Authoring Operations Based on Weighted Schemata

Felix H. Gatzemeier[*]

Lehrstuhl für Informatik III, RWTH Aachen
fxg@i3.informatik.rwth-aachen.de

Abstract Conceptual authoring support enables authors to model the content of their documents, giving them both constructive and analytical aid. One key aspect of the constructive aid are schemata that contain proven structures that the author may instantiate in her document.
This paper presents operations offered in an authoring system partially derived from extended schema definitions. The author may instantiate a schema, expanding an existing concept. In contrast with classical CG, the join with the host graph is not maximal, but determined by the author. Generic commands allow concepts and relations to be created, edited and deleted.
Analytical aid based on schemata checks whether a schema instance has been severely mutilated after instantiation. In order to do this, schema elements (concepts and relations) are given weight parameters. Removal of schema instance elements triggers warnings of varying severity and with appropriate fixing operations.

1 Introduction: Authoring Problems and Conceptual Authoring Support

Creating well-structured documents is a complex activity. The author has to solve the problem of conveying a topic, usually in a given amount of space, building on some presupposed knowledge and serving some interest. Key tasks in achieving this are devising a coherent concept structure of the document and maintaining that structure during edits [7]. Conventional authoring environments, however, concentrate on capturing, manipulation and formatting text as a string of characters.[1] Little or no help is available for building the structure.

As an answer, we are working on a prototype of a high-level authoring tool providing Conceptual Authoring Support (CAS) named CHASID. It operates not only on the *presentation* (the formatted string of characters) and the *presentation structure* (the formal hierarchy of sections and subsections), but also on a *content model*. By explicitly storing and editing this model, flaws in it or in the structure of its presentation

[*] This work is funded by the Deutsche Forschungsgemeinschaft (DFG) Grant No. NA 134/8-1.

[1] For simplicity, I restrict myself here to textual documents. The ideas and implementation presented here do not rely on specific media types.

can be detected. CHASID can also help building the document by offering ready-made substructures to be integrated, thus providing some formalized authoring experience.

Conceptual graphs (CG) and schemata lend themselves well to express the content model and the substructures provided by the tool. The graph model has the appropriate expressiveness and the editability of schema corresponds with the proposing character of the substructures. [7] describes the overall CHASID authoring scenario, where the concept graph is, among other operations, built by instantiating schemata chosen from a schema browser. This implies extensions to the definition of a schema to provide documentation for the author. Some required conditions given in type definitions are enforced. More fundamental direction is given by document patterns presented in a pattern browser.

When joining a schema to a graph, a maximal join is required by [12, Definition 4.1.3, p. 130]. In an interactively built graph, the author may prefer some compatible concepts not joining, or an instance even if the join on the expanded concept is not maximal. So, instead of a maximal join, this article proposes an interactive join.

The use of a schema as a building block resembles a macro that, once executed, leaves new concepts and relations as traces in the document. The author may afterwards modify the traces freely, obscuring or even removing them. When seen as a guide for document construction, however, the schema should retain some integrity across modifications, or it should be abandoned when crucial elements are removed.

To allow this behavior, I propose *weighted schemata*. In such a schema, weights are attached to concepts and relations that are remembered in the schema instance. If crucial elements of a schema instance are later removed, warnings are issued. Optional elements, on the other hand, may be freely removed. Currently, three weights are used: optional, important and crucial.

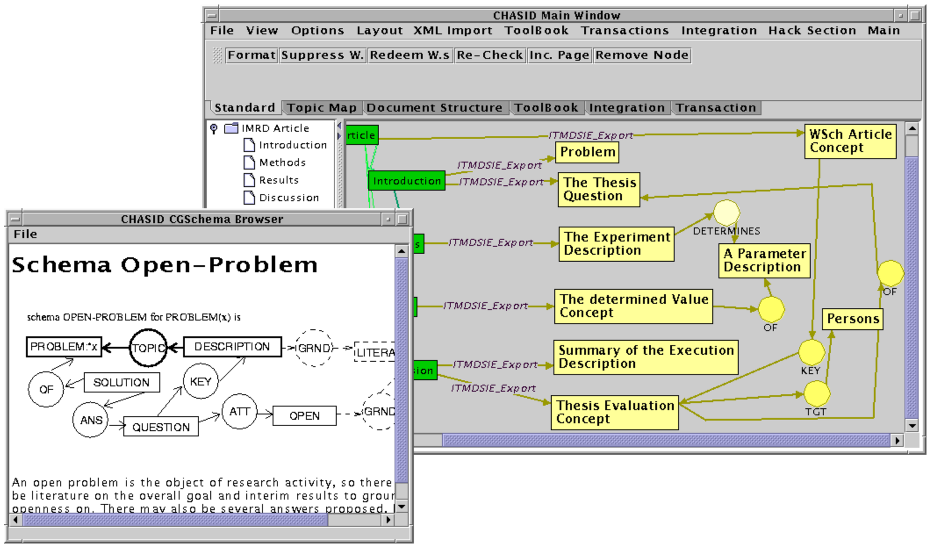
This paper employs an Introduction-Methods-Results-Discussion structure to support the thesis that these extensions allow for a number of useful operations and diagnostics.

