Titre: Title:	Projet ContextEdit - conception du SMML, une structure électronique de document basée sur les macrostructures de Van Dijk
Auteur: Author:	David Gagnon
Date:	1999
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gagnon, D. (1999). Projet ContextEdit - conception du SMML, une structure électronique de document basée sur les macrostructures de Van Dijk [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8811/

UNIVERSITÉ DE MONTRÉAL

PROJET CONTEXTEDIT – CONCEPTION DU SMML. UNE STRUCTURE ÉLECTRONIQUE DE DOCUMENT BASÉE SUR LES MACROSTRUCTURES DE VAN DIJK

DAVID GAGNON
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
MAI 1999

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

PROJET CONTEXTEDIT – CONCEPTION DU SMML, UNE STRUCTURE ÉLECTRONIQUE DE DOCUMENT BASÉE SUR LES MACROSTRUCTURES DE VAN DIJK

présenté par: GAGNON David

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. BOUDREAULT Yves, M.Sc., président

M. ROBILLARD Pierre N., D.Sc., membre et directeur de recherche

M. ROBERT Jean-Marc, D.Sc., membre

REMERCIEMENTS

J'aimerais exprimer ma reconnaissance aux personnes qui ont contribué à la réalisation de ce projet de recherche.

Je tiens à exprimer ma profonde reconnaissance à mon directeur de recherche, M. Pierre N. Robillard qui, en acceptant de diriger ma recherche, a su me mettre sur les traces d'un sujet passionnant. Merci pour avoir partagé temps, connaissances et expérience.

Toute ma reconnaissance va également à mes collègues du laboratoire RGL, M. Patrick d'Astous et M. Simon Labelle qui, dans une ambiance de travail toujours sereine et stimulante, ont contribué de multiples façons à l'aboutissement de ce mémoire. Une mention va d'ailleurs à Didier, Pierre, Marcos et Éric dont j'ai eu le plaisir de superviser les stages et dont le fruit de leurs travaux m'a été fort utile.

Je tiens à remercier sincèrement Jean-François, mon ami avec qui j'ai affronté Poly, pour ses idées, ses conseils judicieux ainsi que pour la revue de mon texte.

Évidemment, les incidences de ces nombreux mois de travail ont été particulièrement rigoureuses sur ma vie privée. Ma gratitude la plus profonde, je la réserverai donc pour Claudia, qui a assumé beaucoup plus qu'une juste part de sacrifices, sans pour autant jamais cesser de me prodiguer les encouragements et l'aide dont j'avais besoin.

Enfin, je désire exprimer ma reconnaissance à ma famille qui m'a encouragé et supporté pendant mes années d'études.

Un merci de plus à M. Robillard pour son soutien financier qui fut grandement apprécié.

RÉSUMÉ

Le document technique constitue le seul moyen tangible pour représenter le logiciel ainsi que le processus logiciel. Typiquement, les projets d'envergure produisent une centaine de documents de toutes sortes. Le document technique est donc à la base du développement logiciel et constitue le principal canal de communication utilisé par les ingénieurs pour communiquer entre eux.

Un projet[1] de conception d'un simulateur de réseau de Pétri a nécessité la rédaction de plusieurs dizaines de documents techniques. L'outil d'édition utilisé est Microsoft Word™, ce qui est souvent le cas dans plusieurs projets logiciels. Ce type d'éditeur est mal adapté pour la rédaction de documents techniques car il met l'accent sur la présentation et la mise en page des documents. Le document technique constitue avant tout un moyen de communiquer de l'information et non un moyen de rendre l'information agréable visuellement.

L'objectif principal de ce projet de recherche consiste à trouver un moyen d'améliorer le canal de communication qu'offre le document en définissant un format électronique adéquat.

Ce mémoire présente les travaux de DeRose et al. qui démontrent que les modèles électroniques existants sont inadéquats principalement parce qu'ils limitent l'élaboration de traitements évolués. DeRose et al. proposent la structure électronique OHCO[2], une

---

[1] Projet réalisé en 1997 au laboratoire de recherche en génie logiciel de l'École Polytechnique de Montréal, par une équipe d'au moins quatre ingénieurs durant environ un an.

[2] Ordered Hierarchy of Content Object.

structure mieux adaptée qui devrait être implantée dans les outils d'édition. Ils démontrent aussi qu'un document peut avoir plus d'une structure de type OHCO valide.

—

On présente par la suite les travaux de Van Dijk sur la macrostructure sémantique. Celui-ci argue qu'un texte n'a pas seulement une microstructure exprimant les relations entre les mots et les phrases, mais aussi une structure plus générale qui en définit la cohérence et l'organisation globale. Il définit la structure globale d'un discours qui permet d'en organiser les idées à l'aide des notions de superstructure et de macrostructure notamment.

À la lumière de tout cela, on propose le SMML (*Semantic Macrostructure Markup Language*), un modèle électronique de type OHCO qui permet de représenter la structure globale, telle que vue par Van Dijk, à même le texte. Cette représentation électronique permet de mettre en relief la structure sémantique d'un document contrairement à sa structure de présentation.

On introduit la notion de classe de macrostructure. Celle-ci fournit un degré d'abstraction intéressant et ouvre la porte à la définition de traitements informatiques évolués. De plus, on propose l'utilisation de patrons pour la définition d'un catalogue de macrostructures, étant donné qu'il ne semble pas exister d'inventaire de macrostructures actuellement disponible qui soit compatible avec le modèle SMML.

On propose également une présentation afin d'afficher un document SMML à l'écran ou sur papier. Cette proposition tient compte des travaux de Bernardt. Elle permet notamment d'expliciter la structure hiérarchique et de délimiter graphiquement les macrostructures dans le texte.

Une validation théorique du modèle SMML est faite à l'aide des travaux de Van Dijk sur la macrostructure et la cognition. Il est ressorti que le modèle SMML devrait permettre d'alléger la tâche de compréhension du lecteur et donc de réduire le coût d'utilisation du canal de communication qu'est le document.

On pousse plus loin la validation théorique afin de raffiner les attentes quant aux incidences cognitives du SMML, en démontrant que la macrostructure respecte la notion d' « Advance Organizer (AO) ». Les recherches faites sur l'AO montrent que les bénéfices de l'utilisation d'une telle construction peuvent varier en fonction de la connaissance, de l'expertise et de la technique de lecture du lecteur.

