(_, _, – Topic, – Language):-
        …
        topic2clusters(Topic, + Clusters),
        refresh_card_cluster(_, _, Clusters, Language),
        ….
```

Das Prädikat *refresh_card_cluster* führt dazu, aus den Clustern der Datenbasis die Oberflächenobjekte zu generieren. Rekursiv wird die Liste der Cluster durchlaufen. Mit der jeweiligen Id kann das konkrete Cluster aus der Datenbasis angefragt werden. Dieses liefert dann die Daten, um die Oberflächenobjekte zu generieren.

```
refresh_card_cluster(_, _, [], _) :- !.

refresh_card_cluster(_, – Workspace, – [ Cluster | Clusters ], – Language) :-
        cluster(Cluster, + Nodes, + Content),
        create_cards(_, Workspace, Cluster, Nodes, Language, Content),
        refresh_card_cluster(_, Workspace, Clusters, Language), !.
```

Dieses Generieren der Oberflächenobjekte wird durch das Prädikat *create_cards* implementiert, in dem die folgenden Fälle unterschieden werden:

Der erste Fall verhindert durch die Abfrage *map*, die nur zutrifft, wenn das Oberflächenobjekt schon existiert, dass Karten, die bereits angezeigt sind, durch ein weiteres „Refresh“ doppelt generiert werden.

```
create_cards(_, – Workspace, – Cluster, – [ (Language, Type, _, _) | Nodes],
                – Language, – Content):-
        map(_, Language, Type, Cluster),
        create_cards(_, Workspace, Cluster, Nodes, Language, Content), !.
```

Einen Sonderfall stellt die Behandlung der „Topic“-Karten dar, da sie immer in die linke, obere Ecke positioniert werden sollen.

```
create_cards(_, – Workspace, – Cluster, – [ (Language, topic, _, State) | _],
                – Language, – Content):-
        create_card(_, type(topic), pos(0, 0), ws(Workspace), content(Content), id(Card)),
        assert( map(Card, Language, topic, Cluster)), !.
```

Das Prädikat *create_card* ist bereits Bestandteil der Schnittstelle zwischen MatchMaker und Dalis und muss für eine konkrete CardBoard-Applikation, wie hier die Diskussionsunterstützung, nicht implementiert werden. Wird eine relevante Information in der Knotenliste gefunden und *create_card* aufgerufen, so terminiert die Rekursion von *create_cards*, da pro Cluster für eine vorgegebene Sprache immer nur ein relevanter Eintrag in der Liste vorhanden ist.
Der dritte Fall von *create_cards* ist der, der zum standardmäßigen Generieren einer Karte führt. Hier wird auch der Zustand relevant, der für die Karten mit verwaltet wird, der dazu führen kann, dass neu zu generierte Karten sofort auch mit einer Zustands-markierung versehen werden. Eine solche Markierung besteht selbst aus einer Konnektorkarte und einem Link auf den Beitrag. Diese Zustandsmarkierungen werden für das Konzept der „delayed changes“ verwendet.

```
create_cards(_, – Workspace, – Cluster, – [ (Language, Type, _, State) | _],
                – Language, – Content):-
        get_position(X, Y),
        create_card(_, type(Type), pos(X, Y), ws(Workspace), content(Content), id(Card)),
        mark(_, Card, Workspace, X, Y, Type, State),
        assert( map(Card, Language, Type, Cluster) ), !.
```

Als Designentscheidung wurde festgelegt, dass die Kartenmodelle in den Clustern keine Information über deren Platzierung verwalten sollen. Dem liegt die Annahme zugrunde, dass nur in ganz wenigen Fällen davon ausgegangen werden kann, dass eine Karte an der

gleichen Position in einem Workspace generiert werden soll an der auch ihr Urbild sich befand. Dies führt dazu, dass beim Generieren von Karten auch eine Position vorgegeben werden muss. *get_position* bezieht sich also nicht auf die Position des Urbildes, sondern stellt eine Schnittstelle zu Sortierungsberechnungen dar.
Die letzten Fälle von *create_cards* bilden die Suche einer relevanten Information in der Knotenliste und die Terminationsbedingung.

```
create_cards(_, – Workspace, – Cluster, – [ (_, _, _, _) | Nodes], – Language, – Content):-
        create_cards(_, Workspace, Cluster, Nodes, Language, Content), !.