The Method section, section 2, describes the operations and diagnostics available. It begins with the generic operations independent of the underlying canon, including the modified procedure of joining a new schema instance to the host graph. It also defines weighted schemata as a basis for diagnostics. For comparison, it also mentions specifically implemented diagnostic patterns. Some notes on the implementation in PROGRES[2], a non-CG graph-based language, are also given.

To illustrate the usefulness, an extended example of instantiation, modification, warning and correction is given in section 3.

In the discussion in section 4, benefits and limitations of schema-based authoring are discussed. Some deviations from CG definitions are recapitulated. Section 5 wraps up the document and gives an outlook on medium-term plans.

[2] PROgrammed Graph REwriting Systems, [10; 11], `http://www-i3.informatik.rwth-aachen.de/research/progres`.


CHASID Main Window
File View Options Layout XML Import ToolBook Transactions Integration Hack Section Main
Format Suppress W. Redeem W.s Re-Check Inc. Page Remove Node
Standard Topic Map Document Structure ToolBook Integration Transaction
IMRD Article
Introduction
Methods
Results
Discussion
rticle
Introduction
ITMDSIE_Export
WSch Article Concept
Problem
The Thesis Question
The Experiment Description
DETERMINES
A Parameter Description
The determined Value Concept
OF
Summary of the Execution Description
Persons
KEY
Thesis Evaluation Concept
TGT
CHASID CGSchema Browser
File
Schema Open-Problem
schema OPEN-PROBLEM for PROBLEM(x) is
PROBLEM:*x
TOPIC
DESCRIPTION
GRND
LITERA
OF
SOLUTION
KEY
ANS
QUESTION
ATT
OPEN
GRND

An open problem is the object of research activity, so there
be literature on the overall goal and interim results to grour
openness on. There may also be several answers proposed.

Figure 1. Sample CHASID window with Introduction-Methods-Results-Discussion (IMRD) schema instance (see section 3) and Open-Problem schema in schema browser

2 Method: Operations, Diagnostics and Implementation

CHASID works as an extension to existing authoring applications. Figure 1 shows the main user interface. It provides additional views on the document, the primary ones being a tree view of the presentation structure and a graph view for the content model. Concepts and relations are of types known to CHASID (the **known types**) and carry arbitrary **type labels**, which the author may use to differentiate sub-types for her information. Concepts additionally have arbitrary **names**. Referents are not considered. Supplementary and other specialized information (see 4.1) is stored in the document graph in ‘light-weight’ nodes and edges.

The presentation structure is linked to the content model through import and export relations. A section exports a concept, if reading the section is required to fully understand the concept. It imports the concept, if some understanding of the concept is required to understand the section.

This section characterizes kinds of editing operations available, ways to design warning functionality and some aspects of the implementation.

2.1 Editing Operations

The editing operations are classified here into three groups according to their origin: generic, schema instantiation, and hand-implemented.

Generic Operations. These work without reference to any canon. They instantiate and remove concepts and relations. Type labels and names of concepts and relations may be changed freely, while the known type remains fixed. Concepts may be **merged**, which is a combination of restriction and join with relaxed conditions, where known types, type labels, and names do not have to fulfill any conditions.

Operations to handle warning messages and conditions also fall into this category, but are described in section 2.2.

Schema-based Operations. Schemata are presented to the author in a schema browser to be instantiated in the graph. If the author chooses to instantiate a schema, she has to indicate the node to be extended and may further *determine the join* of the schema with the existing graph. For each concept in the schema, she may give a concept in the graph it is to join with. The instantiation command then creates instances of the remaining concepts and of the relations of the schema body in the host graph and draws the required arcs.

Schema instantiation is thus based on a controlled or **interactive join** instead of a schematic join. The interactive join is not required to be maximal, not even locally maximal.