Finalement, on présente un exemple pratique d'un document SMML, d'un catalogue de macrostructures et d'une structure spécifique nommée SMML-AR pour l'édition d'articles scientifiques. On présente aussi une analyse comparative afin de montrer certains avantages reliés à l'utilisation du SMML.

ABSTRACT

Technical documents constitute the only tangible way to represent both software and the software process. Typically, large projects lead to hundreds of documents of all kinds. Technical documents are the principal communication channel used by engineers and lay, as such, the foundations of software development.

In a recent project[3] to develop a Petri net simulator where dozens of technical documents where written, the editing tool of choice was Microsoft Word ™. This is the case in a large proportion of development projects. This type of editor is not adapted for writing technical documentation because the emphasis in the editor is put on layout and formatting. But technical documents are first and foremost a way to communicate information, not a way to render information in a pleasant way.

The main objective of this research project is to find a way to improve the communication channel that is the technical document by defining an adequate electronic format.

This Master's thesis presents the work of DeRose and al., who clearly shows that existing electronic models are inadequate, mostly because they limit the elaboration of advanced treatments. DeRose and al. proposes the OHCO[4] electronic structure, better adapted and which should be implemented in editing tools. He also shows that one document can have more than one valid OHCO-type structure.

---

[3] This project was conducted in 1997 in the Software Engineering Laboratory at the École Polytechnique of Montreal, by a team of 4 software engineers over a period of one year.
[4] Ordered Hierarchy of Content Object

We then present Van Dijk's work on semantic macrostructure. Van Dijk argues that text is not only structured at a microscopic level, which expresses relations between words and sentences, but also at a macroscopic level that defines the global organization and coherence of the text. He thus defines the global structure of a discourse, allowing the organizing of ideas through the use of superstructure and macrostructure notions.

In light of all this, we propose the SMML (Semantic Macrostructure Markup Language), an OHCO-type electronic model that allows one to represent the global structure, as defined by Van Dijk, within the text itself. This electronic representation brings out the semantic structure of a document, instead of the structure of the format.

We introduce the notion of a macrostructure class. This class provides an interesting abstraction level and leads the way to the definition of advanced computer processing of documents. In addition, we propose the use of patterns for defining a catalog of macrostructures, since there does not seem to be any existing SMML-compatible inventory of macrostructures available.

We also propose a format adapted to the display of a SMML document, on screen or on paper. This proposal takes into account the works of Bernardt. It allows the clarification of the hierarchical structure and the graphic delimitation of macrostructures in the text.

A theoretic validation of the SMML model is done through Van Dijk's work on macrostructure and cognition. The fact emerged that the SMML model should lessen the reader's burden of understanding a document, and so reduce the costs of using the channel of communication that is the document.

We go further into theoretical validation to refine the expected cognitive qualities of SMML by showing that macrostructures respect the notion of Advanced Organizer (AO). Research on AO showed that the advantages of using this structure are function of prior knowledge, expertise and reading technique.

Finally, we present a practical example of a SMML document, a macrostructure catalog, and a specific structure called SMML-AR for publishing science articles. We also present a comparative analysis showing some of the advantages of using SMML.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES SIGLES ET ABRÉVIATIONS

ANSI	American National Standards Institute
AO	Advance Organizer
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organization
GlStr	Structure Globale
MaStr	MacroStructure
MiStr	MicroStructure
MML	Mathematic Markup Language
OHCO	Ordered Hierarchy of Content Object
PBR	Perspective-Based Reading
SGML	Standard Generalized Markup Language
SMML	Semantic Macrostructure Markup Language

SuStr	SuperStructure
XML	Extensible Markup Language

INTRODUCTION

Le présent travail s'inscrit dans le cadre du projet ContextEdit du laboratoire de recherche en génie logiciel de l'École Polytechnique de Montréal. Ce projet, démarré à l'hiver 1997, vise à développer, à l'aide des technologies d'édition structurée comme le SGML, un outil CASE capable d'assister l'ingénieur tout au long du processus de développement. Les objectifs de ce projet étant d'offrir un outil unique et de réduire sensiblement la duplication d'information due à l'utilisation de plusieurs outils ou d'outils mal adaptés. Cette duplication est dispendieuse en termes d'effort de réalisation et de maintenance et est sujette à l'erreur.

Le logiciel se définit comme une structure d'information complexe composée de concepts provenant à la fois du domaine de l'application et du domaine du logiciel. Par sa nature, celui-ci est une entité complexe et invisible, il n'existe pas de représentation conceptuelle complète capable de le contenir (Brook, 1987).

Selon Sommerville (1995), étant donné la nature intangible du logiciel, les organisations impliquées dans le développement de projets d'envergure utilisent des processus qui sont orientés sur les livrables. Chacune des activités du processus se solde par la production d'un ou de plusieurs documents. Typiquement, un projet important produit une centaine de documents de toutes sortes comme des documents de requis, de design, de tests, de code source, etc.. Le document constitue en fait le seul moyen tangible de représenter le logiciel ainsi que le processus logiciel.

Tout semble indiquer que le document technique est à la base du développement logiciel et constitue le principal canal de communication utilisé par les ingénieurs

logiciels. Il s'ensuit qu'une portion importante de leur temps est consacrée à la rédaction, à la lecture, à la vérification et à la validation de ces documents.

Fondamentalement, qu'est-ce qu'un document technique? Qu'en est-il de son rôle en tant que médium de communication? Est-il adapté ou optimisé pour transférer efficacement et à moindre effort l'information qu'il contient? Est-ce que les documents, dans leur format électronique actuel, permettent de représenter au mieux la valeur intellectuelle de leur contenu?

Parallèlement à ces considérations, DeRose et al. concluent dans un article[5] de fond sur la nature du document que les modèles électroniques existant étaient inadéquats. Ceci étant dû principalement au fait qu'ils limitent l'élaboration de traitements évolués comme la recherche de citations d'un texte ou la déduction d'un document à l'aide d'informations contenues dans d'autres documents.

Il propose la structure électronique OHCO[6] afin d'exploser les limites imposées par les autres formats. DeRose et al. soutiennent qu'un document peut posséder plusieurs structures valides qui sont des hiérarchies ordonnées d'objets de contenu. Par exemple, la Bible en possède au moins trois différentes, soit : une hiérarchie de références, une hiérarchie thématique et une hiérarchie de mises en page[7].