create_cards(_, _, _, [], _, _):-!.
```

6.6.2 Zustandsmarkierungen

Bisher wurden schon einige Male die Zustandsmarkierungen angesprochen. Den Hintergrund für diese Markierungen bildet das folgende Problem: Implementiert wird eine Abbildung von Karten zwischen Perspektiven. In den beschriebenen Knotenclustern werden dafür die Abhängigkeiten verwaltet. Eine Abhängigkeit bedeutet, dass wenn an einer Karte, die aus einem Knotencluster referenziert wird, der Inhalt geändert wird, wird erst einmal davon ausgegangen, dass diese Änderung potentiell auch für die anderen Karten des Clusters wichtig ist. Beispielsweise könnte eine Karte in der Explorationsperspektive, die eine Prämisse für ein Argument darstellt, in die Argumentperspektive übernommen werden. Wird an der einen Karte etwas geändert, so scheint es plausibel, diese Änderung auch in die andere Karte zu übernehmen. Jedoch sind auch Fälle denkbar, in denen das gerade nicht gewünscht ist. So kann eine übernommene Prämisse in der Argumentperspektive verfeinert werden, behält aber ihren Sinn in der Explorationsperspektive in der ursprünglichen Form. Auch ist es möglich, dass eine Karte an einer Stelle ihre Relevanz verliert und gelöscht wird, an anderer Stelle aber noch wichtige Verknüpfungen bestehen. Würden Inhalte unmittelbar propagiert, so gingen vorangegangene Inhalte verloren.
Das Problem, ob Zustände eingeführt werden sollten oder nicht, unterlag einer längeren Diskussion wie die Erwartung der Nutzerinnen und Nutzer in diesem Fall sein könnte. Diese Überlegungen hatten letztlich eine weit höhere Relevanz, als es auf den ersten Blick scheint, denn aus ihnen begründete sich letztlich die gesamte Pragmatik der Entwicklung: Es sollte ein Tool demonstriert werden, das die Diskussion unter verschiedenen Perspektiven motiviert!

Das bedeutet in Bezug auf die Karten, dass zwar durch die automatische Generierung eine Grundlage und Arbeitserleichterung für weitere Diskussionen erzielt werden sollte, sich aber die Workspace-Inhalte und damit auch die Karteninhalte in den unterschiedlichen Perspektiven „auseinander entwickeln“ können. Diese Idee wird später auch für die abgebildeten Strukturen aufrecht erhalten. Würden automatisch die Inhalte propagiert werden, wäre dieser Ansatz nicht möglich, da dadurch eine Kongruenz der Perspektiven implementiert würde, eine sehr strenge Gleichartigkeit der Inhalte auch in unterschiedlichen Kontexten!

Diese Übereinstimmung der Inhalte der unterschiedlichen Perspektiven soll aber gerade nicht angestrebt werden. Trotzdem soll die Abhängigkeit der Karteninhalte solange wie möglich zur Verfügung gestellt werden, um den Diskussionsfluss zu unterstützen und anzuregen.

Außerdem wurde davon ausgegangen, dass das sofortige Anzeigen von Änderungen die Nutzerinnen und Nutzer verwirren würde, da diese unmotiviert und als „Flimmern“ wahrgenommen würden. Warum sich was wo geändert hätte, wäre sehr untransparent.

Die Zustände sind ein Kompromiss, der einerseits erlaubt, Änderungen zu sehen aber andererseits ein lokales, örtlich beschränktes Arbeiten unter einer Perspektive ermöglicht. Änderungen, die gerade nicht relevant sind, müssen auch nicht sofort mit einbezogen werden. Mit diesen explizit dargestellten Zuständen ist es den Nutzerinnen und Nutzern möglich, zu entscheiden, ob Änderungen übernommen werden oder nicht. Solange die Änderungen übernommen werden, besteht die Abhängigkeit weiter. Wird die Änderung nicht übernommen, so wird die Abhängigkeit aufgelöst.

Daraus ergeben sich zwei Kartenzustände bzw. Markierungen wie sie in Abbildung 55 dargestellt sind:

1. Anzeigen, dass eine Karte aus dem Cluster gelöscht wurde.
2. Anzeigen, dass eine Karte aus dem Cluster modifiziert wurde.

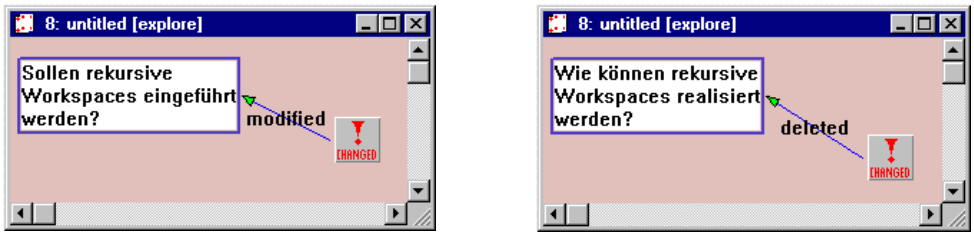


Abbildung 55

Markierung von Kartenzuständen.

Für die Darstellung der Zustände werden selbst wieder Karten verwendet. Um diese Zustände zu entfernen, werden in der Funktionspalette zwei Funktionskarten angeboten:

„Update“ und „Recover“. Mit der „Update“-Karte wird der neue Zustand übernommen, mit der „Recover“-Karte bleibt der alte Zustand bestehen und die Abhängigkeit wird aufgelöst. Die beiden Funktionskarten können jeweils in einen Workspace oder auf eine konkrete Zustandskarte gelegt werden. Dies verändert deren Aktionsrahmen, der sich entweder auf den gesamten Workspace oder nur auf eine konkrete Zustandsmarkierung bezieht.

An einem Beispiel wird in Abbildung 56 die interne Darstellung der Knotencluster den Oberflächenobjekten gegenübergestellt. Ausgangspunkt ist auf der linken Seite die Abbildung einer Karte aus der Protokollperspektive in die Explorationsperspektive. Beide Karten haben erst einmal den gleichen Inhalt (Diese Situation entspricht der, die schon in Abbildung 54 beschrieben wurde.). Darunter sind die Fakten aufgeführt, die diese Situation in der Datenbasis beschreiben. Dazu gehört zum einen das Cluster selbst, das unten dargestellt ist, und zum anderen die Referenzen der Oberflächenobjekte auf dieses Cluster.

Abbildung 56

Oberflächenobjekte und interne Darstellung zu einem Knotencluster vor und nach einer Modifizierung eines Inhaltes.

Auf der rechten Seite von Abbildung 56 wird nun das Resultat einer Änderung der Karte in der Protokollperspektive gezeigt. Die korrespondierende Karte im Explorations-Workspace behält vorerst den alten Inhalt. Allerdings wird die Zustandsänderung durch die „changed“-Markierung angezeigt. Diese Änderung in der Protokollperspektive führt bereits zur Änderung des Inhalts im Knotencluster. Der alte Inhalt ist nun nur noch über die Oberflächenrepräsentation der Karte im Explorations-Workspace zugänglich. Der entsprechende Knoten in der Knotenliste wird durch den Zustand *modification* gekennzeichnet.

Je nachdem, ob eine Löschung oder eine Modifizierung durchgeführt wurde und ob danach ein „Recover“ oder „Update“ angestoßen wird, ergeben sich diverse Fälle wie die Cluster jeweils gewartet werden müssen. Diese Fälle werden im Folgenden in der Form vorgestellt, dass der jeweilige Fall kurz beschrieben wird und danach, wie in Abbildung 56, die Fakten der Datenbasis aufgeführt werden wie sie vor und nach einer Aktivität aussehen.

Modifizieren von Inhalten

Fall 1: Zustandsmarkierungen nach Änderungen in einem Knotencluster

In diesem Fall sind alle Karten eines Clusters als Oberflächenobjekte angezeigt (Abbildung 57, linke Spalte. Darin symbolisieren die weißen Rechtecke Karten in den größeren farbig dargestellten Rechtecken, die für Workspaces stehen.). Es befinden sich Karten aus mehreren Perspektiven - hier drei - in einem Cluster. Vor der Aktion ist keine davon mit einer Zustandsmarkierung versehen. Bei einer Modifikation einer der Karten, hier von *Card*$_1$, ändern sich die Cluster-Daten.

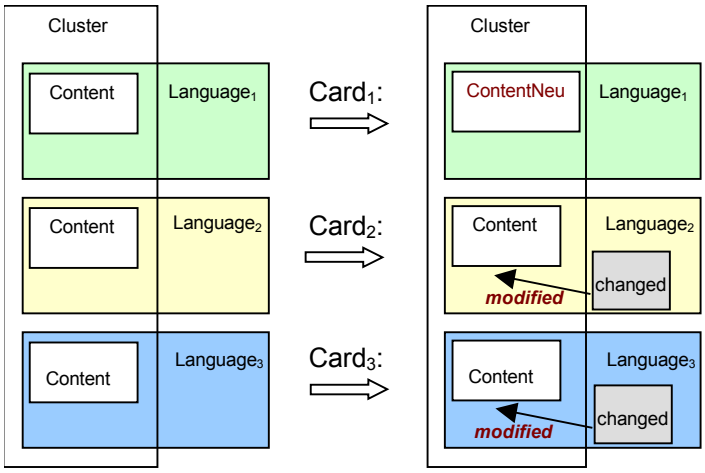


Abbildung 57

Zustandsmarkierungen nach Änderungen in einem Knotencluster.

map(Card$_1$, Language$_1$, Type$_1$, Cluster).		bleibt
map(Card$_2$, Language$_2$, Type$_2$, Cluster).		bleibt
map(Card$_3$, Language$_3$, Type$_3$, Cluster).		bleibt
cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, default), (Language$_3$, Type$_3$, Topic$_3$, default)], Content).	→	cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, modification), (Language$_3$, Type$_3$, Topic$_3$, modification)], ContentNeu).

Der neue Inhalt *ContentNeu* wird direkt in das Cluster übernommen. Die Informationen über den vorangegangenen Inhalt bestehen nur noch über die anderen unveränderten Oberflächenobjekte *Card*$_2$ und *Card*$_3$. Zu diesen werden außerdem die entsprechenden

Zustandsmarkierungen im Workspace hinzugefügt. Die folgende Gegenüberstellung zeigt Teile der Datenbasis vor und nach der Inhaltsveränderung einer Karte.

Fall 2: Weitergehende Inhaltsänderung

In Folge von Fall 1 kann die geänderte Karte, also $Card_1$, weiter geändert werden, ohne dass die angezeigten und intern repräsentierten Zustandsänderungen aufgelöst worden sind. Für die anderen angezeigten Karten, die bereits durch einen Modifizierungsmarker gekennzeichnet sind, ändert sich dabei nichts. Sie behalten weiter ihren alten Inhalt. Im Cluster ändert sich jedoch die Inhaltsinformation. In diesem Fall wurde die Entscheidung getroffen, die Zwischenstadien der Inhalte nicht mit zu verwalten. $Content_1$ steht nach einer so durchgeführten Änderung nicht mehr zur Verfügung.

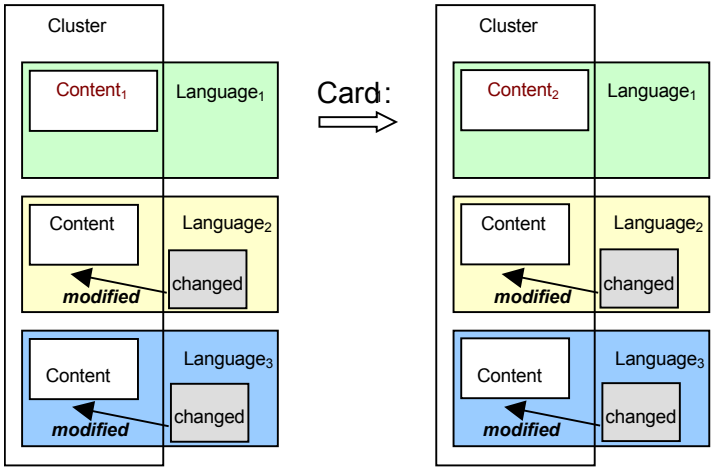


Abbildung 58

Weitergehende Inhaltsänderung.

map(Card$_1$, Language$_1$, Type$_1$, Cluster).	→	bleibt
map(Card$_2$, Language$_2$, Type$_2$, Cluster).		bleibt
map(Card$_3$, Language$_3$, Type$_3$, Cluster).		bleibt
cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, modification), (Language$_3$, Type$_3$, Topic$_3$, modification)], Content$_1$).		**cluster**(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, modification), (Language$_3$, Type$_3$, Topic$_3$, modification)], Content$_2$).

Fall 3: Veränderung einer Karte, die schon eine Zustandsmarkierung trägt

In Folge von Fall 1 kann es außerdem dazu kommen, dass eine Karte, die die Zustandsmarkierung *modification* trägt, selbst modifiziert wird, ohne vorher auf die Markierung einzugehen, d.h. sie zu akzeptieren oder abzulehnen. Dies entspricht zwar nicht der intendierten Nutzung, kann aber auch nicht ausgeschlossen werden.

Es wird davon ausgegangen, dass in diesem Fall die angebotene Modifikation keine Rolle mehr spielt. Das bedeutet, dass die neue Änderung den aktuellen Stand anzeigt, die

Zustandsmarkierung gelöscht wird und der modifizierte Knoten aus dem Cluster herausgenommen wird. Im Beispiel wird *Card*$_3$ geändert.

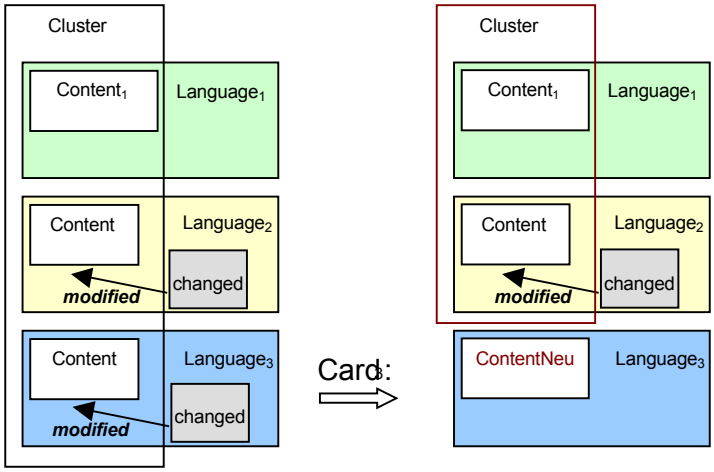


Abbildung 59

Veränderung einer Karte, die schon eine Zustandsmarkierung trägt.

map(Card$_1$, Language$_1$, Type$_1$, Cluster).		bleibt
map(Card$_2$, Language$_2$, Type$_2$, Cluster).		bleibt
map(Card$_3$, Language$_3$, Type$_3$, Cluster).	→	entfernt
cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, modification), (Language$_3$, Type$_3$, Topic$_3$, modification)], Content$_1$).	→	cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, modification)] Content$_1$).

Fall 4: Änderung einer Karte, ohne dass alle Karten des Cluster angezeigt werden

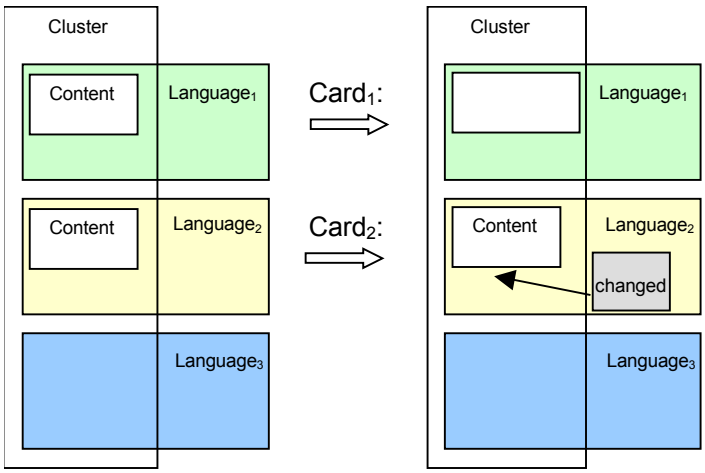


Abbildung 60

Änderung einer Karte, ohne dass alle Karten des Cluster angezeigt werden.

Es ist möglich, dass nicht alle Karten eines Clusters als Oberflächenobjekte angezeigt sind. Für die angezeigten Objekte wird verfahren wie oben beschrieben. Bei den nicht

angezeigten werden keine Zustände außer des *default* eingetragen. Dies ist einerseits eine Designentscheidung, bei der davon ausgegangen wurde, dass es verwirrt, wenn beim erstmaligen Anzeigen einer Karte gleich eine Zustandmarkierung mit erscheinen würde. Aber auch programmiertechnisch harmonierte das nicht mit dem gewählten Ansatz, in dem ja alte Inhalte nur als Inhalte der dargestellten Oberflächenobjekten vorhanden sind. Würde man so vorgehen, müsste eigentlich die gesamte Historie eines Oberflächenobjektes verwaltet werden, um sie Nutzerinnen und Nutzern zur Auswahl zu stellen. Im Beispiel ist $Card_3$ nicht angezeigt, $Card_1$ wird verändert.

map(Card$_1$, Language$_1$, Type$_1$, Cluster).		bleibt
map(Card$_2$, Language$_2$, Type$_2$, Cluster).		bleibt
cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, default), (Language$_3$, Type$_3$, Topic$_3$, default)], Content).	→	**cluster**(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, modification), (Language$_3$, Type$_3$, Topic$_3$, default)], ContentNeu).

Löschen von Karten

Fall 5: Löschen einer Karte aus einem Knotencluster

So, wie Karten, die in Clustern verwaltet werden, modifiziert werden können, können sie auch gelöscht werden. Im folgenden Beispiel sind drei Karten eines Clusters angezeigt. $Card_1$ wird gelöscht. Im Cluster wird daraufhin die Löschinformation aufgenommen. Die anderen dazugehörigen angezeigten Oberflächenobjekte werden durch den entsprechenden Zustand markiert. Der Cluster-Eintrag für $Card_1$ wird aus der Liste gelöscht und die Information *map(...)* wird aus der Datenbasis gelöscht.

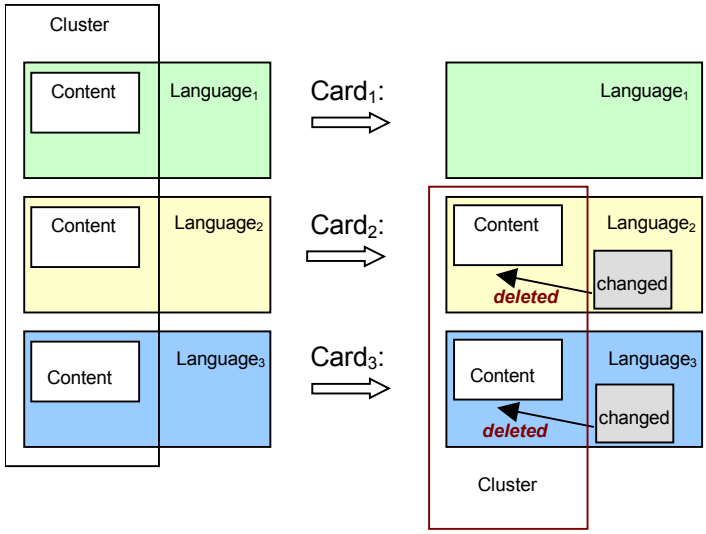


Abbildung 61

Löschen einer Karte aus einem Knotencluster.

map(Card$_1$, Language$_1$, Type$_1$, Cluster).	→	entfernt
map(Card$_2$, Language$_2$, Type$_2$, Cluster).		bleibt
map(Card$_3$, Language$_3$, Type$_3$, Cluster).		bleibt
cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, default), (Language$_3$, Type$_3$, Topic$_3$, default)], Content).	→	cluster(Cluster, [(Language$_2$, Type$_2$, Topic$_2$, deletion), (Language$_3$, Type$_3$, Topic$_3$, deletion)], Content).

Fall 6: Löschen einer Karte, wenn nicht alle Karten angezeigt sind

Ist eine Karte im Cluster enthalten aber nicht angezeigt, so wird beim Löschen einer anderen Karte aus dem Cluster intern keine Löschinformation gespeichert. In diesem Beispiel ist *Card$_3$* nicht angezeigt, *Card$_1$* wird als Oberflächenobjekt gelöscht und daraufhin die dazugehörige Cluster-Information. Da *Card$_2$* angezeigt ist, erhält es auch die „Deletion"-Information im Cluster. *Card$_3$* ist jedoch nicht angezeigt und behält die „Default"-Information.

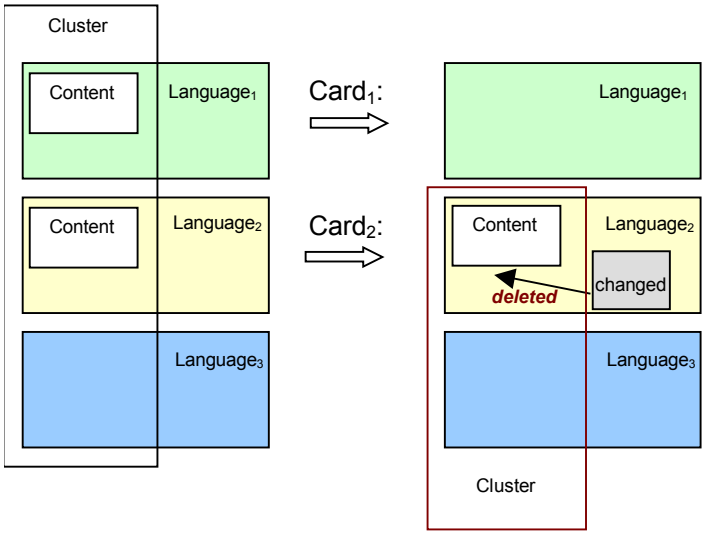


Abbildung 62

Löschen einer Karte, wenn nicht alle Karten angezeigt sind.

map(Card$_1$, Language$_1$, Type$_1$, Cluster).	→	entfernt
map(Card$_2$, Language$_2$, Type$_2$, Cluster).		bleibt
cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, default), (Language$_3$, Type$_3$, Topic$_3$, default)], Content).	→	cluster(Cluster, [(Language$_2$, Type$_2$, Topic$_2$, deletion), (Language$_3$, Type$_3$, Topic$_3$, default)], Content).

Auflösen von Zustandsmarkierungen

Um Markierungskarten wieder zu entfernen, sind zwei Funktionskarten aus der Funktionspalette vorgesehen: die „Update"-Karte und die „Recover"-Karte. Die folgenden Fälle beschreiben deren Wirkung auf die Zustandsmarkierungen. Alle

Aktionen, die durch die Funktionskarten ausgelöst werden, beziehen sich ausschließlich auf den Workspace, in den sie gelegt wurden, bzw. auf die Zustandsmarkierung, auf die die Karte gelegt wurde. Außerdem beschränken sie sich immer auf die angezeigten Karten.

Fall 7: „Update“ einer „Deleted“-Markierung

Ein „Update“ hebt eine „Deletion“-Information so auf, dass die markierte Karte zusammen mit der Markierung gelöscht wird. Nach der Aktualisierung werden die Daten auch aus dem Cluster gelöscht. In diesem Beispiel wird das „Update“ im Workspace von *Card*$_2$ ausgeführt. Vorher war eine dritte Karte des Clusters, *Card*$_1$, gelöscht worden.

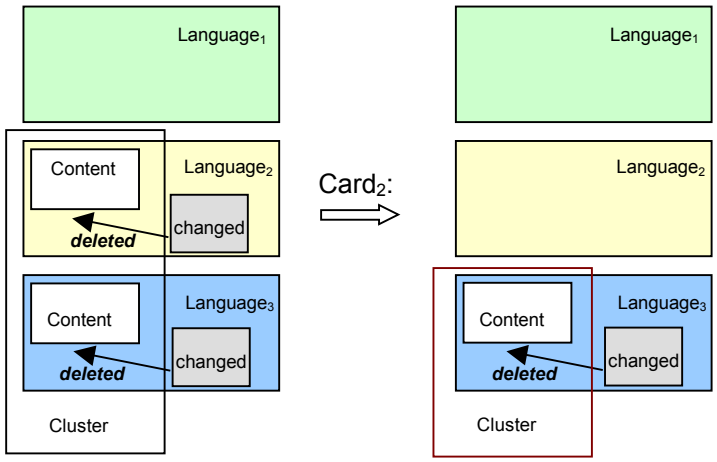


Abbildung 63

Bestätigen einer „Deleted“-Markierung.

map(Card$_2$, Language$_2$, Type$_2$, Cluster).	→	entfernt
map(Card$_3$, Language$_3$, Type$_3$, Cluster).		bleibt
cluster(Cluster, [(Language$_2$, Type$_2$, Topic$_2$, deletion), (Language$_3$, Type$_3$, Topic$_3$, deletion)], Content).	→	**cluster**(Cluster, [(Language$_3$, Type$_3$, Topic$_3$, deletion)], Content).

Fall 8: „Update“ einer Modifizierung

Ein „Update“ hebt eine „Modification“-Information so auf, dass die Markierung gelöscht wird und der Inhalt der Karte an die im Cluster gespeicherte Information angepasst wird. Im Beispiel war *Card*$_1$ modifiziert worden. Das „Update“ wird nun im Workspace von *Card*$_2$ durchgeführt.

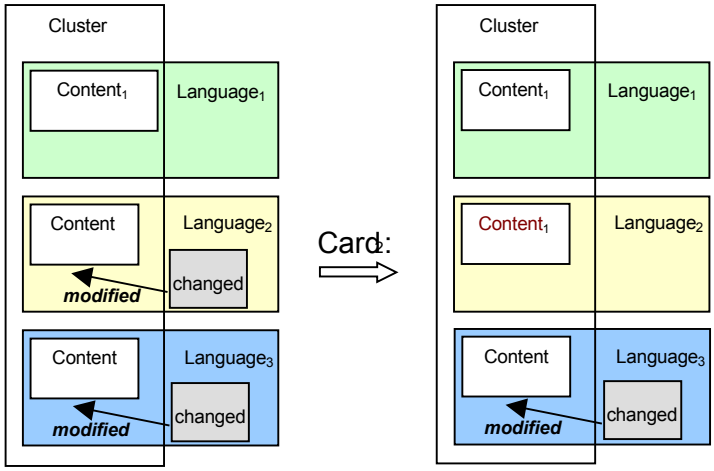


Abbildung 64

Bestätigen einer Modifizierung.

map(Card$_1$, Language$_1$, Type$_1$, Cluster).	bleibt
map(Card$_2$, Language$_2$, Type$_2$, Cluster).	bleibt
map(Card$_3$, Language$_3$, Type$_3$, Cluster).	bleibt
cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, modification), (Language$_3$, Type$_3$, Topic$_3$, modification)], Content$_1$).	→ **cluster**(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, default), (Language$_3$, Type$_3$, Topic$_3$, modification)], Content$_1$).

Fall 9: „Recover“ einer „Deleted“-Markierung

Ein „Recover“ hebt eine „Deletion“-Information so auf, dass die Markierungskarte gelöscht wird, die Karte mit dem alten Inhalt bestehen bleibt und aus dem Cluster herausfällt. In der dargestellten Situation war *Card$_1$* gelöscht worden. Für den Workspace von *Card$_2$* wird ein „Recover“ ausgeführt.

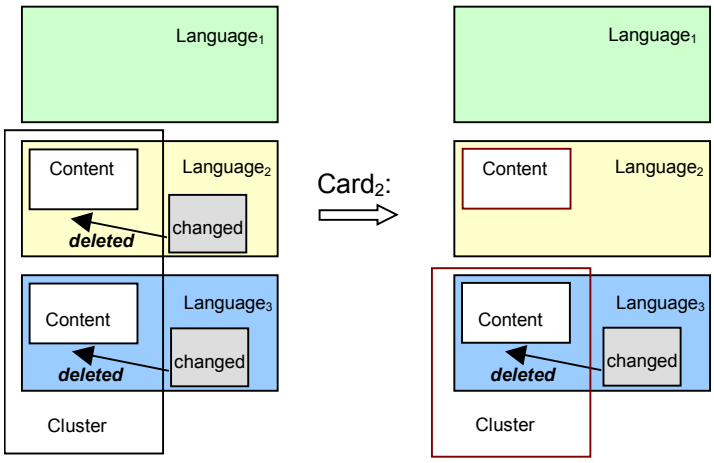


Abbildung 65

„Recover” einer Karte mit einer „Deleted“-Markierung.

Da die *map(...)*-Information nur für Cluster-Karten gespeichert wird, wird sie für $Card_2$ als Resultat ebenfalls gelöscht. Da $Card_2$ durch das „Recover“ aus dem Cluster herausgenommen wird, verändert sich auch das Cluster. Das resultierende Cluster unterscheidet sich nicht von dem, das in Fall 7 entstanden ist. Der Unterschied besteht nur darin, dass die Karte $Card_2$ bestehen bleibt, allerdings ohne weiter zu diesem Cluster zu gehören.

map(Card$_2$, Language$_2$, Type$_2$, Cluster).	→	entfernt
map(Card$_3$, Language$_3$, Type$_3$, Cluster).		bleibt
cluster(Cluster, [(Language$_2$, Type$_2$, Topic$_2$, deletion), (Language$_3$, Type$_3$, Topic$_3$, deletion)], Content).	→	**cluster**(Cluster, [(Language$_3$, Type$_3$, Topic$_3$, deletion)], Content).

Fall 10: „Recover“ einer „Modified“-Markierung

Ein „Recover“ hebt eine „Modification“-Information so auf, dass die Markierung gelöscht wird, und die Karte nicht länger zum Cluster gehört. Wie im vorangegangenen Fall wird entsprechend auch die *map(...)*-Information gelöscht, auch wenn die Karte als Oberflächenobjekt weiter bestehen bleibt. In diesem Beispiel wird das „Recover“ auf den Workspace von $Card_2$ ausgeführt. Vorher war $Card_1$ modifiziert worden.

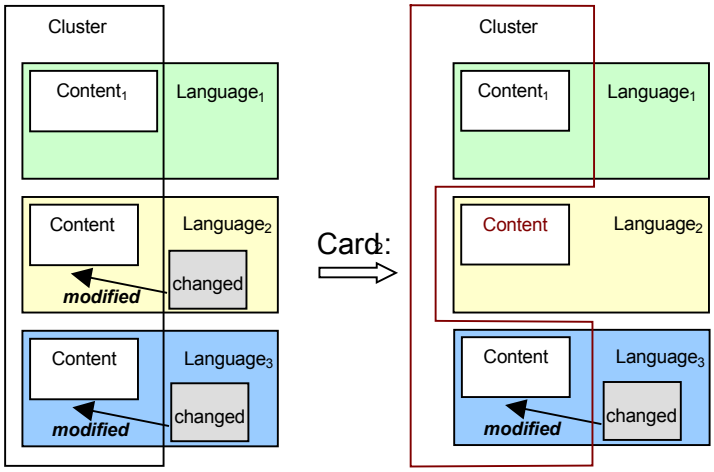


Abbildung 66

„Recover” einer Karte mit einer „Modified“-Markierung.

map(Card$_1$, Language$_1$, Type$_1$, Cluster).		bleibt
map(Card$_2$, Language$_2$, Type$_2$, Cluster).	→	entfernt
map(Card$_3$, Language$_3$, Type$_3$, Cluster).		bleibt
cluster(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_2$, Type$_2$, Topic$_2$, modification), (Language$_3$, Type$_3$, Topic$_3$, modification)], Content$_1$).	→	**cluster**(Cluster, [(Language$_1$, Type$_1$, Topic$_1$, default), (Language$_3$, Type$_3$, Topic$_3$, modification)], Content$_1$).

Fall 11:

Da die Markierungskarten für die Zustände selbst Karten sind, können diese auch direkt von den Nutzerinnen und Nutzern gelöscht werden. Entsprechend muss auch für diese Löschaktion intern eine Interpretation vorhanden sein.

- Das Löschen eines „Deleted"-Zustandes wird intern als „Recover" auf eine Karte interpretiert, die eine „Deleted"-Markierung hat. Das entspricht Fall 9.
- Das Löschen eines „Modified"-Zustandes wird ebenfalls als „Recover" interpretiert. Das entspricht Fall 10.

Schließen von Workspaces

Fall 12:

Wird ein Workspace geschlossen, in dem Zustandsmarkierungen vorlagen, so werden sie bisher nach einem erneuten Öffnen wieder mit hergestellt. Im Falle einer „Deletion"-Markierung kann das sinnvoll durchgeführt werden, da der Inhalt, der im Cluster verwaltet wird, auch den relevanten Stand des Karteninhalts reflektiert. Er entspricht dem ursprünglichen Inhalt bevor eine korrespondierende Karte aus dem Cluster gelöscht wurde.
Im Falle einer „Modification"-Markierung geht jedoch Information verloren, denn auf den ursprüngliche Inhalt vor der Änderung konnte nur über das Oberflächenobjekt selbst zugegriffen werden, das aber durch das Schließen des Workspaces nicht mehr besteht, denn in der Cluster-Beschreibung liegt schon der modifizierte Zustand vor. Eine verbesserte Strategie ist dafür noch nicht implementiert. Es kommt jedoch zu keinen echten Inkonsistenzen. Durch die Funktionskarten „Recover" und „Update" kann die generierte Markierung aufgehoben werden. In einem Fall fällt die Inhaltskarte aus dem Cluster, im anderen bleibt sie darin bestehen.
Eine Lösung dafür würde in eine allgemeinere Realisierung persistenter Daten für die Diskussionsunterstützung fallen.

6.7 Abbildung von Strukturen

Wie vorangegangen für einzelne Karten beschrieben, so ist es auch möglich, mehrere Karten, die miteinander in Relation stehen, gleichzeitig abzubilden. Diese Zusammenhänge werden Strukturen genannt. Es soll weitere Diskussionen zu bestimmten Aspekten anregen. Durch die Abbildung von Strukturen ergibt sich jedoch die zusätzliche Möglichkeit, diese räumlich zu gestalten, sie zu sortieren, und eine definierte Auswahl

von Strukturen zu treffen. Das heißt, dass Daten aus einem Kontext herausgefiltert werden, zusammengefasst werden können und potentiell bei der Abbildung verändert werden. Diese Veränderung wird als Interpretation gewertet.
In den folgenden Unterkapiteln werden die Strukturen detailliert eingeführt. Dazu gehört, wie sie sich zusammensetzen, wie sie erkannt werden und wie der Abbildungsprozess realisiert ist. Die konkreten Abbildungsvorschriften werden erst im nächsten Kapitel, das das Diskussionsmodell beischreibt, erläutert.

6.7.1 Strukturen

Ausdrücke in den hier verwendeten visuellen Sprachen sind Graphen, in denen die Inhaltskarten die Knoten darstellen. Eine Graph-Kante ist die Verbindung zwischen zwei Inhaltskarten, die aus einer Konnektorkarte und zwei Links besteht.

> Eine **Struktur** ist eine Menge von Inhaltskarten und Konnektorkarten, in der keine „unverbundenen“ Karten enthalten sind. Sie ist ein Subgraph eines Ausdruckes in einer visuellen Sprache, jedoch kein induzierter Graph, da nicht alle Kanten zwischen den durch die Struktur bestimmten Inhaltskarten auch Teil der Struktur sein müssen.

Die Implementierung der Strukturabbildungen setzt Multigraphen voraus. Das heißt, dass keine zwei gleichen Kanten zwischen zwei Inhaltskarten bestehen dürfen, wohl aber mehrere unterschiedliche Kanten.
Die Abbildung einer Struktur ist mehrstufig.

- Zuerst einmal muss erkannt werden, dass eine gültige Struktur entstanden ist.
- Für die Beitragskarten, die in dieser Struktur enthalten sind, werden Knotencluster generiert, sofern dafür nicht schon welche bestehen.
- Für die Struktur wird ein Modell angelegt, über das die Typen der Relationen verwaltet werden, die zu einer Struktur gehören und zwar bezogen auf die jeweilige Perspektive, sowie zwischen welchen Knotenclustern sie sich befinden.

Die Strukturen, die für die Diskussionsunterstützung definiert sind, bestehen nicht aus einer festen Anzahl von Knoten und Kanten. Sie wachsen oder schrumpfen während der Diskussion. Allein Knoten- und Kantentypen mit ihrer Konstellation zueinander machen eine Struktur aus. Um diesen Sachverhalt implementieren zu können, werden Strukturen über ihre kleinsten sinntragenden Teilstrukturen definiert und darüber wie diese zueinander positioniert sein müssen. Außerdem wird festgelegt, welche die minimale Ausprägung einer Struktur ist.

Die minimale Ausprägung einer Struktur wird als **Strukturkern** bezeichnet.

Extensionen sind Teilstrukturen, die ohne den Kern keine gültige Struktur darstellen.

Eine Struktur besitzt dementsprechend kein eindeutiges Muster, sondern kann als Bildungsvorschrift verstanden werden. Strukturen sind so festgelegt, dass sie aus einem Kern bestehen und möglichen Erweiterungen, den Extensionen.

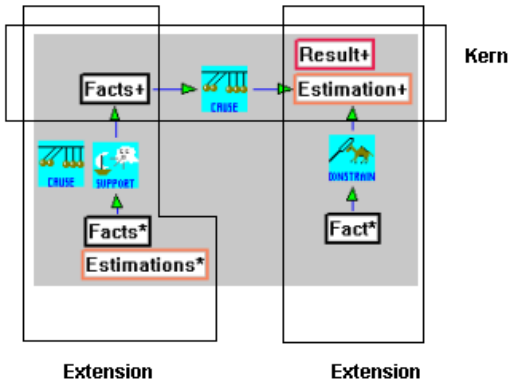


Abbildung 67

Die Teilstrukturen eines Arguments.

Beispielsweise besteht die minimale Struktur eines Arguments, wie es für die Diskussionsunterstützung modelliert ist, in dem Begründungszusammenhang einer Bedingung und dem Fazit, das daraus gezogen wird. Dies ist in Abbildung 67 zu sehen, die den Kern der Struktur darstellt, der aus einem Faktum, einer „Cause"-Relation und einem Resultat („Result") oder einer Einschätzung („Estimation") besteht. Die anderen Bestandteile, die Extensionen, sind optional und können als Teilstrukturen in beliebiger Anzahl vorkommen.

Um die Komplexität für die Strukturerkennung einzuschränken, wurde die Anzahl der Links für diese Anwendung durchgängig auf zwei Stück pro Konnektorkarte eingeschränkt. Dies ist jedoch bei der Verwendung nicht unbedingt als Nachteil anzusehen, da wie bereits geschildert, schon in der Vorstudie deutlich wurde, dass die Lesbarkeit von Hypergraphen mit mehrstelligen Relationen nicht sehr gut war.

Im Diskussionsmodell werden die Strukturen definiert. „Constraint"-artige Prolog-Regeln legen die jeweiligen Kerne und Extensionen fest. Im Vorgriff auf das Kapitel 6.8 zum Diskussionsmodell wird hier kurz das Schema dieser „Constraints" erläutert.

Die Strukturkerne für die Diskussionsunterstützung bestehen alle nur aus einer Relation. Dies ist eine Beschränkung, die von der konkreten Anwendung ausgeht, nicht aber vom implementierten Abbildungsvorgang. Eine Definition eines Kerns gibt für eine gegebene

visuelle Sprache und einen konkreten Strukturtyp, wie z.B. *argument*, vor, welche Bedingungen für die konkreten Oberflächenobjekte gelten müssen.