Hand-implemented. There are also specialized commands building on the known types. For example, the author may move a section in the presentation structure below another one instead of inserting and deleting relations or edges. The command uses the known types of division, first-descendant and next-sibling in its implementation.

2.2 Diagnostics and Weighted Schemata

The content model and presentation structure are evaluated according to information derived from schema instantiation and hand-implemented rules. The result of the evaluation are warning markers that are handled by generic operations.

Weighted Schemata. Each element (concept or relation) of a schema has one of three **weights**.

OPTIONAL: The element may be removed without consequences.
IMPORTANT: The element should not be removed, except for special circumstances.
CRUCIAL: If the element is removed, the schema does not apply any more.

Removal covers both deletion of concepts as well as disconnection of concepts, as that means removal of a relation. There is currently no provision for unwanted elements, whose addition (rather than removal) would trigger warnings. Elements may be modified (given new type labels or names) in any case without entailing warnings.

The different weights are shown in the graphical notation by different border styles: CRUCIAL elements are solid bold, IMPORTANT ones solid plain and OPTIONAL ones dashed plain, as in figure 4.

As an additional documentation measure, the **role** of each schema element is also recorded. This is used to assemble more informative message texts.

Formally, weights and documentation extensions can be added to the definition of a schema given in [12, Definition 4.1.1, p. 129], as follows:

> A **weighted schematic cluster** for a type t is a set of weighted monadic abstractions $\{n_1 : \langle \lambda a_1 u_1, d_1, \rho_1 \rangle, \ldots n_n : \langle \lambda a_n u_n, d_n, \rho_n \rangle\}$, where each formal parameter a_i is of type t. Each weighted abstraction $n_i : \langle \lambda a_i u_i, d_i, \rho_i \rangle$ in the set is called a **weighted schema** for the type t, with n_i being the name of the abstraction, d_i a description for the author and ρ_i a function mapping the concepts and relations of u_i to weights and roles from ({OPTIONAL, IMPORTANT, CRUCIAL} $\times$ String-label).

For readability, I use the regular names 'schema' etc. in the weighted instead of the classical meaning. As only the weighted definitions are used, there should be no cases of doubt.

During schema instantiation, the weights of the schema elements are recorded for their counterpart in the host graph (displayed in figure 6). Thus, graph elements now have weights specific to the schema contexts in which they are used. This change to the graph model can be formally expressed as a set of tuples of schema names and (partial) instantiation functions mapping elements from u_i to the host graph.[3]

Hand-implemented Warning Rules are patterns of unwanted graphs. To describe these, the programmer may use the full set of the underlying PROGRES facilities, most frequently nodes (single nodes or node sets) and edges that must or must not match, attribute restrictions, and edge path expressions. For example, there are warning rules to find presentation structure sections that contain just one subsection, or a section that imports a concept that comes before a section which exports it.

Warnings are attached to the graph as additional nodes with edges to all concerned concepts and relations and a textual attribute containing the warning message. They show up as non-concept nodes in the graphical display and as annotations in the externally edited conventional document. If the condition causing the warning is fixed, the warning and all its representations are removed.

All warning messages can be manipulated with these generic operations:

- Suppress a specific warning, confirming the change that triggered this warning. The warning's representations disappear.
- Suppress all warnings of a type, indicating that this type does not apply to the current document. The representations of all warnings of this type disappear.
- Resurrect all suppressed warnings about one concept or some context,[4] to re-check whether these warnings were intended for suppression. The representations of the warnings re-appear as they were when suppressed.

[3] This is compatible with the graph model described in [4].

[4] The context actually requires a specific command to determine, for example, a subtree in the presentation structure.

Generic operations helping to cure warnings about crucial or important elements missing from schema instances are:

– Re-connect a missing important or crucial element and
– Dissolve the schema instance, leaving the concepts and relations as plain graph elements.

2.3 Implementation Issues

CHASID is implemented using the high-level graph transformation language PROGRES, whose graph model is the directed, node- and edge-typed graph with multiple inheritance and attributes defined on node types. Edges may only connect two nodes, there are no hyper-edges, as in n-adic relations with $n \geq 3$. A language feature used extensively in the implementation of CHASID are graph rewriting operations (productions), which partially replace matches of graph patterns with new graphs. Graph modifications can also be defined in a more imperative manner in so-called transactions.[5] Concepts and relations are actually implemented as nodes, arcs as edges.

Generic and hand-implemented operations are arbitrary productions and transactions, with the generic operations using only knowledge of more abstract node types.