---

[5] Le périodique *The Journal of Computer Documentation* a consacré un numéro en 1997 à l'article "What is Text, Really", de DeRose et al. initialement paru en 1990. Un but visé étant notamment d'attirer l'attention des practiciens sur les articles intellectuellement importants, de permettre un débat sur leur contenu et d'expliquer les nouveaux développements sur le sujet.
[6] Ordered Hierarchy of Content Object.
[7] Voir la section 2.3.3 sur les multiples hiérarchies structurelles d'un texte pour plus de détails.

À la lumière de tout cela, quel pourrait être un modèle électronique de type OHCO permettant de mettre en relief la structure sémantique[8] d'un document, ceci avec l'objectif d'améliorer la lisibilité et l'assimilation de celui-ci? Un tel modèle se distinguerait des modèles déjà existants[9] axés sur la structure de présentation, puis qu'il permettrait de pointer des idées comme les explications, les définitions, contrairement à des éléments de mise en page comme les titres, les listes, etc.

L'objectif principal de ce projet de recherche consiste donc à trouver un moyen d'améliorer le canal de communication qu'offre le document entre les ingénieurs logiciels en définissant un format électronique adéquat. Plus concrètement, ce mémoire expose les théories qui sont à considérer pour les présentes recherches. Notamment, on approfondit la problématique qui vient d'être brièvement exposée (Chapitre 1). Par la suite, on explique la nature du document et on expose les types de format électronique proposés par DeRose et al. (Chapitre 2). La suite (Chapitre 3) est consacrée à l'explication de la théorie de Van Dijk sur les macrostructures. On présente et discute le modèle SMML[10] (Chapitre 4) à l'aide des théories présentées dans les premiers chapitres. On propose par la suite (Chapitre 5), une présentation graphique afin d'afficher un document SMML à l'écran. On fait une validation théorique du modèle SMML afin de vérifier s'il a le potentiel pour améliorer l'efficacité du canal de communication qu'est le document technique (Chapitre 6). Finalement, on présente une application pratique du SMML ainsi qu'une étude comparative avec un document conventionnel.

---

[8] Organisation et articulation structurelle des idées d'un discours écrit ou autre.
[9] On parle de modèle comme le HTML ou ceux implantés dans les éditeurs commerciaux comme Microsoft Word™ ou Corel Word Perfect™.
[10] Semantic Macrostructure Markup Language

CHAPITRE I -

DÉFINITION DU CADRE DE RECHERCHE

Dans ce chapitre, on approfondit individuellement chacun des divers aspects qui ont été présentés dans l'introduction. En premier lieu, on présente les considérations de Hilera et al. (1998) sur l'importance de la documentation dans le génie logiciel. En second lieu, on présente le modèle cognitif d'assimilation de Mayer (1985) qui fournit une base suffisante pour exposer les aspects cognitifs d'une communication entre deux individus. Dans un autre ordre d'idées, on introduit les travaux de Basili et al. (1996). Ceux-ci sont très intéressants étant donné que la problématique est similaire et compatible, en quelque sorte, aux présentes recherches. Finalement, on fait le lien avec tout ce qui est présenté en exposant la problématique de la recherche.

1.1 Le génie logiciel et la documentation

Le document technique a un rôle fondamental dans un processus de développement logiciel afin que le logiciel soit conforme aux attentes du client en termes de fonctionnalités, de coût et d'échéance. Il constitue le moyen de communication par lequel est prise en charge l'invisibilité du logiciel.

De ce fait, le document technique est utilisé tout au long du processus de développement et fait déjà l'objet de nombreuses normes (IEEE, 1997) afin d'en définir et d'en régulariser le contenu. La Figure 1.1 illustre la complexité et l'importance de la documentation dans un processus de génie logiciel.