```
core(language, type, – View) iff condition(– View)
```

Dafür wird ein Ausschnitt des Zustands eines Workspace überprüft, ob dieser die angegebenen Bedingungen erfüllt. Diese Zustandsbeschreibung beschreibt Relationen, die wieder aus drei Informationen kombiniert sind: dem Typ einer Konnektorkarte und zwei mit ihr verbundenen Inhaltskarten. Jeweils eine Relation ist im Term *view_rel* beschrieben, der außerdem eine Richtung der Relation ausdrückt, die durch die Oberflächenobjekte vorgegeben ist: Eine Graphkante verweist von der *Condition* auf die *Conclusion*:

```
core(language, type, [ view_rel(Connector, Condition, Conclusion)]):-
        has_one( relation(types1, Connector,
                          types2, Condition,
                          types3, Conclusion)).
```

Das *has_one*-Prädikat definiert die Typen der Karten von Teilstrukturen und ebenfalls die Richtung der Relationen zwischen diesen Karten. $types_{1\text{-}3}$ steht für Typlisten. Deren Festlegung ist wesentlicher Bestandteil des Diskussionsmodells.
Mit Hilfe des *has_one*-Prädikates, im Zusammenspiel mit den Instanzierungen des Terms *view_rel*, kann überprüft werden, ob eine Relation, die gerade im DiscBoard hergestellt wurde, zur Entstehung eines neuen Strukturkernes geführt hat. Das folgende Beispiel stellt die Definition eines Argumentkernes (*argument*) in der Explorationsperspektive (*explore*) vor.

```
core(explore, argument, [view_rel(Connector, Condition, Conclusion)]):-
        has_one( relation([cause], Connector,
                          [fact], Condition,
                          [result, estimation], Conclusion)).
```

Die Erweiterungsdefinitionen der Kernstrukturen werden durch das Prädikat *extension* definiert. Eine Extension ist erst dann Teil einer Struktur, wenn der Kern dazu bereits besteht. Entsprechend wird in *extension* sowohl die Extension selbst definiert, wie es schon für den Kern durchgeführt wurde (*has_one*), als auch die Konstellation zwischen Extension und Kern vorgegeben.

```
extension(language, type, + Structure, view_rel(– Connector1, ? Condition1, ? Conclusion1)):-
        relation(– Connector1, _, ? Condition1, ? Conclusion1),
        has_one( relation(types1, – Connector1,
                          types2, – Condition1,
                          types3, – Conclusion1)),
        core(language, type, [view_rel(+ Connector2, + Condition2, – Conclusion1)]),
        \==(– Condition1, – Condition2),
        in_structure(– Conclusion1, language, type, + Structure).
```

Beim Aufruf von *extension* sind im Term *view_rel* immer nur die Konnektor-Id (*Connector*$_1$) und eines der beiden anderen Argumente instanziert. Durch das Prädikat *relation* werden die freien Variablen so belegt, dass Bedingung und Konklusion korrekt zugeordnet werden. Nach dem Typ-Check in *has_one* wird definiert, für welchen Kern die Extension gilt. Übereinstimmen muss die Perspektive (*language*) und der Strukturtyp (*type*). Im dargestellten Fall wird vorgegeben, dass die Konklusion des Kerns mit der der Extension übereinstimmen muss. In *core* wird dann ermittelt, ob zu dieser als syntaktisch korrekt erkannten Extension bereits ein Kern vorliegt. Durch die Bedingung der Ungleichheit der zwei Vorbedingungen wird ausgeschlossen, dass für die Extension und den Kern die gleiche Relation ermittelt wird.
Dass die Extension und der Kern im gleichen Workspace vorliegen, wird indirekt dadurch sichergestellt, dass die *Condition*- oder *Conclusion*-Variablen von Kern und Extension zumindest an einem Punkt übereinstimmen. Dies muss immer möglich sein, da sonst kein zusammenhängender Graph vorläge. Die letzte Anfrage (*in_structure*) ermittelt die Struktur-Id für die Extension. Auch hier ist kurz ein Beispiel für die Argumentstruktur zu nennen:

```
extension(explore, argument, Structure, view_rel(Connector1, Condition1, Conclusion1)):-
        relation(Connector1, _, Condition1, Conclusion1),
        has_one( relation([support], Connector1,
                          [fact, estimation], Condition1,
                          [fact], Conclusion1)),
        core(explore, argument, [view_rel(_, Conclusion1, _)]),
        \==(Condition1, Conclusion1),
        in_structure(Conclusion1, explore, argument, Structure).
```

In diesem Beispiel ist eine Extension definiert, die die Vorbedingung eines Argumentkernes erweitert. Die Konklusion der Extension und die Bedingung für den Kern müssen übereinstimmen.

Für diese Art und Weise der Strukturdefinitionen werden im Folgenden verschiedene Beispiele angegeben, die allerdings keinen Bezug zum konkreten Diskussionsmodell nehmen. Es werden allgemeine Möglichkeiten für Strukturbildungen vorgestellt. In den Beispielen stellen die Quadrate Inhaltskarten dar, die Kreise Konnektorkarten.

Beispiel 1 in Abbildung 68 stellt das Definitionsschema für eine Struktur dar, in der die Extension an die Bedingung eines Kerns gebunden ist. Außerdem weist die Relation vom Kern weg. Diese Struktur wird ausschließlich durch die Position der Parameter *D*, *A*, *E* bzw. *B*, *A*, *C* (für dieses Beispiel) bestimmt.

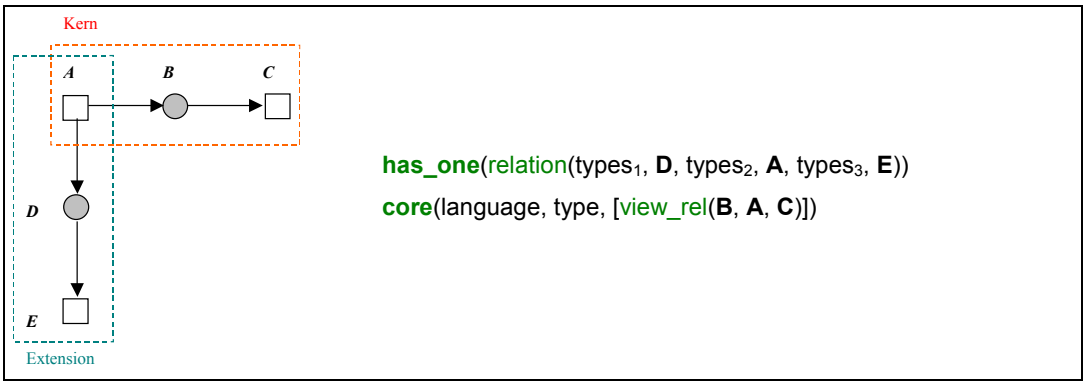


Abbildung 68

Strukturdefinition, Beispiel 1.

Beispiel 2 in Abbildung 69 stellt ein ähnlichen Fall vor, nur dass die Extension an die Konklusion des Kerns gebunden ist. Im Vergleich zwischen Abbildung 69 und Abbildung 70 kann gut erkannt werden, wie die Vertauschung der Variablen *E* und *C* die Richtung der Extensionsrelation verändert.

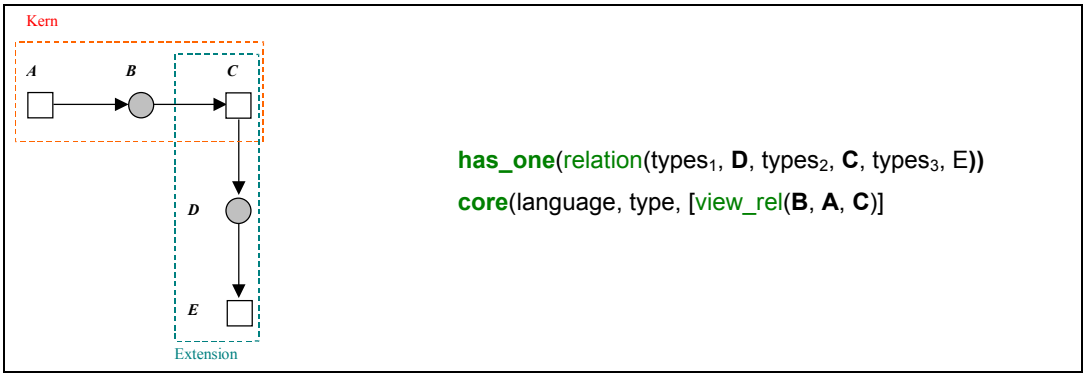


Abbildung 69

Strukturdefinition, Beispiel 2.

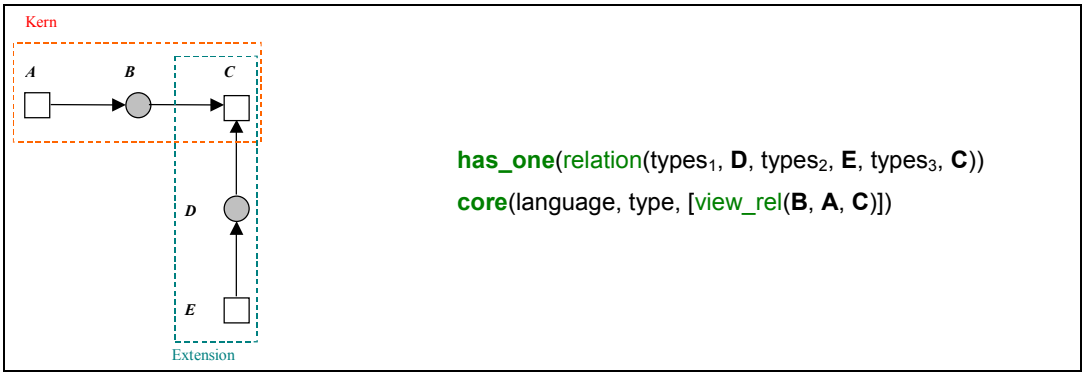


Abbildung 70

Strukturdefinition, Beispiel 3.

Abbildung 71 zeigt eine Struktur mit einem umfangreicheren Kern als Teilstruktur. Für diesen wird deshalb eine Liste von Relationen angegeben.

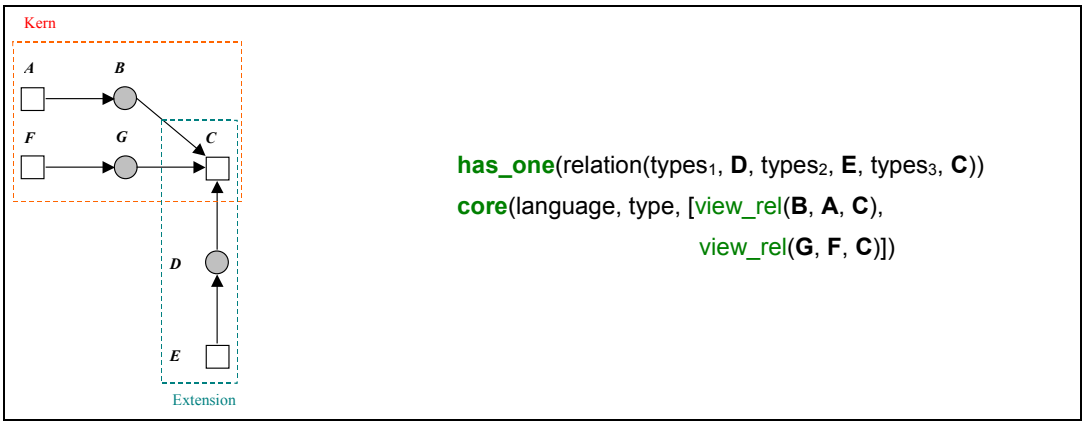


Abbildung 71

Strukturdefinition, Beispiel 4.

6.7.2 Das Strukturmodell

Wie für die Karten die Knotencluster implementiert worden sind, wird bei Abbildungen für die Strukturen ein Strukturmodell aufgebaut, das die Zugehörigkeit der Strukturen zwischen den Perspektiven verwaltet. Der Datenbasiseintrag beinhaltet die Informationen über das Thema, zu dem die Struktur generiert wurde, den Typ der generierten Struktur, den Struktur-Identifikator und das eigentliche Modell. Aus dieser Repräsentation kann die visuelle Darstellung generiert werden.

structure(**?** Topic, **?** Type, **?** Structure, **?** Model).

Das Modell besteht aus einer Liste von Tupeln, die für jede Perspektive die Graphstrukturen verwaltet.

model(**+** [(Language$_1$, Relations$_1$) | Perspectives])

Da Strukturen von einer Perspektive in die andere abgebildet werden, darf jede Sprache nur in einem der Tupel eines Strukturmodells genannt werden. Jede im Modell festgehaltene Graph-Kante, die im Term *model_rel* zusammengefasst ist, besteht wieder aus drei Teilen, dem Typ der Konnektorkarte und zwei Ids für die verbundenen Knotencluster.

model_rel(**+** Type, **+** $Cluster_1$, **+** $Cluster_2$)

In den Strukturmodellen kann nicht auf die konkreten Oberflächenobjekte verwiesen werden, da nicht gewährleistet ist, dass diese immer gerade angezeigt sind. So werden die Relationen als Verbindungen zwischen Knotenclustern definiert. Das bedeutet, dass jede Karte, die sich in einer Struktur befindet, sich außerdem in einem Knotencluster befinden muss. Auch im Term model_rel ist wieder eine Richtung der beschriebenen Kante kodiert, die wie vorher

Kodiert ist außerdem eine Richtung der Kanten. Sie verläuft immer von der ersten Cluster-Nennung in Richtung der zweiten. In detaillierter Form stellt sich die Modell-information wie folgt dar:

model([
(Language$_1$, [model_rel(Type$_{11}$, Cluster$_1$, Cluster$_2$) | Relations]),
:
(Language$_m$, [model_rel(Type$_{m1}$, Cluster$_{m1}$, Cluster$_{m2}$) | Relations])
])

6.7.3 Strukturüberprüfung

Strukturen werden zur Erstellungszeit erkannt und abgebildet, nicht erst dann, wenn eine Perspektive angefordert wird. Dadurch reduziert sich die Zeit, um einen Workspace zu einem bestimmten „Topic“ aufzubauen, wenn er benötigt wird. Wichtiger jedoch ist, dass sich der Aufwand reduziert, um die Strukturen zu identifizieren. Das liegt daran, dass eine neue Struktur oder eine Erweiterung einer bestehenden immer nur dann entstehen kann, wenn ein neuer Link als Oberflächenobjekt hergestellt wird. Dadurch kann von diesem ausgehend eine lokale Prüfung im Graphen durchgeführt werden. Müssten die Strukturen unabhängig von dieser Information, wo ein Link neu entstanden ist, erkannt werden, müsste eine globale Prüfung nicht nur eines Workspaces stattfinden, sondern aller Perspektiven, die zu dem aktuellen „Topic“ gehören, um alle relevanten Strukturen zu erkennen. Das Prädikat *structure_test* implementiert die Strukturüberprüfung.