From the PROGRES specification, C Code is generated, which is used in a framework implemented in Java in which the productions of the specification can be executed interactively [9]. The framework is extended with custom Java code for specialized views and integration with the ToolBook multimedia authoring system.

The implementation instantiates and joins a schema in several steps:

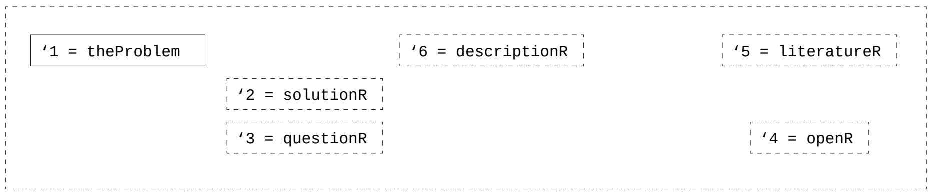
1. The author is asked for *joining concepts* from the host graph;
2. A *complete instance* of the schema is created in the host graph, with *coreference edges* from the concepts to their joining concepts, as far as given by the author;
3. For each element in the instance just created, a *stub node* is created that stores role and weight. For the entire instance, an *instance marker node* is created and connected to the stubs.
4. Coreferring concepts are *joined*; and
5. Parallel relations are *simplified*.

Step 1 is performed through an user interface element automatically generated from the template instantiation transaction. Steps 2 and 3 are the template-specific core, consisting of transactions derived from schema descriptions. This derivation is deterministic, but not automated. As an example, figure 2 shows a simplified graph production performing step 2 for the OPEN-PROBLEM schema (which is visible in figure 1).

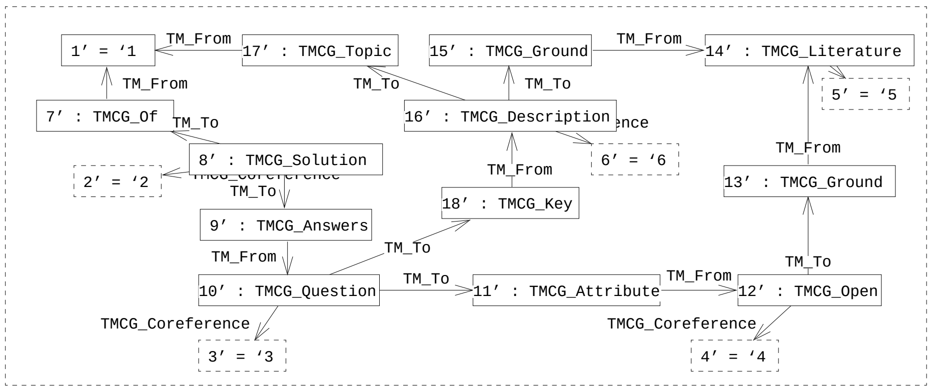
The production begins with the keyword `production`, followed by its name and parameter list (collapsed to an ellipse), a match pattern (the upper dashed box), and a replacement pattern (the lower dashed box). Further parts are possible and present in the actual production, but collapsed to ellipses here. Solid rectangles inside the match

[5] While the terms production and transaction are used consistently here, understanding the difference should not be required to understand the text.

```
production TMCGT_OpenProblem_Instantiate[...] [...] =
```



```
::=
```



```
[...]
end;
```

Figure 2. PROGRES production to instantiate an OPEN-PROBLEM schema, step 2

pattern denote nodes that must be matched, while dashed rectangles denote nodes that may be matched or not. Matched nodes may be kept in the replacement pattern, denoted with equalities. Solid rectangles with a type name behind a colon in the replacement pattern indicate nodes to be created.

The optional nodes in the match pattern indicate the optional joining nodes from the host graph. During execution of this production, they are connected with coreference edges with the respective nodes of the new schema instance. PROGRES’ parameter type checks enforce type conformance on the joining nodes (to the extent available in the known types), so only conforming nodes are unified here. A further transaction creates the instance node and its stubs by calling on sub-transactions for attaching optional, important or crucial parts in their roles.

The connecting steps (4 and 5) are generic actions with no connection to particular schemata. As coreference edges have been drawn by the instantiation productions, the joining production takes two concepts with a coreference edge between them and merges them. Relations to be simplified are detected by a production that requires identical source concept sets and identical targets. One of the relations is then removed.