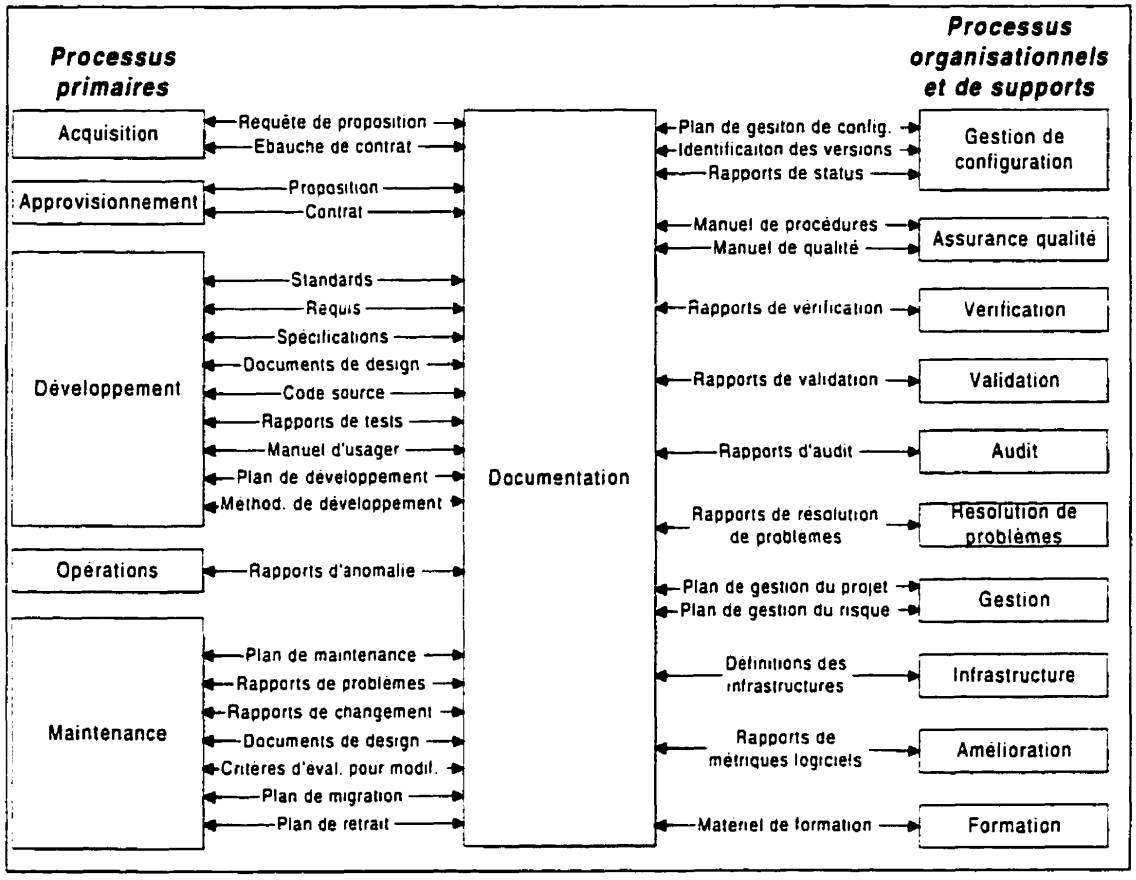


Figure 1.1 – Le processus et quelques documents impliqués dans le cycle de vie du logiciel selon ISO 12207 (Hilera et al., 1998) Traduction libre

La figure permet de voir les documents qui sont reliés au processus de développement et ceux qui sont reliés aux processus organisationnels et de supports.

Selon Hilera et al. (1998), la documentation de logiciels est à ce point importante qu'elle devrait faire l'objet de son propre processus d'ingénierie[11] au même titre que le

[11] L'ingénierie de la documentation.

logiciel et le génie logiciel. Par ailleurs, les normes ISO et ANSI suggèrent implicitement de traiter la documentation comme telle.

1.2 Le document et la communication

En premier lieu, on présente la théorie cognitive sur la compréhension et l'assimilation qui constitue les bases théoriques sur lesquelles reposent les travaux de Mayer (1985)[12]. Cette théorie constitue une base suffisante pour expliquer ensuite quelles sont les bases cognitives d'une communication entre individus qui utilisent le document comme moyen de communication.

1.2.1 Modèle cognitif de compréhension et d'assimilation

La Figure 1.2 schématise un processus de lecture ou, dans un sens plus général, le processus d'assimilation de l'information.

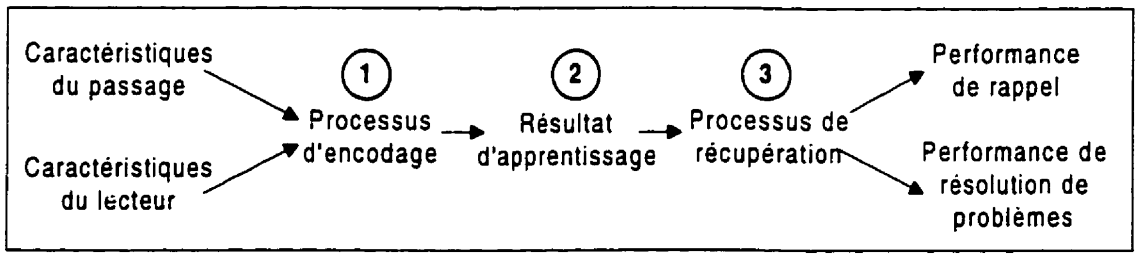


Figure 1.2 – Les processus cognitifs d'assimilation (Mayer, 1985)

[12] Dans son article, Mayer rend compte des résultats de ses recherches sur l'amélioration des performances au niveau de la résolution de problèmes (Structural Analysis of Science Prose : Can We Increase Problem-Solving Performance?).

On remarque que le modèle est composé de trois éléments de base (Processus d'encodage, Résultat d'apprentissage et Processus de récupération) qui permettent de modéliser le fonctionnement cognitif. Selon Mayer, le système comporte deux variables indépendantes, soit les caractéristiques de l'extrait textuel ainsi que les caractéristiques du lecteur. Ces variables influencent le système dans son ensemble et par conséquent les performances au niveau du rappel et de la résolution de problèmes (celle-ci constituent les variables dépendantes du système).

Processus d'encodage (1)

Il s'agit du processus par lequel le lecteur sélectionne l'information d'un extrait textuel et combine cette information avec l'information déjà acquise. La sélection et la combinaison de l'information peuvent s'effectuer de plusieurs façons différentes. Mayer fait la distinction entre deux types de processus d'encodage. Premièrement, le processus par addition : le sujet sélectionne des informations factuelles qui ont un lien arbitraire entre elles. Deuxièmement, le processus par assimilation : le sujet sélectionne des informations explicatives. Une explication exprime une relation fonctionnelle (ou une règle) entre deux ou plusieurs variables qui est détaillée en fonction des sous-mécanismes qui sont mis en jeu.

Résultat d'apprentissage (2)

Le résultat d'apprentissage représente la structure de l'information qui est acquise. Mayer le représente à l'aide de la notion d'idée, qui constitue l'élément atomique de la structure. Dans le cas d'un processus d'encodage par addition, on obtient une structure d'information contenant des idées qui n'ont pas de lien entre elles. Pour un processus par assimilation, on obtient un réseau d'idées reliées entre elles par des relations causales.

Processus de récupération (3)

Il s'agit du processus par lequel le sujet cherche la structure d'information acquise afin de formuler une réponse à une question. Dans le cadre de ce mémoire, le processus est à la base de l'activité d'édition des documents tel qu'effectué par un auteur.

Mayer s'intéresse aux performances, du rappel et de la résolution de problèmes, que l'on peut mesurer à la sortie du système. Celles-ci sont dépendantes de toutes les étapes du processus cognitif. Cependant, il s'intéresse principalement aux aspects du processus d'encodage ainsi qu'aux variables indépendantes du système (caractéristiques de l'extrait et caractéristiques du lecteur) qui influencent ces performances.

1.2.2 Définition d'une communication entre individus

Fondamentalement, un document est le médium d'un canal de communication par lequel une personne (l'auteur) transmet une information à d'autres personnes (lecteurs). Le processus de communication peut être expliqué cognitivement par la Figure 1.3.