```
structure_test(– Topic, – Language, – Connector, – Card):-
        relation(Connector, Card, + Condition, + Conclusion),
        extension_test(Topic, Language, Connector, Condition, Conclusion),
        core_test(Topic, Language, Connector, Condition, Conclusion).
```

structure_test bezieht sich auf eine konkrete Perspektive (*Language*) und einen bestimmten „Topic“. Der Ausgangspunkt der Strukturüberprüfung, ein aktuell erstellter Link, drückt sich in den Parametern *Connector* und *Card* aus.

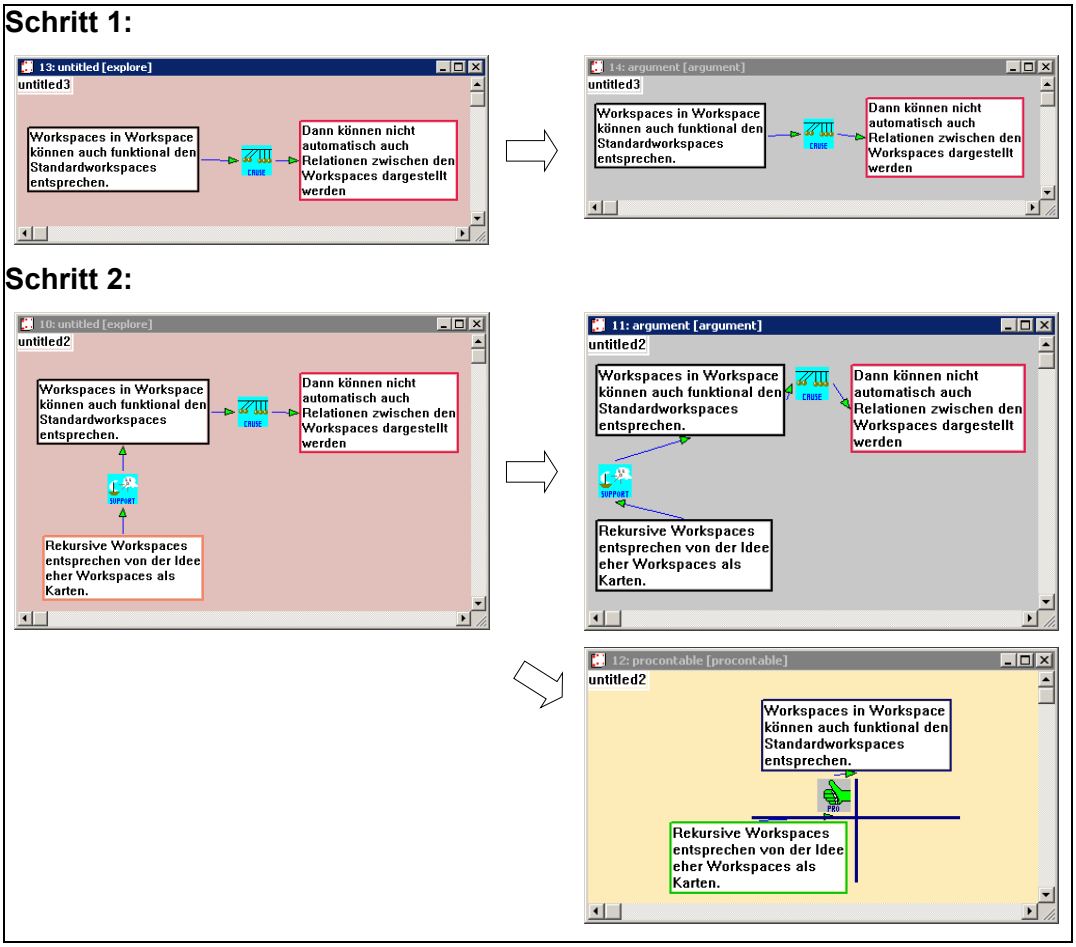


Abbildung 72

Schritt 1: Inhaltskarten sind mit der „Cause“-Relation verbunden (links). Sie werden als Argumentkern erkannt und in die Argumentperspektive abgebildet (rechts).
Schritt 2: Links: Eine weitere Inhaltskarte wird hinzugefügt und mit einer „Support“-Relation verbunden. Rechts oben: Diese Darstellung wird als Erweiterung für das bestehende Argument erkannt und in die Argumentperspektive übernommen. Rechts unten: Gleichzeitig wird ein Strukturkern für die Darstellung als Pro-/Kontratabelle erkannt.

Zu Beginn wird sicher gestellt, dass der erstellte Link auch Teil einer vollständigen Relation ist. Zusätzlich wird die Richtung, in der die Relation verläuft, herausgefunden und mit den entsprechend benannten Variablen unifiziert. Danach wird untersucht, ob es sich um eine Extension bzw. einen Strukturkern handelt. Die Reihenfolge ist dabei weitgehend unwichtig, da beides unabhängig voneinander zutreffen kann. Eine Relation

kann gleichzeitig für eine oder mehrere Strukturen eine Erweiterung sein und den Kern neuer Strukturen bilden.

Ein Beispiel dazu ist in Abbildung 72 dargestellt. Dort sind in der Explorationsperspektive (links) zwei Beiträge in einem ersten Schritt über eine „Cause"-Relation ins Verhältnis gesetzt worden. Dies wird als Argumentstruktur erkannt und in die Argumentperspektive übernommen (rechts oben). In einem zweiten Schritt wurde eine zusätzliche Inhaltskarte über die „Support"-Relation hinzugefügt (links). Dieser Zusatz wird als Extension für das bestehende Argument erkannt und dem Argument der Argumentperspektive hinzugefügt (rechts). Außerdem stellt diese Erweiterung, die in Bezug auf das Argument eine Extension bildet, auch einen neuen Kern für eine Tabelle dar. Die daraus zu entwickelnde Tabellendarstellung ist rechts unten in der Abbildung zu sehen.

```
extension_test(– Topic, – Language, – Connector, – Condition, – Conclusion):-
        extension       (Language, Type, + Structure,
                         view_rel(Connector, Condition, Conclusion)),
        extension_map   (Topic, Language, Type, – Structure,
                         [view_rel(Connector, Condition, Conclusion)]),
        fail.

extension_test(_, _, _, _, _):- !.
```

extension_test erzwingt ein Backtracking über alle im Diskussionsmodell definierten Extensionen, weil eine Erweiterung gleichzeitig mehrere Strukturen erweitern könnte. In den Argumenten spezifiziert ist der „Topic", die visuelle Sprache und die Identifikatoren der Karten, die in der visuellen Darstellung die Extension ausmachen: die Konnektorkarte *Connector* und die Inhaltskarten *Condition* und *Conclusion*. Wird eine Extension gefunden, wird dabei auch der Typ der Struktur spezifiziert (*Type*) und der konkrete Struktur-Identifikator ermittelt. Im zweiten Schritt wird die eigentliche Abbildung durchgeführt (*extension_map*), die zur Veränderung des internen Modells führt. Die Abbildungsvorschrift ist Bestandteil des Diskussionsmodells (siehe Kapitel 6.8) und wird dort weiter erläutert.

Unabhängig davon, ob in *extension_test* schon Extensionen gefunden wurden, werden in *core_test* nun alle zutreffenden Strukturkerne (*core*) gesucht.

```
core_test(– Topic, – Language, – Connector, – Condition, – Conclusion):-
        core(Language, + Type, [view_rel(Connector, Condition, Conclusion)]),
        core_extensions(Topic, Language, Type, view_rel(Connector, Condition, Conclusion)),
        fail.

core_test(_, _, _, _, _, _):- !.
```

Wird ein Kern festgestellt, so kann es außerdem sein, dass für diesen bereits Erweiterungen bestehen, die vorher noch keine eigenständigen Strukturen darstellten, die nun aber ebenfalls mit in das Modell mit aufgenommen werden müssen. Dafür ist das Prädikat *core_extensions* zuständig.

```
core_extensions (_,– Language, – Type, view_rel(– Connector, – Condition, – Conclusion)):-
        extension(Language, Type, _, view_rel(Connector, Condition, Conclusion)), !.

core_extensions (– Topic, – Language, – Type,
                        view_rel(– Connector, – Condition, – Conclusion)):-
        structure_map(– Topic, – Language, – Type, – Structure,
                        [view_rel(– Connector, – Condition, – Conclusion)]), !,
        add_existing_extensions(– Topic, – Language, – Type, – Structure), !.
```

In *core_extensions* muss außerdem das Problem gelöst werden, dass Extensionen und Kerne gleich aussehen können. Abbildung 73 zeigt links so einen Fall. Darin bildet eine „Support"- und eine „Attack"-Relation die Ausgangsinformation für eine Tabellenstruktur (rechts). Beide Relationen können sowohl einen Kern als auch eine Extension bilden. Welche Kategorie zutrifft ist rein zeitlich definiert: Die erste zulässige Relation bildet auch den Kern, alle weiteren bilden Extensionen.

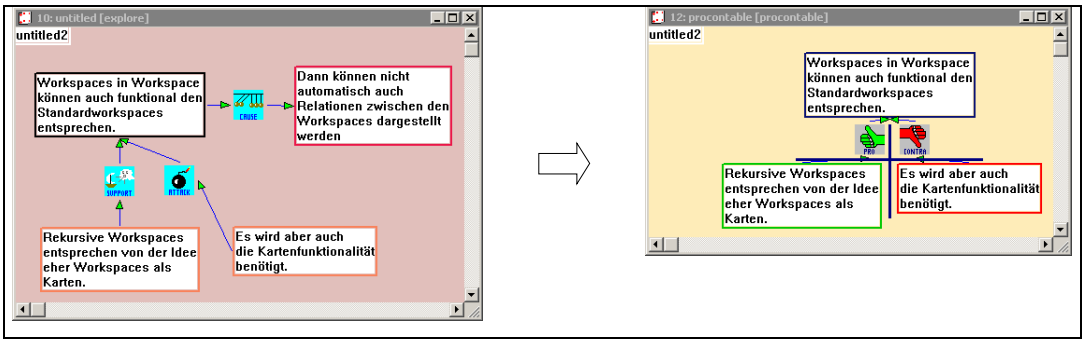


Abbildung 73

Erweiterung einer Tabellenstruktur in Anschluss an Abbildung 72.

Weil Extensionen und Kerne gleich aussehen können, muss in der ersten Regel von *core_extensions* wieder geprüft werden, ob der gefundene Kern aus *core_test* nicht schon

als Extension behandelt worden ist. Der fehlerhafte Fall, der ansonsten auftreten könnte, wäre der folgende:
Unter der Voraussetzung, dass Extension und Kern identisch aussähen, würde zuerst in *extension_test* eine Extension zu einem Kern gefunden, hier die „Attack"-Relation, die erst als Zweites dazu gekommen ist. In *core_test* würde die gleiche Relation auch als Kern erkannt werden. Ohne die erste Regel in *core_extensions* würde nun dieser erkannte Kern zu einem neuen Strukturmodell in der Datenbasis führen. Schon das wäre nicht wünschenswert. Außerdem würde aber noch in *add_existing_extensions* der bereits bestehende Kern als Erweiterung des neu entdeckten Kerns klassifiziert. So hätte man die gleiche Struktur zwei Mal erkannt und der Datenbasis hinzugefügt. In der Tabellendarstellung würde sich das in der doppelten Darstellung der gleichen Tabelle äußern.
Im Prinzip wird dadurch eine Priorität bzgl. Kern und Extension festgelegt, eine Relation, wenn möglich, als Erweiterung, nicht als neue Struktur zu identifizieren.
In der zweiten Regel von *core_extensions* findet die eigentliche Abbildung des Strukturkerns statt (*structure_map*). *add_existing_extensions* erweitert die neuen Modelle um die Modellteile bereits bestehender Erweiterungen.
Dieser Fall, dass für neue Kerne bereits Extensionen existieren, die davor aber noch nicht als solche relevant waren, kann im Prinzip auch für die Extensionen zutreffen. D.h. auch wenn eine Extension erkannt wird, könnten dafür bereits Erweiterungen bestehen, die erst durch das „Verbindungsstück" mit relevant werden. Da aber alle für die Diskussions-unterstützung definierten Strukturen nicht so aufgebaut sind, ist diese Option nicht realisiert worden.

6.7.4 Strukturabbildungen

Genauso, wie die Strukturerkennung mit veränderbaren Strukturen umgehen muss, muss dies auch die Strukturabbildung berücksichtigen. Wie die Strukturüberprüfung finden die Strukturabbildungen sukzessive statt. Das heißt, dass zuerst die Strukturkerne erkannt werden müssen, aus denen das initiale Strukturmodell generiert wird. Alle später hinzukommenden Teile sind Extensionen. Sie führen zur Ergänzung vorhandener Strukturmodelle.
Auch für die Strukturen wird das Konzept der „delayed changes" ausgenutzt. Das bedeutet, dass für die Inhaltskarten, die Bestandteil einer erkannten Struktur sind, ebenfalls Knotencluster erstellt werden. Die Abbildung der Strukturen beinhaltet damit die Abbildung der beteiligten Karten. Im Gegensatz zur Abbildung einzelner Karten, die

sofort beim Erstellen abgebildet werden, können die Karten für Strukturen aber erst abgebildet werden, wenn die Struktur entstanden ist, nicht wenn die Karte erstellt wird. Typischerweise wird für die Karten aus den Strukturen deshalb erst zum Zeitpunkt der Strukturabbildung ein Knotencluster generiert.

Das Prädikat *structure_map* implementiert die Abbildung der Strukturkerne. Diese Abbildungsregeln sind Teil des Diskussionsmodells und werden im gleichnamigen Kapitel 6.8 noch mit verschiedenen konkreten Beispielen vorgestellt. Hier soll lediglich das Prinzip einer Abbildung erläutert werden.

structure_map(– Topic, + language, + type, + Structure, – Relations)

Eine Definition einer Strukturabbildung generiert für eine festgelegte Perspektive (*language*) und einen Strukturtyp (*type*) ein neues Modell für eine Struktur. Generiert wird dafür auch ein Identifikator (*Structure*). Als Parameter der aktuellen Situation wird dafür der gerade bearbeitete „Topic“ mit übergeben und die Identifikatoren von Karten in Relationen, die hier verkürzt als *Relations* benannt sind. *structure_map* generiert das neue Modelle und fügt es direkt der Datenbasis hinzu.

Die Relationen, die auf die konkreten Oberflächenobjekte verweisen, stellen sich in der bereits bekannten Form dar, in der an erster Position die Konnektorkarte referenziert wird, danach die zwei Inhaltskarten einer Relation, einschließlich der Zuordnung einer Richtung für die Relation.

[**view_rel**(Connector, Condition, Conclusion) | Relations]

Für alle Karten, die in diesen Relationen referenziert werden, muss ein Knotencluster erstellt werden. Dafür ist das Prädikat *adapt_cluster* zuständig.

adapt_cluster(+ language, + type, – Structure, + mapDefinition, – Condition, + Cluster)

Dessen Funktion besteht vor allem darin, für eine Karte ein neues Cluster (*Cluster*) zu erstellen und dabei zu überprüfen, ob nicht bereits eines bestanden hat. Die zweite wichtige Aufgabe besteht darin, die Abbildung für die Karten zu definieren, deren Abbildung aus den Strukturen heraus angestoßen wird.

Ein allgemein gehaltenes Beispiel für *structure_map* könnte wie folgt aussehen:

```
structure_map(Topic, language1, type, Structure, [view_rel(Connector, Condition, Conclusion)]):-
        unique_db_id    (Structure),
        adapt_cluster    (language1, type, Structure,
                         [(language2 ,cardType2, Topic, default)], Condition, Cluster1),
```

```
adapt_cluster   (language1, type, Structure,
                [(language2, cardType3, Topic, default)], Conclusion, Cluster2),
assert          (structure(Topic, Type, Structure,
                model   ( [ (language1, [model_rel(connector1,Cluster1,Cluster2)]) ,
                          (language2, [model_rel(connector2, Cluster1, Cluster2) ] )
                ])), !.
```

Im Bedingungsteil ist die Sprache (*language1*) und der Typ der Struktur definiert (*type*), für den diese Abbildung auszuführen ist. *unique_db_id(Structure)* generiert einen eindeutigen Identifikator für die neue Struktur. Danach wird für die Knoten, die in der Abbildung beteiligt sind *adapt_cluster* ausgeführt. Schließlich wird das eigentliche Strukturmodell der Datenbasis hinzugefügt. Für jede Perspektive, unter der die Struktur existieren soll, werden die Typen der Relation und deren Zugehörigkeit zu den Knotenclustern aufgeführt. So kann perspektivenübergreifend das Verhältnis zwischen den Strukturen bestimmt werden.

In einer Abbildung kann es auch dazu kommen, dass zusätzliche Knoten generiert werden sollen (ein Beispiel dazu wird in Kapitel 6.8.2 vorgestellt). Dafür kann mit *new_cluster* auch direkt ein neues Cluster erstellt werden.

Die blau gekennzeichneten Konstanten sind Bestandteile, die das Diskussionsmodell ausmachen. Sie werden nicht generiert, sondern festgelegt.

Wie die Strukturmodelle die Visualisierungen beschreiben wird kurz anhand von Abbildung 74 und Abbildung 75 erläutert. Darin ist jeweils oben ein mögliches Strukturmodell angegeben. In der Grafik sind die zwei Perspektiven (*language1* und *language2*) symbolisiert. Die Rechtecke stellen Inhaltskarten dar, die Kreise Konnektorkarten. Die schwarzen, durchbrochenen Linien deuten an, dass diese zwei Kartenobjekte der unterschiedlichen Perspektiven zum gleichen Knotencluster gehören. Das Beispiel in Abbildung 74 stellt eine Umkehrung der Relationsrichtung zwischen den Perspektiven dar.

```
model( [    (language1, [model_rel(type1, cluster1, cluster2)]),
            (language2, [model_rel(type2, cluster2, cluster1)])    ])
```

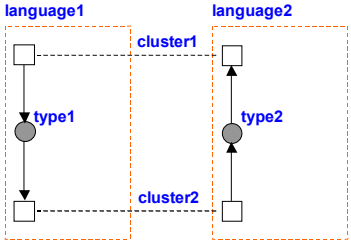


Abbildung 74

Beispiele für eine Strukturabbildung.