When schema instance elements are removed, the stub nodes remain. Warnings are then created by warning patterns that test for crucial or important stub nodes that have no corresponding node and no such warning. The warning message states the role of the missing element and suggests operations based on the weight stored in the stub. If crucial nodes have been removed, it recommends removing the schema instance record, consisting of the instance node and the stubs, as the remaining nodes no longer represent the original schema. This can be done with a generic production, since the instance/stub subgraphs are unambiguous. If merely important nodes have been removed, the stub may be removed to remove the warning. If optional nodes are removed, the stub is quietly removed. The warning also disappears if the author reconnects an appropriate element.

3 Results: An Usage Example

As an example, parts of the structure of the present article are built up, modified and corrected. Even though this is an idealized account, similar considerations have been applied during writing.

The first snapshot, figure 1 on page 63 shows the main graph as an instance of a Introduction-Method-Results-Discussion (IMRD) schema. This has been chosen to achieve a clear distinction between descriptive and evaluative parts. This construction is very common for empirical reports, up to being required in behavioral science [5].

Next, the main thesis is modeled in order to establish a focus. It states that extending schema with weight information allows significantly more useful authoring support. This model is shown in figure 3. As this substructure is the core of the creative authoring work, there are no schemata available to support the author here.

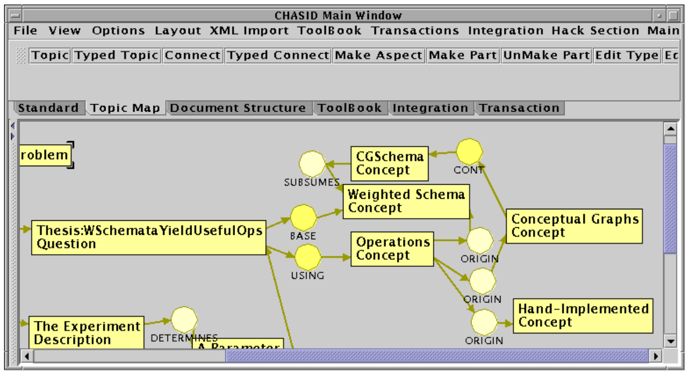
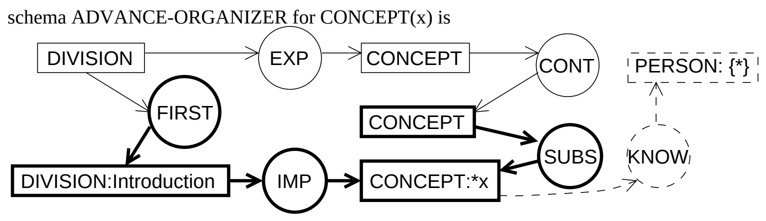

CHASID Main Window
File View Options Layout XML Import ToolBook Transactions Integration Hack Section Main
Topic Typed Topic Connect Typed Connect Make Aspect Make Part UnMake Part Edit Type Ec
Standard Topic Map Document Structure ToolBook Integration Transaction
roblem
SUBSUMES
CGSchema Concept
CONT
Weighted Schema Concept
Thesis:WSchemataYieldUsefulOps Question
BASE
USING
Operations Concept
Conceptual Graphs Concept
ORIGIN
ORIGIN
Hand-Implemented Concept
ORIGIN
The Experiment Description
DETERMINES
A Parameter

Figure 3. Cutout of the document model containing the thesis

Further edits and text creation lead to a first version of the document. Reading it, the author notices that the introduction is rather hasty, and the reader is shoved into an array of unknown terms. Skimming the schema browser for structures of introductory



To prepare defining a concept in the document, you can freshen super-concepts of it that should be known to the reader in an advance organizer. The advance organizer does not introduce the new concept, but should activate the right conceptual context in the reader's mind. Lead the reader gently to the gates of the presentation.
You may structure the connection between presentation structure and content differently, or you may choose not to have the reader modeled at all while still building an advance organizer.