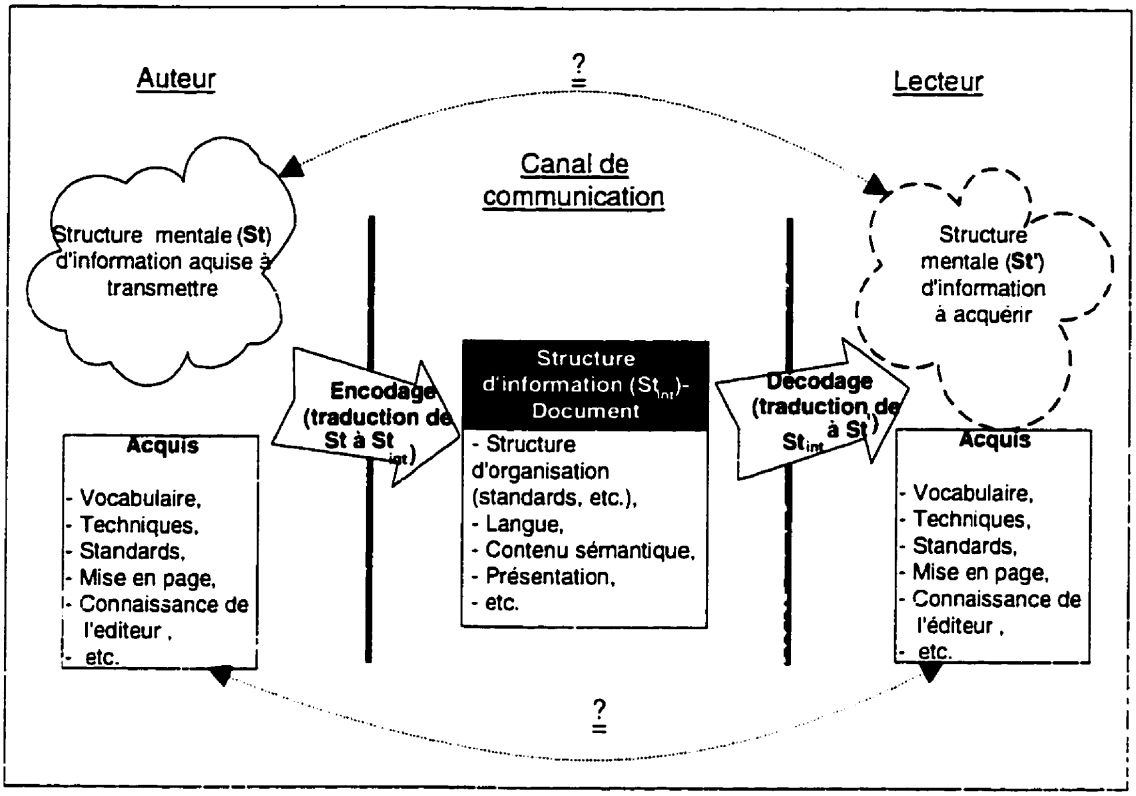


Figure 1.3 – Le processus de communication écrit

Le but de l'auteur est de transmettre une structure d'information mentale (St), le *Learning Outcome* au sens de Mayer (1985), à son auditoire. Plusieurs moyens peuvent être utilisés afin de s'affranchir de cette tâche, par exemple : la parole, les gestes, la communication écrite, etc.

L'utilisation du document comme canal de communication comporte certains avantages. Notamment, il s'agit d'un médium permanent qui permet de conserver l'information indéfiniment. De plus, il ne nécessite pas une synchronisation temporelle des intervenants afin d'établir et de conduire la communication. L'auteur peut ainsi rédiger le document seul et le transmettre au destinataire par la suite.

Le détail du processus de communication entre deux individus qui utilisent un document comme médium de communication, se retrouve dans les étapes suivantes :

- Traduction et encodage de la structure (St) mentale de l'auteur afin de générer la structure (St_{Int}) du document. Ce processus est influencé par l'acquis de l'auteur, notamment, sa connaissance des standards, des règles de mise en page, de l'outil d'édition utilisé, etc.

- Décodage et traduction de la structure (St_{Int}) du document selon les règles (acquis du lecteur) afin de constituer la structure (St') mentale du lecteur.

Qu'en est–il de l'efficacité de ce canal de communication? On peut considérer deux aspects importants. Premièrement, la fidélité qui se définit comme le degré de conformité, en termes de contenu d'idées et de liens causaux, entre la structure (St) mentale initiale de l'auteur et la structure (St') finale du lecteur. Le canal de communication tel que modélisé comporte une sensibilité au bruit (i.e.: erreur d'interprétation et autres). La fidélité des processus cognitifs d'encodage, de traduction et de décodage dépend de plusieurs facteurs plus ou moins contrôlables comme l'attention, la motivation, l'expérience et autres, qui peuvent conduire à une erreur d'interprétation (consciente ou non). De plus, les acquis du lecteur comparativement à ceux de l'auteur peuvent être incohérents ou incomplets, ce qui peut limiter la communication ou introduire des erreurs.

Deuxièmement, l'effort ou l'énergie nécessaire qui est relié à l'utilisation du canal de communication. Par exemple, l'utilisation d'une langue seconde lors de l'édition d'un document peut demander beaucoup plus d'effort que l'utilisation de la langue natale.

1.2.3 Les techniques de lecture basée-perspective de Basili et al.

On introduit les travaux de Basili et al. (1996) étant donné que les objectifs de leurs recherches sont compatibles avec le présent projet. En effet, ils s'intéressent à la documentation dans les projets de génie logiciel en abordant la problématique des techniques de lecture. Leurs travaux portent sur l'élaboration de techniques de lecture basée-perspective.

Ils considèrent que les techniques de lecture employées par les intervenants dans un projet ont un impact important sur l'atteinte d'une haute qualité logiciel. Leurs travaux touchent à l'aspect de la caractérisation du lecteur (*Learner Characteristics*) telles que définies dans le modèle de Mayer (1985).

Selon Basili et al., la lecture constitue une activité clé afin de vérifier et de valider les divers documents (requis, code source, plan de tests, design, ...) associés à un projet logiciel. Peu de recherches ont été faites à ce jour dans ce domaine. De plus, aucune recherche n'a été faite sur les techniques de lecture qui seraient orientées spécifiquement sur un type de documents, par exemple : l'élaboration d'une technique de lecture spécifique pour lire un document de design ou de requis. L'auteur propose une technique de lecture basée-perspective ou PBR[13] qui permet de mettre l'accent sur le point de vue (développeur, testeur, ...) ou sur les besoins du lecteur.