```
model( [    (language1, [model_rel(type1, cluster1,cluster2)]) ,
            (language2, [model_rel(type2, cluster3, cluster1),
                         model_rel(type3, cluster2, cluster3)])    ])
```

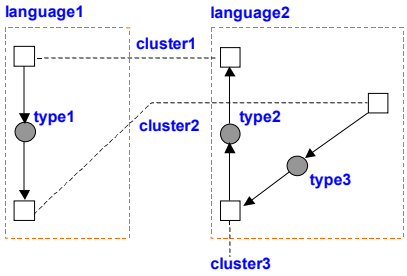


Abbildung 75

Beispiel für eine Strukturabbildung.

In Abbildung 75 wird ein Beispiel vorgestellt, in dem die Strukturen der verschiedenen Perspektiven deutlicher unterschieden sind. Es existieren drei Knotencluster, von denen eines nur eine Karte beinhaltet (*cluster3*). In einer Perspektive gehört nur eine Relation zur Struktur (*language1*), in der anderen zwei Relationen (*language2*). Die Modelldarstellung beinhaltet keine Annahmen über die Ähnlichkeit der Strukturen in den Perspektiven. Pro Perspektive werden die zugehörigen Relationen aufgeführt, die jeweils nicht auf die Karten, sondern auf die Knotencluster verweisen.

Entsprechend zu den Abbildungen der Strukturkerne müssen auch die Abbildungen der Extensionen definiert werden. Das Prädikat *extension_map* entspricht dem bereits vorgestellten Prädikat *structure_map* und es besitzt die gleichen Parameter.

```
extension_map(– Topic, + language, + type, + Structure, – Relations)
```

Auch in *extension_map* werden die Abbildungen der Strukturteile mit dem Generieren von Knotenclustern verbunden. Dazu wird ebenfalls die Prozedur *adapt_cluster* eingesetzt. Im Prädikat *extension_map* muss das bestehende Modell erweitert werden. Dafür wird die bestehende Eintragung aus der Datenbasis gelöscht und in veränderter Form wieder hinzugefügt.

6.7.5 Löschen in Strukturen

In Strukturen zu löschen bedeutet, dass Inhaltskarten oder Konnektorkarten gelöscht werden. In beiden Fällen müssen die Strukturmodelle angepasst werden. Für jede Veränderung der Struktur in den Workspaces werden die Einträge *model_rel* im Modell angepasst, beim Löschen also aus dem Modell herausgenommen.

Das Löschen von Karten verwendet wieder das Konzept der „delayed changes“. Werden in einem Fenster Karten gelöscht, wird in dem anderen die Löschung aber nicht akzeptiert, so entwickeln sich die Strukturen auseinander.
Das Löschen von Konnektorkarten führt direkt zur Änderung im Strukturmodell. Eine Löschung einer Konnektorkarte betrifft immer nur die Perspektive, aus der die Löschung durchgeführt wurde.

6.8 Das Diskussionsmodell

Das Diskussionsmodell beinhaltet die diskussionsspezifischen Abbildungen. Darunter fallen die Abbildungsvorschriften der Inhaltskarten und die der Strukturen.

6.8.1 Abbildungsdefinitionen für Inhaltskarten

Die Abbildungen der Karten zwischen den Perspektiven sind in dem Prädikat *translate* im Diskussionsmodell definiert.

```
translate (– Language1, – Type1, – Topic1, + Language2, + Type2, + Topic2).
```

Das Prädikat liest sich von links nach rechts: Die Karte vom Typ *Type1* aus der visuellen Sprache *Language1* wird abgebildet in eine Karte vom Typ *Type2* einer Sprache *Language2*. Das Thema kann dabei mit definiert werden. Eine Abbildung zwischen zwei Perspektiven stellt sich im Standardfall dar wie z.B.:

```
translate(explore, subject, Topic, protocol, task, Topic).
```

Dabei wird eine Inhaltskarte vom Typ „Subject“ aus der Sprache „Explore“ abgebildet in eine Karte vom Typ „Task“ der visuellen Sprache „Protocol“. Das Thema bleibt dabei identisch.

Abbildungen aus der Protokollperspektive

Ausgehend von der Annahme, dass die Protokollperspektive auch zur Planung einer Diskussion verwendet werden soll, werden Grundfragestellungen („Task“), Teilfragen („Question“) und einfache Beiträge („Contribution“), die im Verlauf fallen, in die Explorationsperspektive übernommen. Es wird davon ausgegangen, dass alle diese Beiträge in einer anschließenden vertiefenden Diskussion wieder aufgegriffen werden sollten.

```
%% protocol to explore
        translate(protocol, task, Topic, explore, subject, Topic).
```

```
translate(protocol, question, Topic, explore, subject, Topic).
translate(protocol, contribution, Topic, explore, estimation, Topic).
translate(protocol, topic, Topic, explore, topic, Topic).
```

Diese Vorschriften, die den Arbeitsprozess unterstützen sollen und durch den typischen Ablauf einer Diskussion motiviert sind, nehmen keinen Einfluss auf den „Topic“. Die nachfolgenden Vorschriften, die zur Verwaltung der „Topics“ notwending sind, definieren den aktuellen „Topic“ auch für die anderen Perspektiven. Die „Topic“-Karten bilden somit für jedes behandelte Thema jeweils ein Cluster. Auf diese Weise kann die Themenbenennung perspektivenübergreifend koordiniert werden.

```
%% protocol to argument
        translate(protocol, topic, Topic, argument, topic, Topic).
%% protocol to brainstorm
        translate(protocol, topic, Topic, brainstorm, topic, Topic).
%% protocol to procontable
        translate(protocol, topic, Topic, procontable, topic, Topic).
%% protocol to topics
        translate(protocol, topic, Topic, topics, topicnode, themen).
```

Für die Abbildung in die „Topic Map“ verändert sich das Thema und der Knotentyp. Die vorher in den Themen-Workspaces als „Topic“-Karte eingeblendeten Themen werden in der „Topic Map“ als „Topicnode“-Karten repräsentiert und werden dort in die Themenübersicht einbezogen. Da die „Topic Map“ ein vordefiniertes Thema (*themen*) besitzt, ist dies in der Abbildungsvorschrift angegeben.

Abbildungen aus der Explorationsperspektive

Die Explorationsperspektive kann in vieler Hinsicht als zentrale Perspektive angesehen werden. Es wird davon ausgegangen, dass die meisten Themen hauptsächlich darin aufgearbeitet werden. Sie bildet zwar die Grundlage für viele Strukturabbildungen, einzelne Karten werden bisher aber nur zurück in die Protokollperspektive propagiert, wo sie einer Zusammenfassung der Diskussion dienen. Dazu gehören Ergebnisse („Result“) und neue Aspekte, die erst während der Exploration entstanden sind („Subject“).

```
%% explore to protocol
        translate(explore, subject, Topic, protocol, task, Topic).
        translate(explore, result, Topic, protocol, contribution, Topic).
        translate(explore, topic, Topic, protocol, topic, Topic).
%% explore to argument
        translate(explore, topic, Topic, argument, topic, Topic).
%% explore to brainstorm
        translate(explore, topic, Topic, brainstorm, topic, Topic).
```

```
%% explore to procontable
        translate(explore, topic, Topic, procontable, topic, Topic).
%% explore to topics
        translate(explore, topic, Topic, topics, topicnode, themen).
```

Abbildungen aus der Brainstormingperspektive

Wiederum mit dem Ziel, entstandene Inhalte noch einmal in der Protokollperspektive aufzugreifen, werden aus der Brainstormingperspektive Gründe und Optionen aufgegriffen.

```
%% brainstorm to protocol
        translate(brainstorm, option, Topic, protocol, idea, Topic).
        translate(brainstorm, reason, Topic, protocol, contribution, Topic).
        translate(brainstorm, topic, Topic, protocol, topic, Topic).
%% brainstorm to argument
        translate(brainstorm, topic, Topic, argument, topic, Topic).
%% brainstorm to explore
        translate(brainstorm, topic, Topic, explore, topic, Topic).
%% brainstorm to procontable
        translate(brainstorm, topic, Topic, procontable, topic, Topic).
%% brainstorm to topics
        translate(brainstorm, topic, Topic, topics, topicnode, themen).
```

Abbildungen aus der Argumentperspektive

Aus der Argumentperspektive finden keine Kartenabbildungen statt, außer der Verwaltung der „Topics“. Die Argumentperspektive stellt selbst eine fokussierte Auswertung dar.

```
%% argument to protocol
        translate(argument, topic, Topic, protocol, topic, Topic).
%% argument to brainstorm
        translate(argument, topic, Topic, brainstorm, topic, Topic).
%% argument to procontable
        translate(argument, topic, Topic, procontable, topic, Topic).
%% argument to explore
        translate(argument, topic, Topic, explore, topic, Topic).
%% argument to topics
        translate(argument, topic, Topic, topics, topicnode, themen).
```

Abbildungen aus der Pro/Kontraperspektive

Wie die Argumentperspektive dient auch die Pro/Kontraperspektive der übersichtlichen Zusammenfassung. Deshalb werden auch hier keine Einzelaspekte weiter propagiert.

```
%% procontable to protocol
        translate(procontable, topic, Topic, protocol, topic, Topic).
%% procontable to brainstorm
        translate(procontable, topic, Topic, brainstorm, topic, Topic).
%% procontable to argument
        translate(procontable, topic, Topic, argument, topic, Topic).
%% procontable to explore
        translate(procontable, topic, Topic, explore, topic, Topic).
%% procontable to topics
        translate(procontable, topic, Topic, topics, topicnode, themen).
```

Abbildungen aus der Topicperspektive

Eine Möglichkeit besteht darin, zwischen Perspektiven abzubilden, jedoch das Thema dabei nicht zu beeinflussen. Dies stellt den Standardfall dar, bei dem typischerweise, ausgehend von einer Karte, nicht in alle möglichen Perspektiven abgebildet wird, sondern nur in eine kleine Auswahl, für die eine Bedeutung in einem Arbeitsprozess gesehen wird.

Die Behandlung der „Topics“ unterscheidet sich davon. Sie ordnen den Inhalten der Perspektiven Themen zu und strukturieren diese damit semantisch. Die Behandlung der „Topics“ unterliegt einer besonderen Kontrolle, da sichergestellt werden muss, dass jeweils nur eine Themenkarte in einem Workspace enthalten ist. Die „Topic“-Karten in den Perspektiven stellen eine Verbindung zu der übergeordneten „Topic Map“ dar. Die „Topic Map“ ist aber selbst wieder eine Perspektive und ist deshalb selbst einem Thema zugeordnet. Da es nur eine dieser Übersichten geben darf, ist dieses Thema festgelegt als *themen*. Die Inhaltskarten der visuellen Sprache der „Topic Map“ müssen davon unterschieden werden und werden als „Topicnode“-Karten bezeichnet. Wird ein neuer Workspace geöffnet, so wird dieser einem neuen Thema in Form einer generierten „Topic“-Karte zugeordnet, die in der „Topic Map“ als „Topicnode“-Karte aufgenommen wird.

Anders herum muss ebenfalls ein Cluster für ein Thema erstellt werden, wenn direkt in der „Topic Map“ ein Thema definiert wird. Die frei Variable *Topic*, die in den Abbildungsvorschriften definiert ist, wird bei der Erstellung des Themas instanziert.

```
%% topics to protocol
        translate(topics, topicnode, themen, protocol, topic, Topic).
%% topics to explore
        translate(topics, topicnode, themen, explore, topic, Topic).
%% topics to argument
        translate(topics, topicnode, themen, argument, topic, Topic).
```

```
%% topics to procontable
        translate(topics, topicnode, themen, procontable, topic, Topic).
%% topics to brainstorm
        translate(topics, topicnode, themen, brainstorm, topic, Topic).
```

6.8.2 Abbildungsdefinitionen für Strukturen

Neben den Abbildungsdefinitionen für einzelne Inhaltskarten werden auch die Abbildungen für die Strukturen im Diskussionsmodell definiert. Für das Diskussionsmodell werden zwei als relevant eingestufte Arten von Strukturen herausgegriffen:

- Tabellen
- Argumente

Für diese Darstellungen existieren die beiden bereits erläuterten Perspektiven, die Pro-/Kontraperspektive und die Argumentperspektive. Die Abbildungsdefinitionen für diese Strukturen beinhalten die Informationsfilter, um die dafür geeigneten Informationen aus anderen, unter Umständen mehreren, Perspektiven herauszugreifen und die Generatoren, die daraus Strukturmodelle generieren. Die folgenden Beispiele stellen die implementierten Strukturabbildungen dar.

Filtern von Argumenten aus der Explorationsperspektive

In der Argumentperspektive werden die Argumente der Explorations- und Brainstorming-perspektive zusammengefasst, die sich dort jeweils unterschiedlich darstellen. Wegen der Übereinstimmung der Sprachobjekte der Argumentationssprache mit einem Teil der Explorationssprache beschränkt sich das Filtern der Argumente aus der Explorationsperspektive auf das Erkennen derselben in größeren Strukturen, deren Selektion und deren darauffolgende Rekonstruktion in der Argumentperspektive. Abbildung 76 greift noch einmal auf, wie Argumente in der Explorationsperspektive dargestellt werden.

Abbildung 76

Schematische Darstellung von Argumenten in der Explorationsperspektive. Links: Schema eines Argumentkerns. Rechts: Kern mit Erweiterungen.

Abbildung 77 zeigt dazu ein größeres Beispiel[15]. Darin ist auf den ersten Blick nicht zu erkennen, an welcher Stelle sich Argumente herausgebildet haben. Inhaltlich werden Aspekte zum Thema Staatshaushalt aufgeführt.

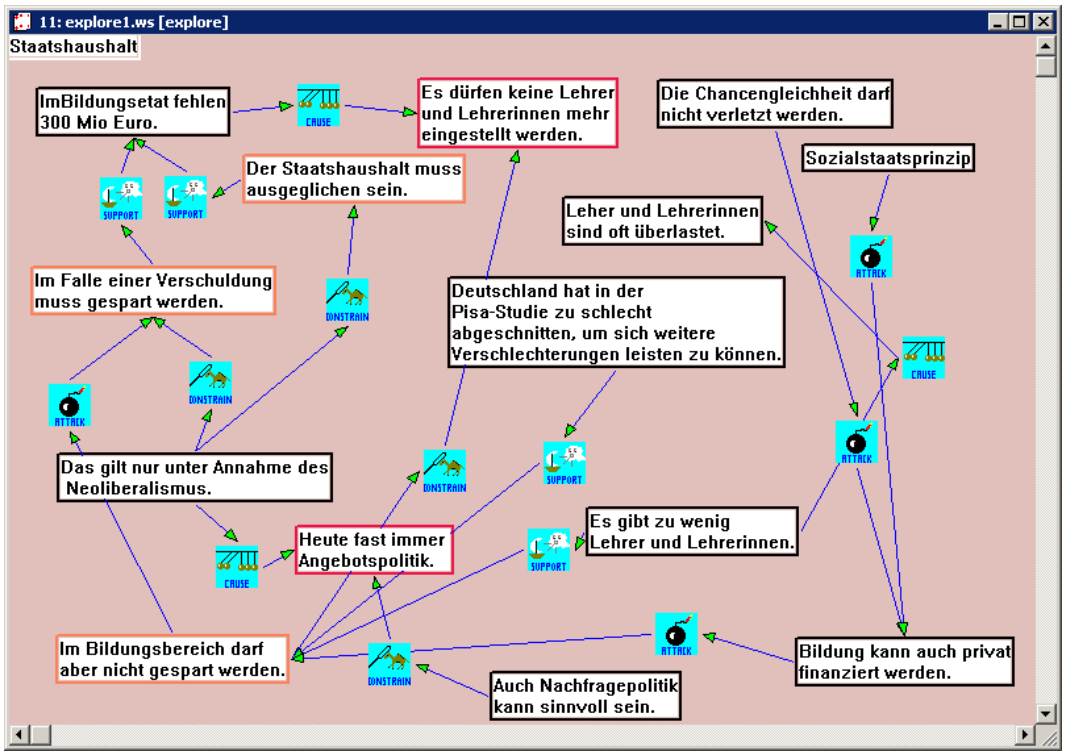


Abbildung 77

Beispiel für einen Explorations-Workspace, der verschiedene Argumente enthält, die in einen größeren Kontext gesetzt sind.

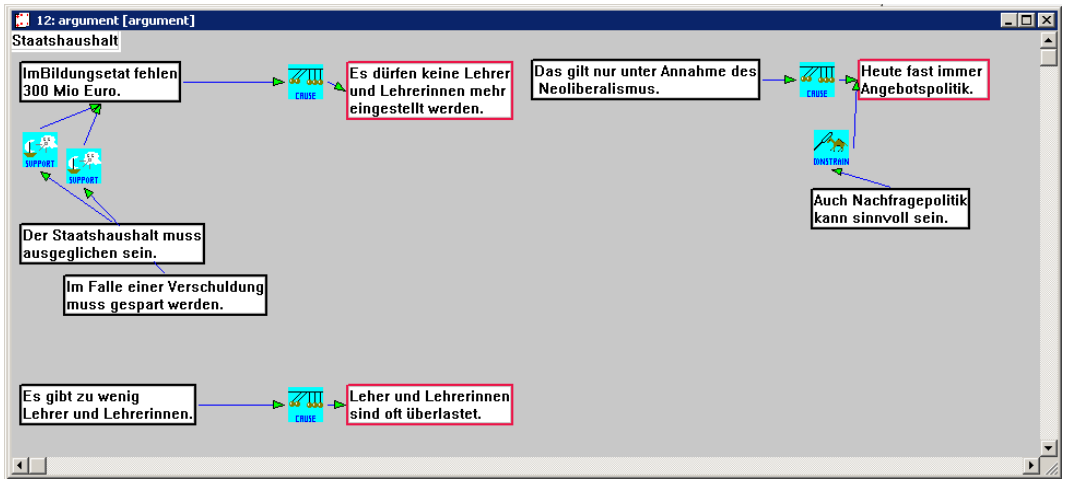


Abbildung 78

Der Argument-Workspace basiert auf den Informationen, die in Abbildung 77 dargestellt sind.

[15] Das Beispiel lehnt sich an einer Darstellung von W. Harst an [harst].

Abbildung 78 zeigt das Resultat nach dem Filtern der Argumente in einem Argument-Workspace. Die Bestandteile der Argumente haben sich wegen der gleichen Sprachkonstrukte der beiden visuellen Sprachen nicht verändert. Sie wurden jedoch aus dem Kontext herausgegriffen. Es wird deutlich, dass auch ein einfaches Filtern von Teilstrukturen zur Übersicht beitragen kann.

Im weiteren Verlauf können sich die Strukturen in den beiden Perspektiven unterschiedlich entwickeln. Alle Erweiterungen aus der Explorationsperspektive werden, soweit möglich, in die Argumentperspektive propagiert. Die umgekehrte Richtung ist nicht definiert, so dass die Darstellung in der Argumentperspektive letztlich zur verfeinerten Darstellung der Explorationsperspektive wird. Diese Tatsache ist nicht durch den technischen Ansatz determiniert. Es könnten durchaus Regeln für die Abbildung in die entgegengesetzte Richtung aufgenommen werden. Dies nicht zu tun resultiert aus der Annahme, dass die Verfeinerung unter einer anderen Perspektive gerade sinnvoll ist und nicht notwendig in die allgemeine Erörterung der Explorationsperspektive übernommen werden braucht.

Die folgenden Regeln sind für die Abbildung des Strukturkernes definiert. Da deren Prinzip bereits ausführlich erläutert wurde, werden Teile der folgenden Abbildungen in verkürzter Form aufgeführt und an entsprechenden Stellen durch ein „*“ gekennzeichnet.