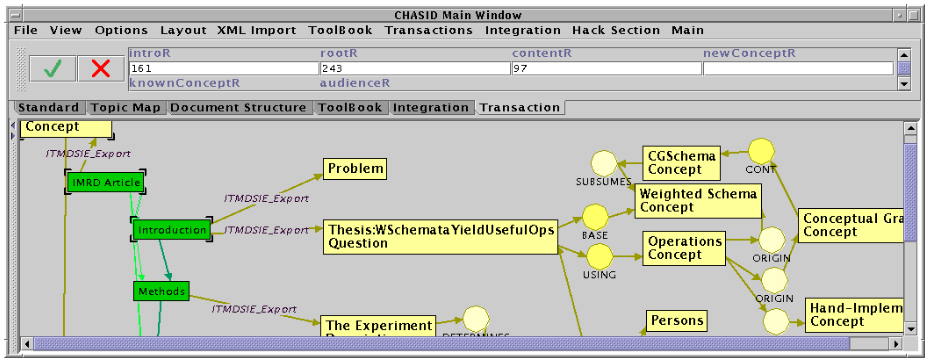
Figure 4. Schema ADVANCE ORGANIZER

sections, the author finds the advance organizer schema. An advance organizer is located early in a document and activates concepts already known to the reader in order to prepare him to add new refining concepts underneath them. Numerous psychological studies have verified advance organizers to have a positive influence on text recall, so it is listed in psychologically oriented writing instructions like [1].

The schema for an advanced organizer (figure 4) contains part of the presentation structure to identify the introduction and parts of the content model to identify presupposed knowledge of the reader and the parts to be introduced. The crucial core of the pattern are the introduction, the subsumed new concept and the presupposed concept. The structural model connecting top division and content is optional to avoid harsh warnings should the author construct this otherwise. The reader model is deemed optional, for authors that rather keep track of that themselves to gain smaller graphs with fewer layout problems.

The process of instantiation is illustrated in figures 5 and 6. The concepts of classical and weighted schemata are given as referents and the new concepts and relations are created. Figure 6 also displays the schema instance and stub nodes.

During further editing, the author introduces additional concepts and connects them with the advance organizer pattern. In the end, the paper becomes too long and must be truncated. As the author does not want to lose any of the new material to be presented, he shortens the introduction to mention only his previous work, which in turn introduces the used terms. The note on schemata is therefore deleted, together (manually) with the import relation.



introR 161
rootR 243
contentR 97
newConceptR
knownConceptR
audienceR

Figure 5. Giving coreferents for schema instance

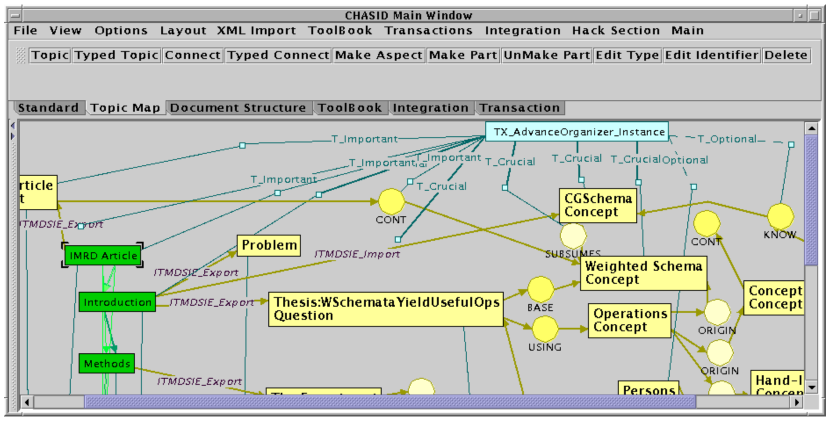
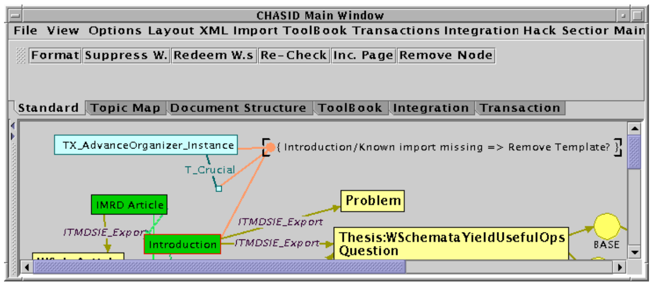

CHASID Main Window
File View Options Layout XML Import ToolBook Transactions Integration Hack Section Main
Topic Typed Topic Connect Typed Connect Make Aspect Make Part UnMake Part Edit Type Edit Identifier Delete
Standard Topic Map Document Structure ToolBook Integration Transaction
TX_AdvanceOrganizer_Instance
T_Important
T_Optional
T_Crucial
T_CrucialOptional
rticle
ITMDSIE_Export
ITMDSIE_Import
IMRD Article
Introduction
Methods
Problem
Thesis:WSchemataYieldUsefulOps Question
CGSchema Concept
Weighted Schema Concept
Operations Concept
Concept Concept
Hand-I Concep
Persons
CONT
KNOW
SUBSUMES
BASE
USING
ORIGIN

Figure 6. The newly created Advance Organizer

Since the import relation has been a crucial part of the schema instance, a warning to this effect appears, as in figure 7. The author can now consciously decide whether to let go of the advance organizer due to space restrictions, or to uphold this structure, making cuts elsewhere.