Les diverses techniques sont décrites à l'aide de scénarios opérationnels. Fondamentalement, le lecteur doit, quelque soit la technique, concevoir un artefact

---

[13] Perspective-Based Reading.

durant sa lecture et répondre à des questions prédéfinies concernant le document qui permettent d'en déceler les défauts.

Par exemple, une personne chargée des tests (vue du testeur) doit lire un document de requis en utilisant une technique PBR. Par conséquent, selon le scénario opérationnel qui définit la technique, la personne doit concevoir les cas de tests pour chaque requis (artefact). De plus, elle doit répondre à des questions permettant de réviser certains aspects de chaque requis (« Avez-vous toutes les informations nécessaires pour faire le test ? », etc.).

L'hypothèse qui est à la base des travaux de Basili et al. est que l'union des perspectives permet d'obtenir une couverture plus complète d'un document. Chaque lecteur est responsable d'un aspect ou d'une vue bien définie, ce qui devrait conduire à une analyse plus en profondeur du document et donc trouver un plus grand nombre d'erreurs.

1.3 Définition de la problématique de recherche

Le document a un impact important sur l'atteinte d'une haute qualité logiciel. En effet, il permet de prendre en charge l'aspect invisible du logiciel. De plus, il est l'artefact de base produit par un nombre important de tâches d'un processus logiciel. Le répertoire fait par Hilera et al. (Figure 1.1) permet de considérer l'ampleur de son utilisation.

Étant donné l'importance des tâches, en termes de temps et de ressources, reliées directement ou indirectement à la communication écrite, il serait intéressant d'améliorer le canal en terme de coût ou d'effort et de fiabilité (qualité de l'information transmise).

Selon le modèle cognitif fourni par Mayer (1985), si l'on considère le lecteur il y a deux variables qui influencent le processus, soit les caractéristiques du lecteur et les caractéristiques du passage ou de l'extrait textuel qui est traité.

Dans leurs recherches, Basili et al. (1996) se sont intéressés aux caractéristiques du lecteur en définissant des stratégies de lecture qui améliorent la qualité de l'information assimilée. Dans le cadre de ce projet de maîtrise, on se penche sur l'autre variable dont dépend le système, soit les caractéristiques de l'extrait textuel ou d'une façon plus large, les caractéristiques du document qui influencent la qualité de l'information assimilée. On suppose que si la structure électronique ou physique du document se rapproche de la structure mentale d'un individu, l'effort relié à l'encodage et/ou au décodage est réduit. Ceci peut avoir une influence bénéfique sur le coût et la fiabilité de ce type de communication.

Un aspect qui motive l'étude est le constat des technologies (éditeurs, structures électroniques de textes, etc.) disponibles et utilisées dans le cadre des projets logiciels. On peut se poser la question de savoir si elles sont adaptées et optimisées pour le transfert d'information technique.

Actuellement, l'édition et l'organisation structurelle des documents reposent sur des bases d'esthétique graphique importantes. Par exemple, les éditeurs de textes conventionnels comme Word™ et les formats électroniques comme le HTML, permettent d'identifier surtout les éléments[14] du texte qui ont des fonctions graphiques ou de mise en page, par exemple à l'aide des styles: *Titre1*, *Titre2* . On peut s'interroger sur la valeur intellectuelle de ces structures et s'il n'est pas préférable de pointer des structures exprimant des idées comme des définitions ou des explications.

Des travaux importants ont été faits depuis les années 80 dans le domaine de l'analyse de discours. L'analyse de discours a pour but, notamment de comprendre l'organisation structurelle des communications textuelles, orales ou autres. Il serait intéressant de voir si certaines théories peuvent être adaptées afin de constituer un modèle électronique pour représenter les documents et ainsi améliorer le document technique en tant que canal de communication.

1.4 Résumé du chapitre

Dans ce chapitre, on a défini la problématique du projet. Fondamentalement, celui-ci a pour objectif de définir une structure de document qui soit adaptée au génie logiciel et en mesure d'améliorer l'efficacité du canal de communication.

On a préalablement présenté la place du document dans le cadre du génie logiciel ainsi que les processus cognitifs sous-jacents à l'utilisation du document comme moyen de communication.

Le chapitre suivant présente l'essence du document technique, à savoir comment on peut le définir. De plus, on présente l'état de la technologie concernant la représentation électronique des documents.

---

[14] Pour ce faire, il utilise la notion de style.

CHAPITRE II –

L'ESSENCE DU DOCUMENT TECHNIQUE

Les objectifs de ce chapitre sont premièrement, d'expliquer la nature du document à l'aide du standard SGML. Deuxièmement, on présente le répertoire de modèles électroniques existants répertorié par DeRose et al.(1990). On explique les raisons pour lesquelles ils sont inadéquats ainsi que leurs limites au niveau de l'élaboration de traitements évolués. Finalement, on présente le modèle OHCO, un format électronique souhaitable afin d'exploser les possibilités de traitements de l'ordinateur.

2.1 Le standard SGML

Le SGML, qui signifie « *Standard Generalized Markup Language* » est un standard international[15] publié en 1988 selon les travaux de Dr Goldfarb d'IBM. Son objectif est de permettre la définition d'une représentation électronique pour qu'un document soit portable, c'est-à-dire, indépendant de la plate-forme et des logiciels (Goldfarb, 1990).

Le SGML constitue une solution aux problèmes posés par le nombre croissant de formats électroniques de documents, comme le PostScript, TeX, MSWord, etc. (Burnard, 1995). De plus, ceux-ci sont souvent la propriété d'une compagnie et/ou sont définis dans un but très spécifique et sont par conséquent peu adaptables à d'autres fins.

---

[15] Fait l'objet de la norme ISO 8879.

Le SGML est un métalangage qui fournit une base commune unique afin de représenter électroniquement divers types de documents. Il permet de décrire les propriétés formelles et les interrelations entre les composantes (éléments) d'un document. Il a été utilisé notamment, pour définir le HTML, XML, MML, des formats de documents connus grâce au réseau Internet.

On peut définir n'importe quel type de traitements sur un fichier SGML et on peut le transférer d'une application supportant le SGML à une autre sans problème de compatibilité.