```
core(explore, argument, [view_rel(Connector, Condition, Conclusion)]):-
        has_one(relation([cause], Connector, [fact], Condition, [result, estimation], Conclusion)).
```

Als Kern ist eine „Cause“-Relation deklariert, die eine „Fact“-Karte im Bedingungsteil mit einer „Result“- oder „Estimation“-Karte in der Konklusion verbindet. *explore* steht für die visuelle Sprache, für die dieser Kern definiert ist und *argument* für den Strukturtyp.

```
structure_map(Topic, explore, argument, Structure, [view_rel(_, Condition, Conclusion)]):-
        …
        unique(+ Structure),
        adapt_cluster( (argument, fact, Topic, default), Condition, Cluster1), *
        adapt_cluster( (argument, result, Topic, default), Conclusion, Cluster2),  *
        assert(     structure(Topic, argument, Structure,
                            model( [  (explore,    [model_rel( cause, Cluster1, Cluster2) ]) ,
                                      (argument, [model_rel( cause, Cluster1, Cluster2) ] ) ]
        ))),
        !.
```

Die Vorschrift für die Abbildung des Strukturkerns bewirkt, dass zwei neue Knotencluster angelegt werden, die für die Argumentperspektive auf eine „Fact"-Karte (*Cluster1*) und eine „Result"-Karte (*Cluster2*) verweisen. In das Strukturmodell wird für beide betroffenen Perspektiven jeweils eine „Cause"-Relation aufgenommen, die auf *Cluster1* und *Cluster2* referenziert.
Als Informationsgrundlage für die Argumentstruktur sind für die Explorationsperspektive außerdem die folgenden Extensionen festegelegt.

```
extension(explore, argument, Structure, view_rel(Connector, Condition1, Conclusion1)):-
        ...
        has_one(relation(       [support], Connector,
                                [fact, estimation], Condition1,
                                [fact], Conclusion1)),
        core(explore, argument, [view_rel(_, Conclusion1, _)]),
        ...
        in_structure(Conclusion1, explore, argument, Structure).

extension(explore, argument, Structure, view_rel(Connector, Condition1, Conclusion1)):-
        ...
        has_one(relation(       [constrain], Connector,
                                [fact], Condition1,
                                [result, estimation], Conclusion1)),
        core(explore, argument, [view_rel(_, Condition2, Conclusion1)]),
        ...
        in_structure(Conclusion1, explore, argument, Structure).
```

Definiert sind zwei unterschiedliche Erweiterungsmuster mit jeweils zwei möglichen Ausprägungen. Der erste „Constraint" deklariert Extensionen für den Bedingungsteil eines Arguments, der zweite für die Konklusion. Die Kartentypen der Erweiterungen werden in *has_one* vorgegeben, die Strukturdefinition ergibt sich aus dem Zusammenhang der Parameter aus dem Term *view_rel*. Der erste „Constraint" fordert, dass die Konklusion (*Conclusion1*) aus der Extension, der Bedingung des Argumentkernes entsprechen muss. Der zweite „Constraint" fordert, dass die Konklusion (*Conclusion1*) aus der Extension, der Konklusion des Argumentkernes entsprechen muss. *in_structure* stellt letztlich sicher, dass für die Extension tatsächlich schon eine Struktur existiert.
Der Extensionsüberprüfung folgt deren Abbildung und Aufnahme in das Strukturmodell.

```
extension_map(Topic, explore, argument, Structure, [view_rel(Connector, Condition, _)]) :-
        …
        retract( structure(Structure, + Model) ), *
        find_model_rel(Model, argument, model_rel( cause, _, Cluster3) ),
        adapt_cluster( (argument, fact, Topic, default), Condition, Cluster1), *
        adapt_model(Model, + NewModel,
                        [ (explore,     [model_rel( constrain, Cluster1, Cluster3)]),
                          (argument,  [model_rel( constrain, Cluster1, Cluster3)]) ]),
        assert( structure(Topic, argument, Structure, NewModel)), !.

extension_map(Topic, explore, argument, Structure, [view_rel(Connector, Condition, _)]) :-
        …
        retract( structure(Structure, + Model) ), *
        find_model_rel(Model, argument, model_rel( cause, Cluster2, _) ),
        adapt_cluster( (argument, fact, Topic, default), Condition, Cluster1),
        adapt_model(Model, + NewModel,
                        [ (explore,     [model_rel( support, Cluster1,Cluster2)]),
                          (argument,  [model_rel( support, Cluster1, Cluster2)]) ]),
        assert( structure(Topic, argument, Structure, NewModel)), !.
```

extension_map greift auf ein bestehendes Modell zu. Darin wird die Relation gesucht, mit der die Extension verbunden werden soll (*find_model_rel*). Für die neuen Karten der Extension, hier nur *Condition*, wird in *adapt_cluster* ein neues Knotencluster angelegt (*Cluster1*). Die neuen Relationen werden im Modell für die beiden betroffenen visuellen Sprachen, *explore* und *argument*, eingetragen.

Filtern von Argumenten aus der Brainstormingperspektive

Bei der Diskussionsunterstützung ist es nicht nur in der Explorationssprache, sondern auch in der Brainstormingsprache möglich, Argumente zu formulieren, die gemäß der Toulmin'schen Argumentstruktur interpretiert werden können.

Anders als bei der Explorationssprache ist die Brainstormingsprache jedoch grundverschieden von der Argumentsprache. Das heißt, dass eine wirkliche Interpretation der Karten mit ihrer Stellung innerhalb der Struktur stattfinden muss. Dafür vergegenwärtigt Abbildung 79 noch einmal die Idee der Strukturen in der Brainstormingsprache. Ausgehend von einer allgemeineren Frage („Task") werden Optionen aufgeführt („Option1" und „Option2"), die positiv und negativ gewichtet werden können. Auch Kriterien für diese Optionen können formuliert werden. Die Sterne in den Karten bedeuten, dass beliebig viele Wertungen hinzugefügt werden können.

In Abbildung 80 ist links eine Argument in der Brainstormingperspektive umrandet dargestellt. Die positiven Bewertungen können als stützende Bestandteile eines Argumentes verstanden werden. Die Kriterien dienen als Einschränkungen. Eine „Option“-Karte kann damit als Ausgangspunkt eines Argumentes verwendet werden. Für ein Argument fehlt dann jedoch noch die Konklusion, also die Frage danach, was daraus folgt, wenn eine Option verfolgt würde. Daraus könnten z.B. weitere Handlungsnotwendigkeiten erwachsen.

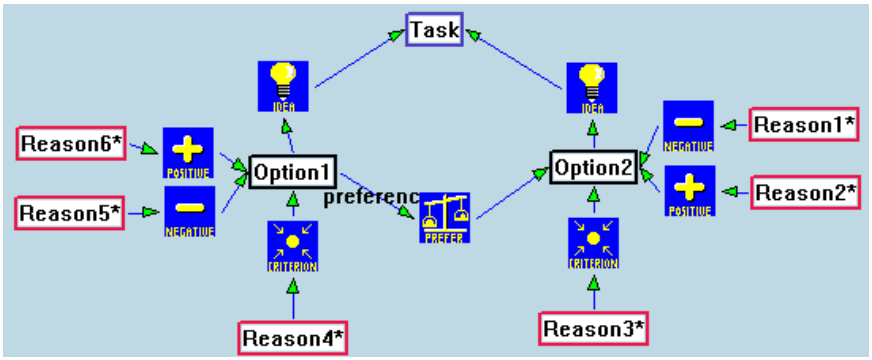


Abbildung 79

Idee der Strukturen in der Brainstormingsprache.

Abbildung 80 zeigt rechts, welche Lösung in diesem Ansatz zur Diskussionsunterstützung angestrebt wird. Und zwar wird an der Stelle der Folgerung in dem abgebildeten Argument eine leere Karte erzeugt. Diese Karte verweist auf eine potentielle Lücke bei der Argumentation. Dadurch wird hinterfragt, zu welchen Konsequenzen eine Option führen könnte.

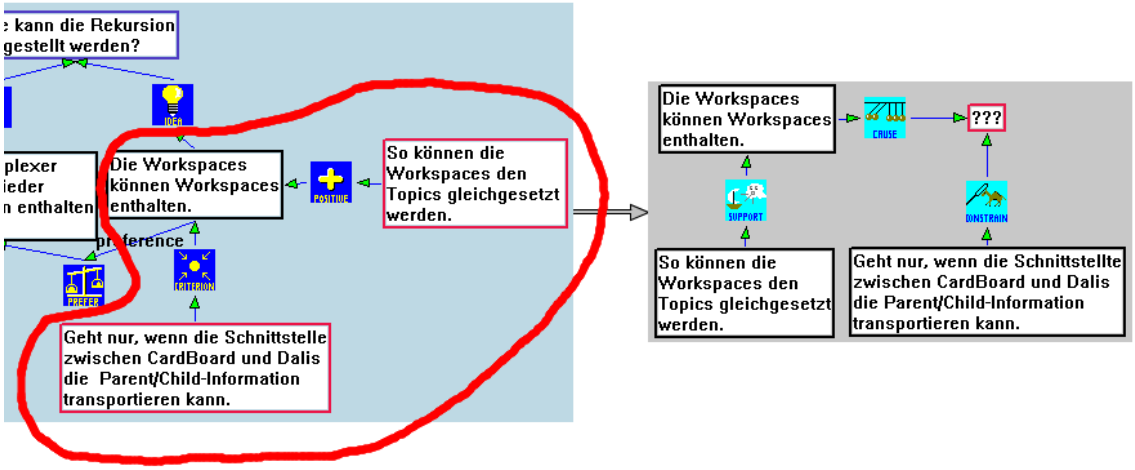


Abbildung 80

Beispiel für die Struktur eines Argumentes in der Brainstormingperspektive und die dafür implementierte Abbildung.

Negative Einschätzungen einer Option werden nicht in die Argumentdarstellung übernommen. Das liegt daran, dass die Toulmin'schen Darstellung keine Erörterung vorsieht. Zwar sind Einschränkungen der Argumentation vorgesehen aber nicht der Widerspruch. Sinnvoll ist es, die positiven und negativen Einschätzungen einer Option zusätzlich in einen Pro-/Kontra-Workspace abzubilden. Diese Vorschriften sind für diese Arbeit aber nicht mehr realisiert worden.

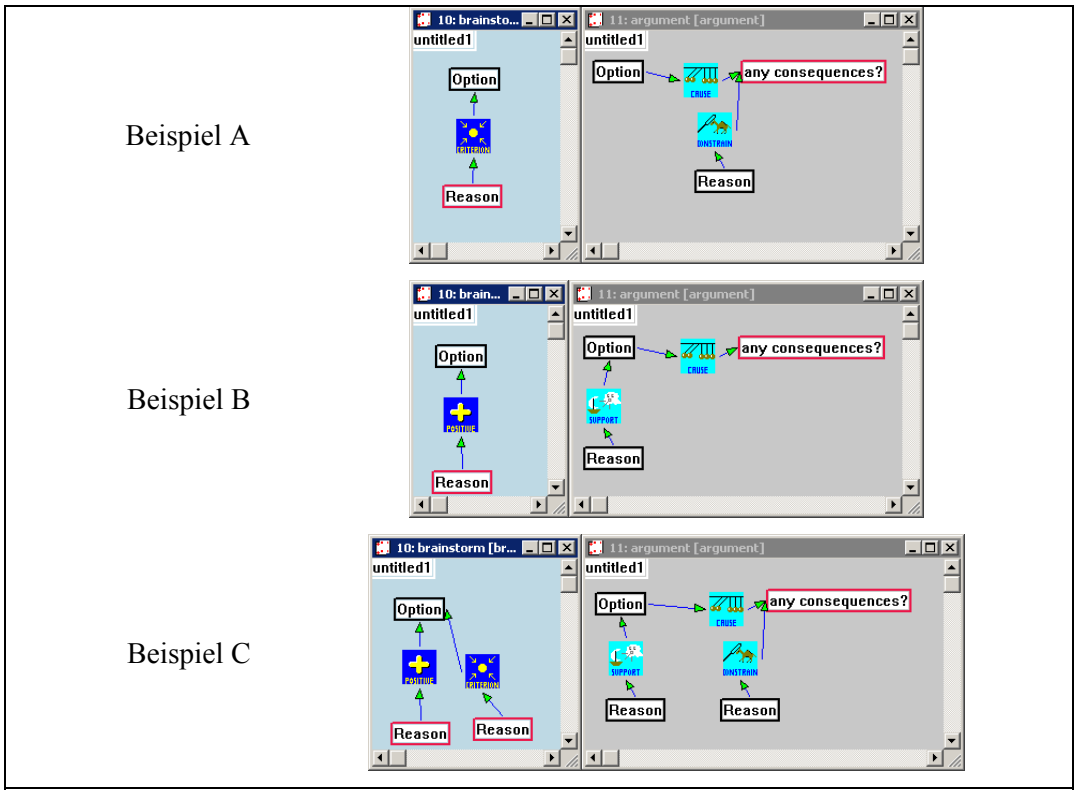


Abbildung 81

Schematische Darstellung der Abbildung von Strukturkernen von Argumenten aus der Brainstormingperspektive in die Argumentperspektive (Beispiel A und B). Erweiterungen der Argumente (Beispiel C).

Im Unterschied zu den Argumenten der Explorationsperspektive können Argumente in der Brainstormingperspektive durch zwei unterschiedliche Kerne begründet werden. Diese beiden Möglichkeiten werden in Abbildung 81, Beispiel A und B, verdeutlicht. Ein Kern besteht aus einer Konnektorkarte vom Typ „Criterion", die eine Begründung mit einer Option in Verbindung setzt. Der andere verbindet die gleichen Typen von Inhaltskarten mit der „Plus"-Relation. Da jeweils die Konklusion eines Argumentes generiert wird, reicht eine „Criterion"- oder „Plus"-Relation für das Generieren einer Argumentstruktur in der Argumentperspektive aus. Entsprechend sind für die Brainstormingperspektive zwei Kernstrukturen definiert:

```
core(brainstorm, argument, [view_rel(Connector, Condition, Conclusion)]):-
        has_one(relation([criterion], Connector, [reason], Condition, [option], Conclusion)).

core(brainstorm, argument, [view_rel(Connector, Condition, Conclusion)]):-
        has_one(relation([plus], Connector, [reason], Condition, [option], Conclusion)).
```

Die gleichen Strukturen, die als Kerne erkannt werden müssen, sind in der Erweiterung auch als Extensionen zulässig:

```
extension(brainstorm, argument, _, view_rel(Connector, Condition1, Conclusion1)):-
        ...
        has_one(relation(       [plus], Connector,
                                [reason], Condition1,
                                [option], Conclusion1)),
        core(brainstorm, argument, [view_rel(_, Condition2, Conclusion1)]),
        ...

extension(brainstorm, argument, _, view_rel(Connector, Condition1, Conclusion1)):-
        ...
        has_one(relation(       [criterion], Connector,
                                [reason], Condition1,
                                [option], Conclusion1)),
        core(brainstorm, argument, [view_rel(_, Condition2, Conclusion1)]),
        ...
```

Bei der Abbildung der Kernstrukturen wird die zusätzliche Karte für den Konklusionsteil des Arguments generiert (*new_cluster*). Für diese neue Karte wird auch ein Inhalt vorgegeben. Er besteht in der Frage „any consequences?“, die den Zweck dieser Karte erklärt und zur Reflexion anregen soll.

```
structure_map(Topic, brainstorm, argument, Structure, [view_rel(_, Condition, Conclusion)]):-
        …
        unique(+ Structure),
        adapt_cluster( (argument, fact, Topic, default), Condition, +Cluster1), *
        adapt_cluster( (argument, fact, Topic, default), Conclusion, +Cluster2), *
        new_cluster("any consequences?", (argument, result, Topic, default)], + Cluster3), *
        assert( structure(Topic, argument, Structure,
                        model( [    (brainstorm,    [model_rel( criterion, Cluster1, Cluster2) ]) ,
                                    (argument,      [model_rel( constrain, Cluster1, Cluster3),
                                                     model_rel( cause, Cluster2, Cluster3)] )] ))), !.
```

```
structure_map(Topic, brainstorm, argument, Structure, [view_rel(_, Condition, Conclusion)]):-
        …
        unique(+ Structure),
        adapt_cluster( (argument, fact, Topic, default), Condition, + Cluster1), *
        adapt_cluster( (argument, fact, Topic, default), Conclusion, + Cluster2), *
        new_cluster("any consequences?", (argument, result, Topic, default)], + Cluster3), *
        assert( structure(Topic, argument, Structure,
                        model( [   (brainstorm,   [model_rel( plus, Cluster1, Cluster2) ]) ,
                                   (argument,     [model_rel( support, Cluster1, Cluster2),
                                                   model_rel( cause, Cluster2, Cluster3)] )] ))), !.
```

Eine Schwierigkeit bei den Argumenten der Brainstormingperspektive liegt darin, dass die Extensionen und die Strukturkerne syntaktisch gleich sind. Aus diesem Grunde muss vor der Abbildung immer eindeutig ermittelt werden, ob eine Extension eben eine Erweiterung oder der Kern eines neuen Arguments darstellt. Diese Überprüfungen wurden im Kapitel 6.7.3 erläutert. Die Erweiterung einer Struktur um eine Extension hat dabei Vorrang vor dem Generieren eines neuen Strukturkernes. Die Abbildungsdefinitionen für die Extensionen lauten wie folgt:

```
extension_map(Topic, brainstorm, argument, Structure, [view_rel(Connector, Condition, _)]) :-
        …
        retract( structure(Structure, + Model) ), *
        find_model_rel(Model, argument, model_rel( cause, Cluster2, _) ),
        adapt_cluster( (brainstorm, fact, Topic, default), Condition, Cluster1), *
        adapt_model(Model, + NewModel,
                        [  (brainstorm,   [model_rel( plus, Cluster1, Cluster2)]),
                           (argument,     [model_rel( support, Cluster1, Cluster2)]) ]),
        assert( structure(Topic, argument, Structure, NewModel)), !.