4 Discussion

This discussion addresses several aspects of the authoring support presented here. There is the discussion of the provided functionality itself, its benefits and shortcomings. Looking at the implementation, some differences between the abstract conceptual graphs view and the concrete nodes and edges appear. Finally, there are other connections to related work.



{ Introduction/Known import missing => Remove Template? }

Figure 7. The ‘removed crucial part’ warning

4.1 Benefits and Shortcomings

The instantiation and weighting mechanisms presented here allow creation of graph models with parts of discriminated importance with some guidance. Giving reference points at instantiation time is an intuitive and efficient way to integrate schemata into graphs under construction. Weights are an useful addition to avoid less-than-helpful warnings about lost fringe elements, and to post warnings about severe mutilations where appropriate.

A shortcoming of the interactive join is that there are some places where referents are ‘obvious’, for example the root and content nodes in the advance organizer: there is only one root, which is linked to the one overall content concept. Grasping this obviousness formally could take the form of locally maximal joins, context search heuristics (for example using only unambiguous locally maximal joins), or further parameterization of the schema. Either possibility makes the matching more complicated. This may conflict with the requirement that operations must remain predictable for the author.

The current implementation detects only missing parts; a schema cannot inhibit relations being attached to it. This may be added by incorporating negated nodes and relations, for which additional warning patterns would have to be implemented that check negation instead of stubs. These patterns would lead in the direction of satisfiability tests.

CHASID does not attempt to infer anything from the document content. Solely the formal presentation structure is integrated automatically. Some heuristics or natural language processing may be useful here, but are not in the focus of our research.

4.2 Deviations from CG

For various reasons, CHASID is not fully ‘CG-compliant’ in its operation. For comparison, I list some of the interesting deviations here:

– Nodes and therefore concepts carry attributes within them, not attached through (ATTR) relations. This reduces the number of nodes in the graph and allows some simplifications in the technical handling. (ATTR) relations may still be created in the content model.

– When relations are simplified, the order of the incoming arcs is ignored. Arcs are implemented as edges, which have neither attributes nor identity in PROGRES. This is a technical limitation. As only dyadic relations have occured, this is not yet pressing.
– The presentation structure is implemented using nodes and edges rather than concepts and relations. This simplifies hand-implemented operations, but causes additional effort for schema instantiation.
– The arbitrary type label softens the type concept. There are no constraints on editing them, but some operations check their equality. This may be just the flexibility needed, or it may unexpectedly cause operations to fail.

4.3 Related Work

CHASID being an application (a program aimed at supporting users in solving real-world problems) based on (Conceptual) Graphs, it may be interesting to classify it according to [3] into natural language processing, information retrieval, knowledge acquisition, requirements engineering, or miscellaneous. CHASID is, however, not really an artificial intelligence application, as it does not use the document structure (which would be the model of the natural language text or the basis of document retrieval, or the acquired knowledge) for deduction — unless evaluating the structure is regarded as deducing warnings. It merely supports the author in building a good structure, avoiding structures that are known as problematic. So, it would be a dedicated system of the 'Miscellaneous' class, on the fringes of NLP.

Argumentative structures have been explored analytically in [8]. The result is a system of data structures consisting of the text, the thematic map (a model of the subject area), the author's/discursive goals structure, the conceptual map (charting the path of argument in the subject area) and the evaluation map (evaluating of the subject with respect to the subject area). Since the aim of CHASID is not analytical, its model is simpler. The presentation structure resembles text data structure, but does not model argumentative units. Depending on the modeling by the author, the content model may contain parts of a goals structure, evaluation map, and thematic map. Keeping these aspects in single graphs would probably overtax average authors. CHASID does not contain a model that traces the flow of argumentation like the conceptual map. Such a view would probably be advantageous, since losing this flow is a frequent flaw in documents.

The idea of using structures with attached weights is based on CIDRE.[6] To support a document structure recognition process, certain hierarchical arrangements in SGML are given a probability rating. This uses a logical extension of the proven optical character and text recognition technique of statistical n-gram modeling. The weights are derived from a body of structured documents, which should allow this technology to scale well to large document management systems, as it does not require human intervention. This is, however, chiefly possible on the basis of the formal nature of the structure. The

[6] Cooperative & Interactive Document Reverse Engineering, [2].

principle accounts well for the fact that formal structure definitions are bound to cater for exotic cases in order not to be too restrictive, so that the regular cases may get lost.