2.1.1 Les dimensions logiques d'un document

Selon le standard SGML (Goldfarb, 1990), un document est une construction logique qui contient des éléments comme par exemple un chapitre, un paragraphe, une image, etc. Un document est considéré comme une structure arborescente dont les nœuds représentent des éléments structurels et les feuilles constituent le contenu du document. Un document peut être divisé selon trois niveaux d'information, soit:

- La structure générique: C'est l'organisation des éléments d'une classe de document. Par exemple, tous les courriers électroniques respectent une structure générique dans laquelle on retrouve les éléments tels l'entête, le corps du message, le destinataire, l'expéditeur, le sujet, etc. Il s'agit de la classe du document. La structure générique d'une classe de document est spécifiée à l'aide d'un DTD[16].

---

[16] Document Type Definition.

- Le contenu: C'est l'information utile qui est véhiculée dans le document, celle-ci est organisée et respecte la structure générique de sa classe de document.

- Le style et la mise en page : C'est l'organisation spatiale et la représentation graphique des éléments du document à l'écran ou sur papier. Il est à noter qu'il est possible d'avoir plus d'une présentation graphique pour une structure générique donnée.

Le SGML permet de séparer ces niveaux d'information, mais il ne s'occupe pas de la mise en page d'un document. Il permet d'expliciter les relations entre la structure et le contenu à l'aide du DTD. Il existe le DSSSL[17] et le FOSI[18] qui peuvent être utilisés afin de préciser la mise en page d'un document SGML.

La Figure 2.1 permet d'illustrer les trois niveaux d'information d'un document.

[17] Document Style Semantics and Specification Language.
[18] Formatting Output Specification Instance.

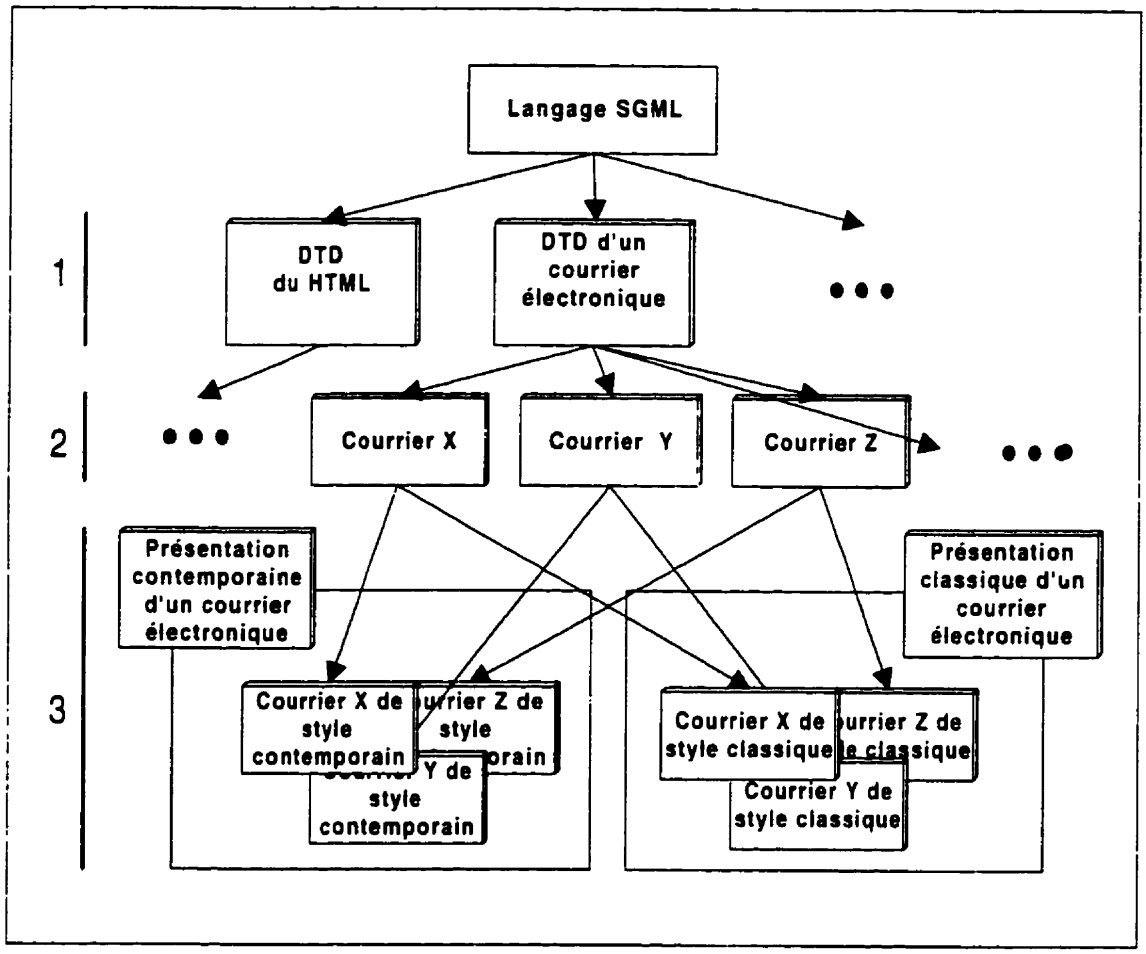


Figure 2.1 – Les trois niveaux d'information d'un document selon le SGML

On note dans cette figure que le SGML permet de définir plusieurs structures génériques de documents (DTD). De plus, il est possible de créer une multitude de documents (fichiers SGML) qui respectent la structure générique de leur classe (DTD). Par exemple, dans la figure, on remarque qu'il y a trois courriers de la classe *courrier électronique* qui ont été rédigés.

D'un autre côté, on peut définir des présentations différentes qui sont basées sur une classe de documents. Dans la figure, on illustre deux schémas de présentation (i.e. : une présentation *contemporaine* et une *classique*) pour la classe courrier électronique.

Finalement, avec ces présentations, il est possible de représenter graphiquement, de différentes façons, tous les fichiers SGML de cette classe, que ce soit à l'écran ou sur papier.

2.1.2 La structure générique du document : le DTD

Chaque document SGML doit être conforme à son DTD (*Document Type Definition*), car celui-ci contient la définition formelle de la structure du document. Le DTD permet d'identifier les éléments qui composent le document ainsi que les liens entre ceux-ci. Le SGML permet seulement de spécifier qu'un document doit avoir un certain ensemble d'éléments précis. Il ne permet pas de statuer sur la pertinence sémantique de l'existence ou de l'utilisation de tels éléments.