extension_map(Topic, brainstorm, argument, Structure, [view_rel(Connector, Condition, _)]) :-
        …
        retract( structure(Structure, + Model) ), *
        find_model_rel(Model, argument, model_rel( cause, Cluster2, Cluster3) ),
        adapt_cluster( (brainstorm, fact, Topic, default), Condition, Cluster1), *
        adapt_model(Model, + NewModel,
                        [  (brainstorm,   [model_rel( criterion, Cluster1, Cluster2)]),
                           (argument,     [model_rel( constrain, Cluster1, Cluster3)]) ]),
        assert( structure(Topic, argument, Structure, NewModel)), !.
```

Filtern von Tabellen aus der Explorationsperspektive

Die Pro-/Kontraperspektive stellt Informationen in einer einfachen Tabellenstruktur dar. Das Ziel besteht darin, auf die Weise zu erkennen, welche Aspekte noch ungenügend besprochen wurden. Die Informationen dafür entstehen in der Explorationsperspektive. Wird in der Explorationsperspektive irgendein Beitrag angegriffen („Attack“) oder unterstützt („Support“) wird daraus sofort gefolgert, dass dort eine weiterführende Einschätzung sinnvoll wäre und eine Tabellenstruktur angelegt. Im Gegensatz zur „Attack“-Relation kann die „Support“-Relation in dem beschriebenen Diskussionsmodell Bestandteil mehrerer Strukturen sein. Die selbe Relation könnte sowohl als Teil eines Arguments erkannt werden als auch in eine Tabelle übernommen werden.

Es wäre auch sinnvoll, Informationen aus der Brainstormingperspektive in die Tabellendarstellung zu überführen. Anbieten würden sich dafür die „Plus“- und „Minus“-Relationen. Das Diskussionsmodell wurde aber bisher noch nicht um diese Regeln erweitert.

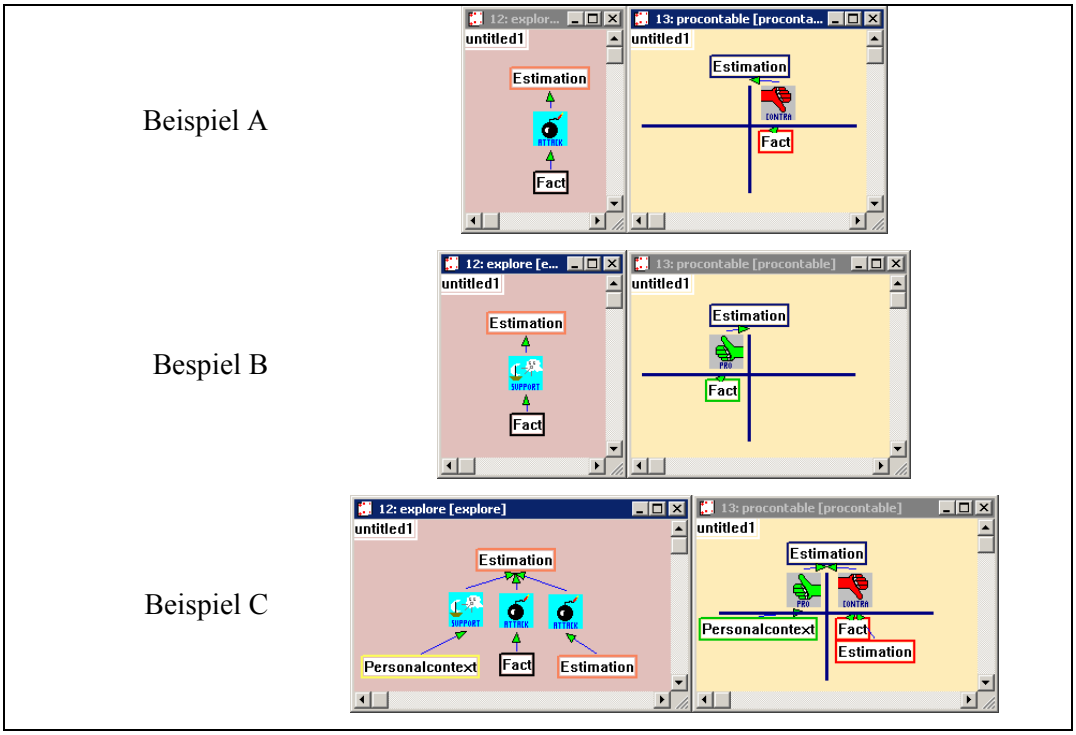


Abbildung 82

Schematische Darstellung der Abbildung der Strukturkerne aus der Explorationsperspektive in die Pro-/Kontraperspektive (Beispiel A und B). Abbildung von Erweiterungen (Beispie C).

Wie auch im vorangegangenen Beispiel gibt es zwei unterschiedliche Strukturen in der Explorationsperspektive, die zur Generierung einer initialen Tabelle führen. In Abbildung 82 sind diese Strukturkerne in Beispiel A und B dargestellt zusammen mit dem Resultat einer Abbildung in der Pro-/Kontraperspektive auf der rechten Seite. Anders als bei der

Argumentabbildung werden bei dieser Abbildung kaum Einschränkungen für die Typen der Inhaltskarten angegeben, das heißt, dass die Liste der zulässigen Typen recht lang ist.

```
core(explore, procon, [view_rel(Connector, Condition, Conclusion)]):-
        has_one(relation(    [support], Connector,
                             [fact,subject,idea,personalcontext,result,estimation], Condition,
                             [fact,subject,idea,personalcontext,result,estimation], Conclusion)).


core(explore, procon, [view_rel(Connector, Condition, Conclusion)]):-
        has_one(relation(    [attack], Connector,
                             [fact,subject,idea,personalcontext,result,estimation], Condition,
                             [fact,subject,idea,personalcontext,result,estimation], Conclusion)).
```

Wie schon für die Brainstormingperspektive erläutert, sind auch in diesem Fall die Extensionen strukturell identisch zu den Strukturkernen:

```
extension(explore, procon, _, view_rel(Connector, Condition1, Conclusion1)):-
        ...
        has_one(relation(      [attack], Connector,
                               [fact,subject,idea,personalcontext,result,estimation], Condition1,
                               [fact,subject,idea,personalcontext,result,estimation], Conclusion1)),
        core(explore, procon, [view_rel(_, Condition2, Conclusion1)]),
        ...

extension(explore, procon, _, view_rel(Connector, Condition1, Conclusion1)):-
        ...
        has_one(relation(      [support], Connector,
                               [fact,subject,idea,personalcontext,result,estimation], Condition1,
                               [fact,subject,idea,personalcontext,result,estimation], Conclusion1)),
        core(explore, procon, [view_rel(_, Condition2, Conclusion1)]),
        ...
```

Die Struktur, die in der Explorationsperspektive mit den Erweiterungen entsteht, ist sternförmig (vgl. Abbildung 82, Beispiel C). Der Mittelpunkt des Sternes bildet den Aspekt, der in der Tabellendarstellung die Überschrift bildet. Die Beiträge, die in der Explorationsperspektive über eine „Support“-Karte mit diesem Aspekt verbunden sind, werden in der Tabellendarstellung über eine „Proheader“-Karte mit der Tabellenüberschrift verbunden. Die Verbindungen über eine „Attack“-Karte werden in die „Conheader“-Relation abgebildet. Die Abbildung des Strukturkernes wird wie folgt angegeben:

```
structure_map(Topic, explore, procon, Structure, [view_rel(_, Condition, Conclusion)]):-
        …
        unique(+ Structure),
        adapt_cluster( (procontable, pro, Topic, default), Condition, + Cluster1), *
        adapt_cluster( (procontable, aspect, Topic, default), Conclusion, + Cluster2), *
        assert( structure(Topic, argument, Structure,
                        model( [  (explore,      [model_rel( support, Cluster1, Cluster2) ]),
                                  (procontable,  [model_rel( proheader, Cluster1, Cluster2)])]
        ))), !.

structure_map(Topic, explore, procon, Structure, [view_rel(_, Condition, Conclusion)]):-
        …
        unique(+ Structure),
        adapt_cluster( (procontable, con, Topic, default), Condition, + Cluster1), *
        adapt_cluster( (procontable, aspect, Topic, default), Conclusion, + Cluster2), *
        assert( structure(Topic, argument, Structure,
                        model( [  (explore,      [model_rel( attack, Cluster1, Cluster2) ]),
                                  (procontable,  [model_rel( conheader, Cluster1, Cluster2)])]
        ))), !.
```

Die neue Extension wird im Strukturmodell zu dem Cluster verbunden, in dem der Aspekt, also die Tabellenüberschrift, verwaltet wird (*Cluster3*).

```
extension_map(Topic, explore, procon, Structure, [view_rel(Connector, Condition, _)]) :-
        …
        retract( structure(Structure, + Model) ), *
        find_model_rel(Model, explore, model_rel( _, _, Cluster3) ),
        adapt_cluster( (explore, procon, Topic, default), Condition, Cluster1), *
        adapt_model(Model, + NewModel,
                        [ (explore,    [model_rel( attack, Cluster1, Cluster3)]),
                          (procon,     [model_rel( conheader, Cluster1, Cluster3)]) ]),
        assert( structure(Topic, argument, Structure, NewModel)), !.


extension_map(Topic, explore, procon, Structure, [view_rel(Connector, Condition, _)]) :-
        …
        retract( structure(Structure, + Model) ), *
        find_model_rel(Model, explore, model_rel( _, _, Cluster3) ),
        adapt_cluster( (explore, procon, Topic, default), Condition, Cluster1), *
        adapt_model(Model, + NewModel,
                        [ (explore,    [model_rel( support, Cluster1, Cluster3)]),
                          (procon,     [model_rel( proheader, Cluster1, Cluster3)]) ]),
        assert( structure(Topic, argument, Structure, NewModel)), !.
```

Die Darstellung einer Tabelle birgt ein besonderes Problem in dem implementierten Abbildungsansatz. Dieser geht immer von der expliziten Darstellung der Relationen aus, die im Modell verwaltet werden. Die Visualisierung der Relationen in einer Tabelle ist jedoch erst einmal nicht plausibel, da schon das Tabellenformat der Übersichtlichkeit dient und die expliziten Relationen ersetzt. Die Zugehörigkeit eines Beitrages zu einer Tabelle ist durch seine Positionierung optisch eindeutig bestimmt. Es wäre im Prinzip möglich, diese Nähe intern als Relation zu modellieren.

Was optisch leicht zu sehen ist, ist allerdings formal unter Umständen nur schwierig definierbar, insbesondere wenn die Tabellen sehr eng stehen und insbesondere dann, wenn gar nicht davon ausgegangen werden kann, dass Karten überhaupt immer korrekt und sinnvoll platziert sind. Karten können ja auch vorübergehend hierhin und dorthin geschoben werden, beispielsweise um Platz zu schaffen.

Für die Tabellendarstellung muss deshalb entschieden werden, ob die Beibehaltung der expliziten Relationen bei einer Abbildung von einem Workspace in einen anderen für diesen Fall aufgegeben werden soll. Dem widerspricht, dass die Tabellen nicht nur einmalig aus der Datenbasis generiert werden, sondern auch ergänzbar sein sollen. Wenn also eine Struktur in der Explorationsperspektive ergänzt wird, soll diese Ergänzung auch mit in die richtige Tabelle aufgenommen werden. Dazu muss aber bekannt sein zu welcher Tabelle und wo sich diese befindet. Diese Information ist ohne explizite Relationen aus der visuellen Darstellung kaum deduzierbar, denn es können kaum sinnvolle Annahmen darüber gemacht werden, wie „gut“ oder „schlecht“ die Karten durch Nutzerinnen und Nutzer platziert werden.

Die Lösung besteht in einem Trick. Die Zugehörigkeit der Beiträge zu den Tabellen wird auch in dieser Perspektive durch explizite Relationen ausgedrückt. Damit ist die Zugehörigkeit der Karten zu den Aspekten eindeutig. Auch die Positionierung ist dadurch frei gestaltbar, da die Zugehörigkeit über die Relationen wieder zugreifbar ist. Der Trick besteht darin, das alleine durch die Anordnung der Karten die Tabellen so aufgebaut werden, dass die Relationen kaum sichtbar sind und deshalb auch kaum stören. Dazu gehört, dass die Konnektorkarten, die die Überschriften der Tabellenspalten bilden, alle übereinandergelegt werden, so dass nur eine sichtbar ist. Die Beiträge werden eng aneinandergelegt, so dass die Linklinien verdeckt werden (Abbildung 84). In Abbildung 83 ist ein Beispiel für eine Explorationsperspektive dargestellt. Es dient als Grundlage für die generierten Tabellen in Abbildung 84.

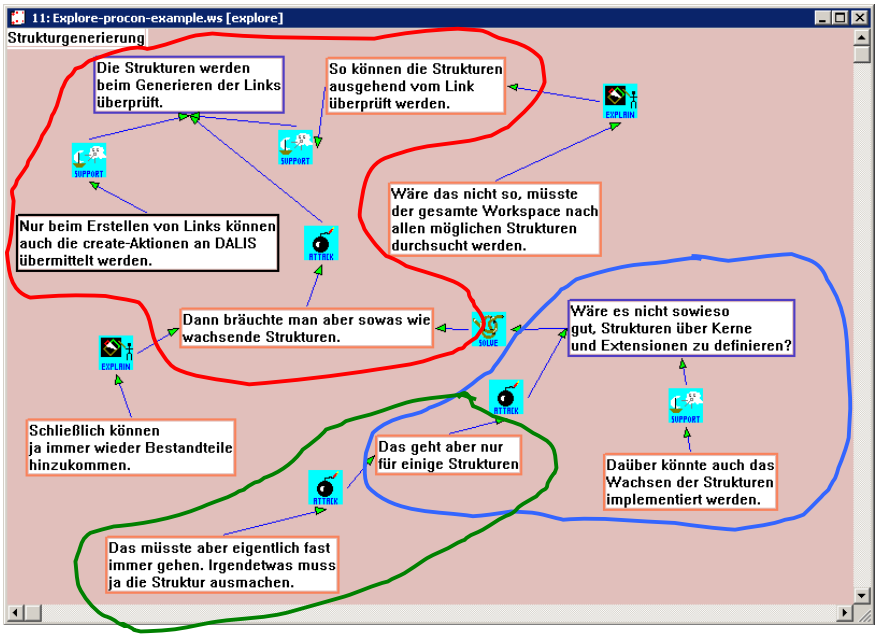


Abbildung 83

Der Explorations-Workspace stellt die Grundlage für die Tabellendarstellung in Abbildung 84 dar.

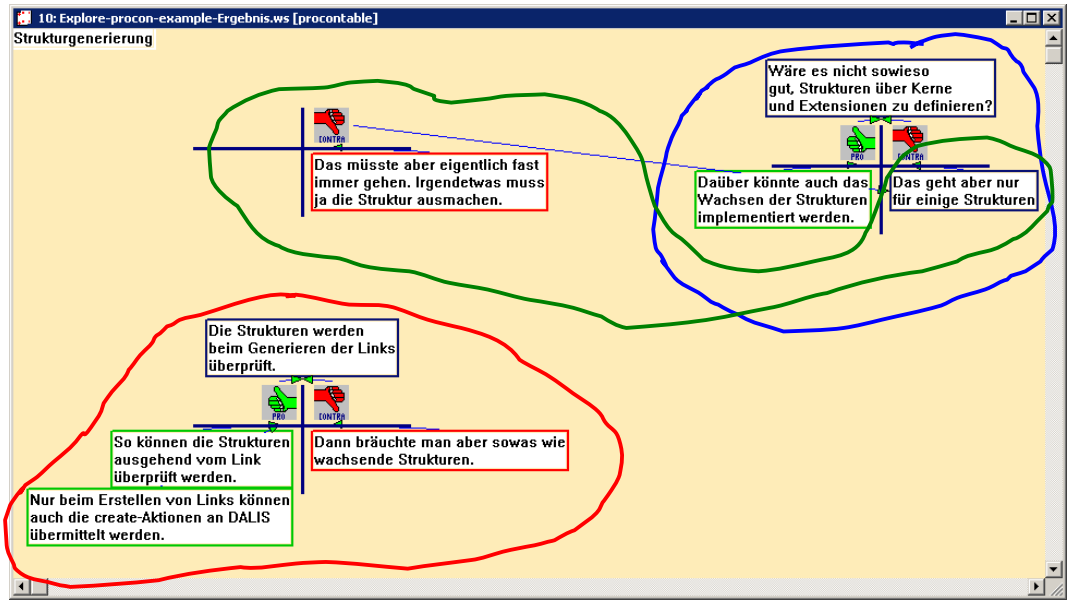


Abbildung 84

Beispiel einer Pro-/Kontradarstellung, die aus dem Explorations-Workspace in Abbildung 83 generiert wurde.

Die Tabellen, die aus den Strukturen in der Explorationsperspektive hervorgegangen sind, sind in beiden Abbildungen mit den gleichen farblichen Umrandungen markiert.

Aus der Darstellung in Abbildung 84 wird noch ein weiteres Problem bei der Generierung der Tabellen deutlich. In der Graph-Repräsentation eines Explorations-Workspaces

können die Karten mehrfach mit anderen Karten verbunden werden. In der Tabellendarstellung würde das dazu führen, dass eine Karte mehrfach dargestellt werden müsste, um z.B. ein Argument für mehrere Aspekte zu bilden. Diese Mehrfachdarstellung von Karten in einem Workspace ist aber nicht zugelassen, da sie dem implementierten Prinzip der Knotencluster widerspricht.
Auch für die Tabellen musste sichergestellt werden, dass jede Karte nur genau ein Mal im Arbeitsbereich vorliegt. Das wird durch die Sortierung so unterstützt, dass eine Karte wenn sie mehrfach referenziert wird, immer wieder an den „neuesten" Platz der Sortierung gelegt wird: sie wandert. An dieser Stelle bewährt sich die vorangegangene Entscheidung, die Relationen doch anzuzeigen. Diese können hier ausgenutzt werden, um Zugehörigkeiten von Karten tabellenübergreifend darzustellen. So ein Fall ist in Abbildung 84 für die Tabellenüberschrift zu erkennen (links oben), die gleichzeitig als Kontraargument in einer anderen Tabelle (rechts oben) aufgeführt ist.

6.9 Sortierungen

Die bisher beschriebenen Informationsfilter und Abbildungsvorschriften machen keine Angaben zum Layout der Darstellung. Für manche Darstellungen ist die Übersichtlichkeit aber notwendig, um überhaupt einen Vorteil aus der Visualisierung ziehen zu können. Dies betrifft alle jene Workspace-Typen, die von einer strukturierten Darstellung ausgehen, bzw. deren Strukturen einem bestimmten Muster folgen. Das betrifft die Argumentperspektive und die Pro-/Kontraperspektive.
Für die dort verwendeten Muster können keine allgemeinen Platzierungsalgorithmen verwendet werden, wie sie z.B. für die Darstellung von Bäumen, gleichmäßige Verteilungen oder möglichst wenige Überschneidungen von Kanten, angewendet werden. Eine Platzierung der Karten muss sich an der Strukturidee der Perspektive orientieren und zusätzlich deren graphischen Mittel berücksichtigen. Beispielsweise werden die Tabellen auch über Inhaltskarten formatiert.
Für die einzelnen Perspektiven werden Layouts implementiert, in denen die Anordnung einzelner Strukturen definiert ist. Im Falle der Argumentperspektive sind das die einzelnen Argumente, bei der Pro-/Kontraperspektive die einzelnen Tabellen. Zusätzlich wurde ein Layout implementiert, das die Verteilung der einzelnen Strukturen im Workspace realisiert. Es gibt also zwei Arten von Layouts:

1. Layouts für die jeweilige Perspektive, die die relative Platzierung der Karten der einzelnen Strukturen berechen.
2. Layouts, die die absolute Platzierung der Strukturen im Workspace berechnen.

Um die Perspektivenlayouts jeweils kompatibel zu dem allgemeineren Layouts realisieren zu können, implementieren sie eine an Java-Interfaces angelehnte Gruppe von Prolog-Prädikaten.