Well-documented document type descriptions, such as DocBook [14], provide a host of technical type and schema definitions. Relevant content structures occur only rarely there. They are, therefore, at most a starting point.

5 Conclusion and Plans

Conceptual authoring support systems have to cope with the cost of adding the structure. While having the structure may pay off for the author in the long run, average authors do not look so far ahead. Patterns and schemata help in this respect by guiding the author through planning the document, which can speed up this process. Schemata extend a current working graph, so that instantiation has to take creating connections into account. I have proposed an interactive join for this here. Later in document production, the conceptual structure behind the document can be useful in warning about local changes that induce wider-scope problems. Using weights in schemata offers a way to encode knowledge about problems in the schema instance scope easily. This allows non-programmers to extend the authoring environment significantly.

This article has discussed the problems using an article as the example document. Given the current tools, a content model of a 15-page article may realistically be handled. The writing span of such a document is so short, however, that this will frequently not pay off. When considering a more coarse-grained model of a book, a benefit becomes more likely: the time spent writing increases, making it easier to get lost. There are, however, even less accounts of discernible reusable structures in books than there are in articles.

To fully deliver on the promise of extending the authoring environment with schemata, the necessary productions should be generated from schema definitions constructed in a schema editor. This seems to be possible, but has not yet been done. This may also lead to a more appealing user interface for giving the co-referring concepts.

As content graphs grow larger, problems of stable semi-automatic layout arise. A related topic are navigation and visibility operations.

For reasonable text production, integration with word processors remains an open problem, since elements are not as clearly discernable there.

References

[1] S.-P. Ballstaedt. *Wissensvermittlung: Die Gestaltung von Lernmaterial (Conveying Knowledge: The Design of Teaching Material).* Beltz, Psychologie-Verlags-Union, Weinheim, 1997. ISBN 3-621-27381-6.

[2] R. Brugger, F. Bapst, and R. Ingold. A DTD Extension for Document Structure Recognition. In R. D. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging, and Digital Typography*, volume 1375 of *Lecture Notes in Computer Science*, page 343ff. Springer,

1998. URL `http://link.springer.de/link/service/series/0558/papers/1375/13750343.pdf`.

[3] M. Chein and D. Genest. CGs Applications: Where Are We 7 Years after the First ICCS? In Ganter and Mineau [6], pages 127–139.

[4] D. Corbett. A Framework for Conceptual Graph Unification. In Stumme [13], pages 162–174.

[5] *Richtlinien zur Manuskriptgestaltung (Guidelines for mauscript design)*. Deutsche Gesellschaft für Psychologie (German association for psychology), Göttingen, second edition, 1997.

[6] B. Ganter and G. W. Mineau, editors. *Proc. International Conference on Conceptual Structures 2000*, volume 1867 of *LNAI*, 2000. Springer. ISBN 3-540-67859-X.

[7] F. Gatzemeier. Patterns, Schemata, and Types — Author Support through Formalized Experience. In Ganter and Mineau [6], pages 27–40.

[8] H. Irandoust and B. Moulin. Pragmatic representation of argumentatitve text: A challenge for the conceptual graph approach. In Stumme [13], pages 30–44.

[9] D. Jäger. Generating tools from graph-based specifications. In J. Gray, J. Harvey, A. Liu, and L. Scott, editors, *Proceedings First International Symposium on Constructing Software Engineering Tools (CoSET '99)*. University of South Australia, School of Computer Science, 1999.

[10] M. Nagl, editor. *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, volume 1170 of *LNCS*. Springer, Heidelberg, 1996. ISBN 3-540-61985-2.

[11] A. Schürr. *Operationelles Spezifizieren mit programmierten Graphersetzungssystemen (Operational specification with programmed graph rewriting systems)*. PhD thesis, RWTH Aachen, Deutscher Universitätsverlag, Wiesbaden, 1991.

[12] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. The Systems Programming Series. Addison-Wesley, Reading, MA, 1984. ISBN 0-201-14472-7.

[13] G. Stumme, editor. *Working with Conceptual Structures: Contributions to ICCS 2000*, 2000. Shaker Verlag. ISBN 3-8265-7668-1.

[14] N. Walsh and L. Muellner. *DocBook: The Definitive Guide*. O'Reilly, Oct. 1999. ISBN 1-56592-580-7. URL `http://www.oasis-open.org/docbook/documentation/reference/html/docbook.html`.