```
init(_, _, + language)
get_structures(_, Structures, + language)
check_size_optimum(Structures, + language, + Wgrid, + Hgrid)
check_size_heuristical(Structures, + language, + Wgrid, + Hgrid)
check_size_predefined(_, + language, + wgrid, + hgrid)
print_grid(_, _, + language, Xc, Yc, Structure)
```

Angeboten wird die Prozedur *init*, die den Workspace in einen Zustand überführt, in dem die darauffolgende Sortierung durchgeführt werden kann. Darunter fällt das Löschen von Karten, die alleine zur Formatierung dienen, wie die senkrechten und vertikalen Linien der Tabellen. *get_structures* liefert die vorhandenen Einzelstrukturen in Listenform.
Die drei alternativen *check_size*-Prädikate berechnen die Fläche, die zur Darstellung der Strukturen notwendig sind. In einem Fall berechnen sie eine optimierte Darstellung (*check_size_optimal*), in der Abstände weder zu groß, noch zu klein werden sollten, und sich die Karten nicht überdecken. Dafür müssen alle Strukturen untersucht werden. In der heuristischen Herangehensweise (*check_size_heuristical*) wird eine Struktur zufällig herausgegriffen und für diese die benötigte Fläche ermittelt. Diese Sortierung kann dazu führen, dass die Strukturen dann ihre zugeordnete Fläche überschreiten. *check_size_predefined* bietet eine feste Wertvergabe ohne Berechnungen, die in den meisten Fällen nicht befriedigend sein wird aber durchaus eine Grundstruktur vorgeben kann. Die zwei letzten Varianten ermöglichen es, den Aufwand für die Sortierung zu verringern, der im Allgemeinen recht hoch ist. Alle drei *check_size*-Prädikate liefern als Rückgabewerte die Dimensionen einer Fläche, die im übergeordneten Layout verwendet werden, um das Sortierungsgitter zu bestimmen.
print_grid platziert nach der Berechnung des Gitters die einzelnen Argumente und Tabellen in die Gitterfelder.
Angesteuert werden diese unterschiedlichen Sortierungen durch Karten der Funktionspalette (Tabelle 17).

	check_size_optimum
	check_size_heuristical
	check_size_predefined

Tabelle 17

Drei Funktionskarten leiten unterschiedliche Sortierungsgenauigkeiten ein.

Die Sortierungsprädikate des übergeordneten, perspektivenunabhängigen Layouts lösen zuerst die Initialisierung des zu sortierenden Workspaces aus und fragen dann die zu platzierenden Strukturen an. Für diese wird die erforderliche Größe bestimmt und auf dieser Basis das Gitterlayout berechnet. Dies bezieht zusätzlich noch definierte Abstände zwischen den Strukturen ein und die mögliche oder definierte Anzahl von Reihen und Spalten.

```
sort_o/h/p(_, _, - Language) :-
        init (_, _, - Language),
        get_structures (_,+ Structures, - Language),
        check_size_o/h/p(- Structures, - Language, + Wgrid, + Hgrid),
        get_layout(+ Gridlayout, gridlayout(- Wgrid, -  Hgrid) ),
        …
        arrange(_, _, - Language, - Structures, - Gridlayout).
```

6.10 Zusammenfassung

Mit dem DiscBoard ist ein System entwickelt worden, das Perspektiven über das Medium visueller Sprachen zur Verfügung stellt. Diese Perspektiven wurden als epistemische Formen (Collins & Ferguson, 1993) eingeführt. Der Ansatz wurde in den Kontext von Mind Tools gestellt (Jonassen & Carr, 2000; Lajoie, 1993; Pea, 1985), mit denen Denkprozesse angeregt und die Wissenskonstruktion unterstützt werden soll. Es ist davon ausgegangen worden, dass für verschiedene Arbeitsziele, Aufgaben oder Probleme unterschiedliche Darstellungsmittel geeignet sind (vgl. Suthers, 1999A; 1999B; 1999C; 2003; Carlson, 1995; Cooke, 1994; Jonassen & Carr, 2000; Jüngst & Strittmatter, 1995; Streitz, Haake et al., 1998).

Um eine abgestimmte Arbeit mit den Perspektiven zu ermöglichen, wurde eine Übernahme von Inhalten zwischen den Perspektiven definiert, die alleine auf die Objekttypen und strukturelle Informationen zugreift.

Durch diesen Ansatz werden im wesentlichen zwei Ziele verfolgt:

1. Die Beeinflussung der Externalisierung durch epistemische Formen.
2. Die Unterstützung eines Diskussionsprozesses durch die Abbildungen zwischen den Perspektiven.

6.10.1 Die Perspektiven

Für die Diskussionsunterstützung wurden die folgenden visuellen Sprachen als Perspektiven umgesetzt:

- Die Diskussionsstrukturierung (Protokollsprache).
  Hauptsächlich für die Anfangsphase gedacht, zur Strukturierung der geplanten Inhalte, Sammeln von Besprechungspunkten, Fragen, Aspekten, Ideen, Zusammenhängen, für schnelle Notizen mit wenigen Beitragstypen. Zusätzlich für die Aufnahme von Resultaten als Ergebnis der Diskussion.
- Die Explorationssprache.
  Zur Vor- und Nachbereitung anderer Arbeitsphasen, zur Exploration von Beiträgen, Darstellung von Hintergründen, Zusammenhängen, Meinungen. Erwartet werden umfangreichere Diagramme.
- Die Brainstormingsprache.
  Es können UND/ODER-Graphen entwickelt werden. Ideen bzw. Fragen oder Probleme werden über Alternativen, Optionen, Vorschläge aufgeschlüsselt. Darauf bezieht sich auch der Name der visuellen Sprache. Sinnvoll ist es, diese Darstellung durch einen echten Brainstormingprozess einzuleiten, der vorgegebene Relationen erst einmal gar nicht verwendet. Die Brainstormingsprache ist weniger als Hilfe für das Brainstorming selbst zu sehen, als dass eine Möglichkeit geboten wird, die Ergebnisse nachfolgend zu strukturieren. Am Schluss können Entscheidungen und Präferenzen verdeutlicht werden.
- Die Argumentsprache.
  Darstellung von Argumenten angelehnt an die Argumentstruktur, wie sie in Toulmin (1958) vorgestellt wird. Argumente können frei erstellt werden. Automatisch werden Argumente aus der Brainstorming- sowie aus der Explorationsperspektive gefiltert. Als Arbeitshypothese wird den Darstellungen in der Argumentsprache ein dokumentarischer Wert zugeschrieben.
- Die Pro-/Kontrasprache.
  Darstellung einer einfachen Tabellenstruktur zur Auflistung von Pro- und Kontraaspekten. Es wird davon ausgegangen, dass unvollständige Tabellen eine

Reflexion motivieren. Auch für diese Sprache wird ein besonderer dokumentarischer Wert angenommen. Inhalte für die Tabellen werden automatisch aus der Explorationsperspektive gefiltert.

- Die „Topic Map".
  Übersicht über behandelte Themen, die unter den einzelnen Perspektiven angegeben werden können.

Im Sinne von Collins und Ferguson (1993) stellen die Sprachen die epistemischen Formen bereit, wie sie in Tabelle 18 aufgelistet sind.

Visuelle Sprache	**Epistemische Form**
Diskussionsstrukturierung (Protokollsprache)	Liste
Explorationssprache	Gerichteter Graph
Brainstormingsprache	UND/ODER-Graphen
Argumentsprache	Argumentstruktur nach Toulmin
Pro-/Kontrasprache	Tabelle
„Topic Map"	Gerichteter Graph und Hierarchie

Tabelle 18

Gegenüberstellung der verwendeten visuellen Sprachen zu den darin angestrebten epistemischen Formen („target epistemic form": Collins und Ferguson, 1993, S. 28).

6.10.2 Unterstützung eines Diskussionsprozesses

Ergänzend zu den Perspektiven, die für bestimmte Arbeitsphasen entwickelt wurden, wurde der Diskussionsprozesses unterstützt, indem der Übergang zwischen den Perspektiven erleichtert wurde. Dafür sind Regeln, die Abbildungsvorschriften, definiert worden.

Die Entwicklung der Abbildungen basierte auf den folgenden Hypothesen und Voraussetzungen:

- Geht man von epistemischen Formen aus und drückt diese mit visuellen Sprachen der hier verwendeten Art aus, so sollte es vielfach möglich sein, dafür Strukturmuster als Klasse von Graphen zu definieren. Als Arbeitshypothese wurde davon ausgegangen, dass es möglich ist, über strukturelle Information relevante Teile zu filtern.
- Durch generierte Strukturen, die bestimmten Anordnungen unterliegen, kann die Diskussion motiviert werden. Diese Motivation besteht darin,
  - dass unvollständige Strukturen offensichtlich werden,
  - dass Zusammenfassungen bestimmter Strukturen eine vergleichende Diskussion erleichtern,

  - dass die offensichtliche Verwendung bestimmter Strukturen dazu anregt, diese zu erlernen.
- Es kann und sollte keine eindeutige Reihenfolge definiert werden, in der zwischen diesen Perspektiven gewechselt wird. Deshalb müssen die Abbildungen zu jedem Zeitpunkt korrekt funktionieren, auch wenn Urbild und Bild frei veränderbar sind.
- Da die Perspektiven grundsätzlich für unterschiedliche Arbeitsphasen gedacht sind, ist davon auszugehen, dass die Inhalte in den Arbeitsbereichen nicht deckungsgleich sind, sondern sich ergänzen. Die Abbildung unterstützt deshalb nur den Übergang zwischen den Phasen und bietet einen Ausgangspunkt weiterer Arbeitsschritte. Für die auseinander hervorgegangenen Strukturen bedeutet das, dass sie einerseits abgeglichen werden müssen, solange noch eine Übereinstimmung vorliegt, dass sie ansonsten aber ohne Beschränkungen weiter entwickelt werden können.

Unter diesen Voraussetzungen wurden drei Aspekte realisiert:

- Die Verwaltung der Übereinstimmungen zwischen den Strukturen und Karten der Perspektiven.
- Der Vorgang der Abbildung.
- Die Abbildungsvorschriften.

Verwaltung der Übereinstimmungen. Knotencluster verwalten die Übereinstimmung von Karten zwischen den Perspektiven. Abbildung 85 stellt unten den Datensatz dar, der für ein Knotencluster generiert wird.

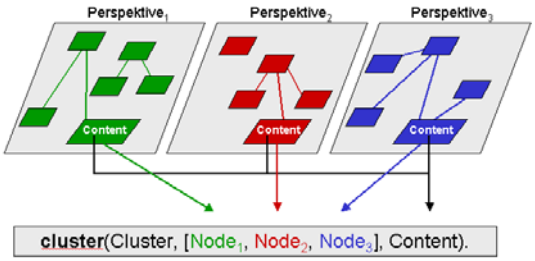


Abbildung 85

Informationen, die in einem Knotencluster verwaltet werden.

Wird eine Struktur abgebildet, so werden für die darin einbezogenen Karten Knotencluster und für die Strukturen Modelle generiert (siehe Abbildung 86). In den Strukturmodellen werden die Strukturen über deren Kantentypen beschrieben, die Knotencluster verbinden.

Eine Übereinstimmung von Strukturen oder Karten zwischen den Perspektiven besteht nach einer Abbildung solange, bis sie durch Nutzerinnen oder Nutzer explizit aufgehoben wird („delayed changes“).

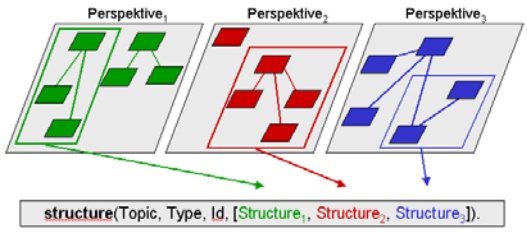


Abbildung 86

Informationen zu den Abhängigkeiten von Strukturen, die aus einer Abbildung hervorgegangen sind.

Abbildung. Eine Abbildung besteht jeweils aus

- einer Auswahl von Informationen, also einem Filter,
- dem Anlegen der Modell, potentiell einschließlich einer Interpretation der Strukturen und Kartentypen
- sowie einer sortierten Darstellung.

Die Filter beziehen sich ausschließlich auf Strukturen, dass sind Teilgraphen der Darstellungen mit den visuellen Sprachen. Sie reagieren dynamisch auf die Fertigstellung von Relationen in den Diagrammen, worauf jeweils untersucht wird, ob eine neue, relevante Struktur entstanden ist oder eine bestehende erweitert wurde. Für dieses Vorgehen werde die Strukturen intern definiert über:

- eine minimale Ausprägung (Kern)
- sowie mögliche Erweiterungen (Extensionen).

Geparst wird das Diagramm ausgehend von einer entstandenen Relation nur so lange, bis eine Neuentstehung oder Erweiterung erkannt ist, bzw. bis erkannt wird, dass dies nicht der Fall ist (lokales Parsen).

Interpretiert werden die gefilterten Strukturen insofern, als dass sie unter der neuen Perspektive im Normalfall anderen Kartentypen und anderen Relationstypen zugeordnet werden. Außerdem kann sich die Topologie der Strukturen ändern. Die Interpretation wird durch die Abbildungsvorschrift spezifiziert.

Abbildungsvorschriften. Die Abbildungsvorschriften für die Karten und Strukturen bilden das Diskussionsmodell. Vor der Abbildung einer Struktur, die zur Generierung des Strukturmodells führt, muss die Struktur, die das Urbild der Abbildung darstellt erkannt und klassifiziert worden sein. Das heißt der Typ der abzubildenden Struktur muss erkannt werden (z.B. Argument). In der Abbildungsvorschrift wird die resultierende Struktur definiert und generiert.

6.11 Diskussion

„Epistemic games", „Epistemic forms". Ein epistemisches Spiel wird nach Collins und Ferguson (1993) durch vier Charakteristika bestimmt: durch die Eingangsbedingung, durch die möglichen Aktionen, durch die Übergangsmöglichkeiten zu anderen epistemischen Spielen und die angestrebte Darstellungsform.

- Die Eingangsbedingung stellt eine Aussage darüber dar, wann ein bestimmtes Vorgehen (Spiel) gewählt werden sollte, also über die Art des Problems. Die Abbildung zwischen den Perspektiven kann als explizite Visualisierung einer Eingangsbedingung interpretiert werden: Liegen generierte Inhalte vor, die aber noch nicht weiter diskutiert wurden, sollte unter dieser Perspektive weiter gearbeitet werden.
- Die Abbildungsvorschriften definieren Übergangsbedingungen zwischen den epistemischen Spielen.
- Die angestrebte Form wird einerseits durch die Syntax der visuellen Sprache spezifiziert, andererseits soll sie durch die Bedeutung der verwendeten Symbole präzisiert werden.
- Die möglichen Aktionen während des Arbeitens unter einer Perspektive („inquiry") hängen spezifisch mit dem konkreten „epistemic game" zusammen. Einfache Aktionen, die beispielsweise typisch für die Arbeit mit Listen sind (Collins & Ferguson, 1993), wie das Hinzufügen von Einträgen, die Kombination von Einträgen, die Aufteilung von Einträgen und das Löschen von Einträgen zielen auf die Restrukturierung und werden generell durch die visuellen Sprachen gut unterstützt.

Divergierende Perspektiven. Grundlegend für einen Lernprozess sind nach Collins und Ferguson (1993) Änderungen der Fragestellung. Sie kennzeichnen, dass ein Problem anders verstanden wird und andere Aspekte wichtig werden. Auch diese Art von Aktionen sind mit dem DiscBoard möglich. Diese inhaltlichen Änderungen von Karten, führen im vorliegenden Ansatz zum Divergieren der Inhalte unter den verschiedenen Perspektiven.

Werden solche Änderungen als Indiz für einen Lernprozess gesehen, so bestätigt das nach meiner Ansicht den Ansatz divergierender Perspektiven, denn einerseits muss die Änderung möglich sein, andererseits kann nicht davon ausgegangen werden, dass sich die Frage unter einer anderen Perspektive gleichermaßen entwickelt. Es ist deshalb sinnvoll, diese Entscheidung den Nutzerinnen und Nutzern zu überlassen, was durch das Konzept der „delayed changes" unterstützt wird. In diesem Sinn können die Markierungen, die

eine Änderung in einem korrespondierenden Workspace anzeigen, auch als zusätzlicher Anreiz zur Diskussion bzw. Reflexion gewertet werden.

Für das DiscBoard muss allerdings kritisiert werden, dass es den Vergleich zwischen einem alten und dem neuen Inhalt nicht unterstützt. Beide Bestandteile müssten parallel betrachtet werden können, um sie gegeneinander abzuwägen. Erst dann wäre eine geeignete Reflexion möglich.

Das Interface. Die Kommunikation zwischen den Teilsystemen C++-MatchMaker, Dalis und DiscBoard ist vereinheitlicht worden (vgl. Mühlenbrock, 2001) und verwendet die drei Sprachpimitive create, modify und delete, um Änderungen in den Arbeitsbereichen zu übermitteln.

Dieses Interfaces stellte sich als relativ unflexibel heraus, wenn zusätzliche Informationen zu denen versendet werden sollen, die in den Parametern vorgesehen sind. Im Rahmen der Überlegungen, ob eine rekursive Gestaltung der Workspace-Struktur ermöglicht werden sollten, war beispielsweise die Übermittlung der „Parent"-Information nicht unmittelbar möglich.

Weiterhin bezieht sich die Verwendungsidee der Sprachprimitive auf die Karten in den Workspaces. Andere Ereignisse, wie Menü-Auswahlen, können damit generell nicht übermittelt werden. Wenn, wie es in der Diskussionsunterstützung durchgeführt wurde, ein Modell in Dalis hinterlegt wird, das auf Ereignisse reagieren soll, so müssen diese immer auf Karteninteraktionen abgebildet werden. Das führte im DiscBoard zur Entwicklung der Funktionspaletten. Dieses Vorgehen ist für Nutzerinnen und Nutzer ungewohnt. Außerdem ist es aufwändig in der Programmierung. Komplexere Interaktionen bzw. Dialogstrukturen sind damit nur umständlich zu implementieren.

Auch der Zustand der Applikation, wie Variablenbelegungen, kann nur schwer als Kontext für die Workspace-Interaktion mit in Betracht gezogen werden.

Wird Dalis ganz allgemein für die Verwaltung einer Datenbasis verwendet, wäre ein gefächertes Spektrum für Anfragen sinnvoll.

Parsing. Für die Abbildungen wurde der Ansatz des lokalen Parsens gewählt. Dafür wird jeweils ein Inkrement in einem Diagramm daraufhin untersucht, ob es eine bestehende Struktur erweitert oder die Grundlage für eine neue bildet. Dieses Erkennen von Strukturen wird dadurch vereinfacht, dass die Gruppe der erkennbaren Strukturen eingeschränkt wird, insbesondere werden keine rekursiven Strukturdefinitionen unterstützt. Dadurch wird auch ein Problem im Umgang mit Zyklen vermieden. Die Definitionen für die Syntax bedeutungstragender Strukturen ist zweischrittig in die Erkennung eines Kernes und einer Extension untergliedert. Diese Bestandteile können

selbst jedoch umfangreicher sein. Ohne solche Einschränkungen wäre das Parsen von Graphen oft zu aufwändig, um es in einer sinnvollen Zeit durchzuführen. Die Einschränkungen für die definierbaren Strukturen werden speziell für den Kontext der Diskussionsunterstützung als gering eingeschätzt, da sich bisher kein Bedarf ergeben hat, komplexeren Strukturen eine Bedeutung zuzuweisen. Das inkrementelle Vorgehen bei den Abbildungen in Verbindung mit der Verwaltung der Strukturmodelle hat sich als gut durchführbar erwiesen.