Le DTD permet aussi d'expliciter le type de chaque élément, par exemple : texte, image, élément composé, élément terminal, etc. ainsi que ses propriétés que l'on nomme attributs. La Figure 2.2 montre un DTD qui définit une classe de documents courrier électronique.

```
<!--Courrier du club de golf DTD-->
<!--1> <!ELEMENT courrier - - (en_tete,corps)>
<!--2> <!ELEMENT en_tete  - O ((de & a) & sb?)>
<!--3> <!ELEMENT corps    - O (p*)>
<!--4> <!ELEMENT de       - O (#PCDATA)>
<!--5> <!ELEMENT a        - O (#PCDATA)>
<!--6> <!ELEMENT sujet    - O (#PCDATA)>
<!--7> <!ELEMENT para     - O ((#PCDATA|citation)*)>
<!--8> <!ELEMENT citation - - (#PCDATA)>
```

Figure 2.2 – DTD d'un courrier électronique

Le DTD contient des déclarations qui ont la forme <!Declaration>. La figure permet de voir des déclarations de type commentaires (<!--commentaires>), ils sont utilisés afin de numéroter les lignes et pour indiquer une brève description du DTD (ligne 0). Les déclarations avec le mot clé *ELEMENT* permettent de définir les éléments du DTD.

On remarque que l'élément *courrier* (ligne 1) est un élément composé des éléments *en_tete* et *corps*. Il serait possible de définir des attributs à chaque élément en ajoutant une ligne du type :

```
<!ATTLIST courrier ceci_est_un_attribut_de_type_texte CDATA>.
```

Celle-ci déclare un attribut de type texte attaché à l'élément *courrier*.

La définition des éléments composés est à la base de la structure hiérarchique d'un document. Le DTD permet, via la définition des éléments composés, d'expliciter les liens entre les divers éléments du document. Selon les besoins, on peut définir une structure qui soit très ou peu restrictive quant au contenu en éléments admis à un endroit du document. Dans l'exemple, un paragraphe (élément *para* à la ligne 7) ne peut pas contenir d'image, seulement du texte et des citations. Le DTD du HTML, quant à lui, est faiblement hiérarchisé et très souple. À un endroit donné d'un document HTML, il y a souvent plusieurs types d'éléments qui peuvent être valides ce qui a pour effet de rendre moins évidente la structure générique du document HTML.

Le DTD permet de préciser les étiquettes de début ou de fin d'un élément qui sont optionnelles à l'aide des symboles – (requis) et *o* (optionnel). Ce mécanisme permet d'omettre les étiquettes lorsqu'il n'y a pas d'ambiguïté possible quant à l'identification et la délimitation d'un élément. Dans l'exemple de la Figure 2.2, la rencontre du début de l'élément *sujet* marque sans ambiguïté la fin de l'élément *de*. Il s'agit d'un

mécanisme d'optimisation qui peut réduire substantiellement la taille d'un fichier volumineux.

Les symboles * (0 et plus), & (Et), | (Ou), + (Un et plus), permettent de préciser l'organisation et l'occurrence des éléments constituant un élément composé.

2.1.3 Le contenu du document : le fichier SGML

Le fichier SGML contient l'information utile. Celle-ci est exprimée sous forme d'un texte balisé. Les étiquettes utilisées pour baliser le texte sont conformes au DTD et l'ordre de présentation respecte la structure hiérarchique du document.

La Figure 2.3 montre un fichier SGML de la classe *courrier électronique*.

```
<!DOCTYPE mail SYSTEM "c:\sgml\dtd\courrier.dtd">
<courrier>
<en_tete>
<a>Secretariat du club
<de>Membre senior
<sujet>L'apport du SGML?
<corps>
<para>Cher Secretariat
<para>P. G. Wodehouse a dedie The Heart of a Goof
<citation>A ma fille Leonora dont la sympathie et
l'encouragement ont permis de realiser ce livre en un
temps record.</citation> Est-ce que vous croyez que le
SGML y est pour quelque chose?
<para>Mes salutations
<para>Membre senior
</courrier>
```

Figure 2.3 – Fichier SGML d'un courrier électronique

On remarque que le fichier ressemble en plusieurs points à un fichier HTML, notamment sur le format d'étiquettes utilisées afin d'identifier les éléments :

<TypeElement>TEXTE</TypeElement>. La première ligne précise le nom du fichier contenant le DTD à utiliser afin de valider et traiter le document. On remarque que certaines étiquettes de fin sont manquantes : , </de>,</sujet>. Celles-ci sont déclarées optionnelles dans le DTD.

2.1.4 La présentation du document SGML

En définissant une présentation à l'aide d'un autre standard (DSSSL ou FOSI) qui précise la mise en page des documents SGML, on pourrait obtenir comme résultat la disposition montrée à la Figure 2.4.

Courrier electronique

A: Secretariat du club
De: Membre senior
Sujet: L'apport du SGML?

Cher secreariat

P. G. Wodehouse a dedie The Heart of a Goof «A ma fille Leonora dont la sympathie et l'encouragement ont permis de realiser ce livre en un temps record.». Est-ce que vous croyez que le SGML y est pour quelque chose?

Mes salutations

Membre senior

Figure 2.4 – Un courrier électronique

Cette représentation semble sobre à première vue, néanmoins, elle comporte plusieurs caractéristiques importantes. Notamment, des chaînes de caractères ont été ajoutées devant les éléments *a*, *de* et *sujet* afin d'identifier le destinataire, l'expéditeur et le sujet du courrier. Ces chaînes ne font pas partie de l'information utile du document (fichier

SGML), ce sont des artifices de présentation. Il serait possible d'afficher le document avec une entête identifiée en anglais en modifiant le fichier de présentation sans toutefois changer le fichier SGML. Dans cette présentation, on note aussi que la citation (élément *citation*) est soulignée.

2.1.5 Le SGML et les types d'interactions

Dans leur livre[19], Maler et al. (1996) exposent clairement les interactions possibles avec un document SGML. Les documents SGML sont normalement créés pour communiquer de l'information ou pour garder trace de celle-ci (Maler, 1996). La Figure 2.5 illustre les interactions possibles avec des documents SGML.

[19] Developing SGML DTDs, from the text, to model, to markup.

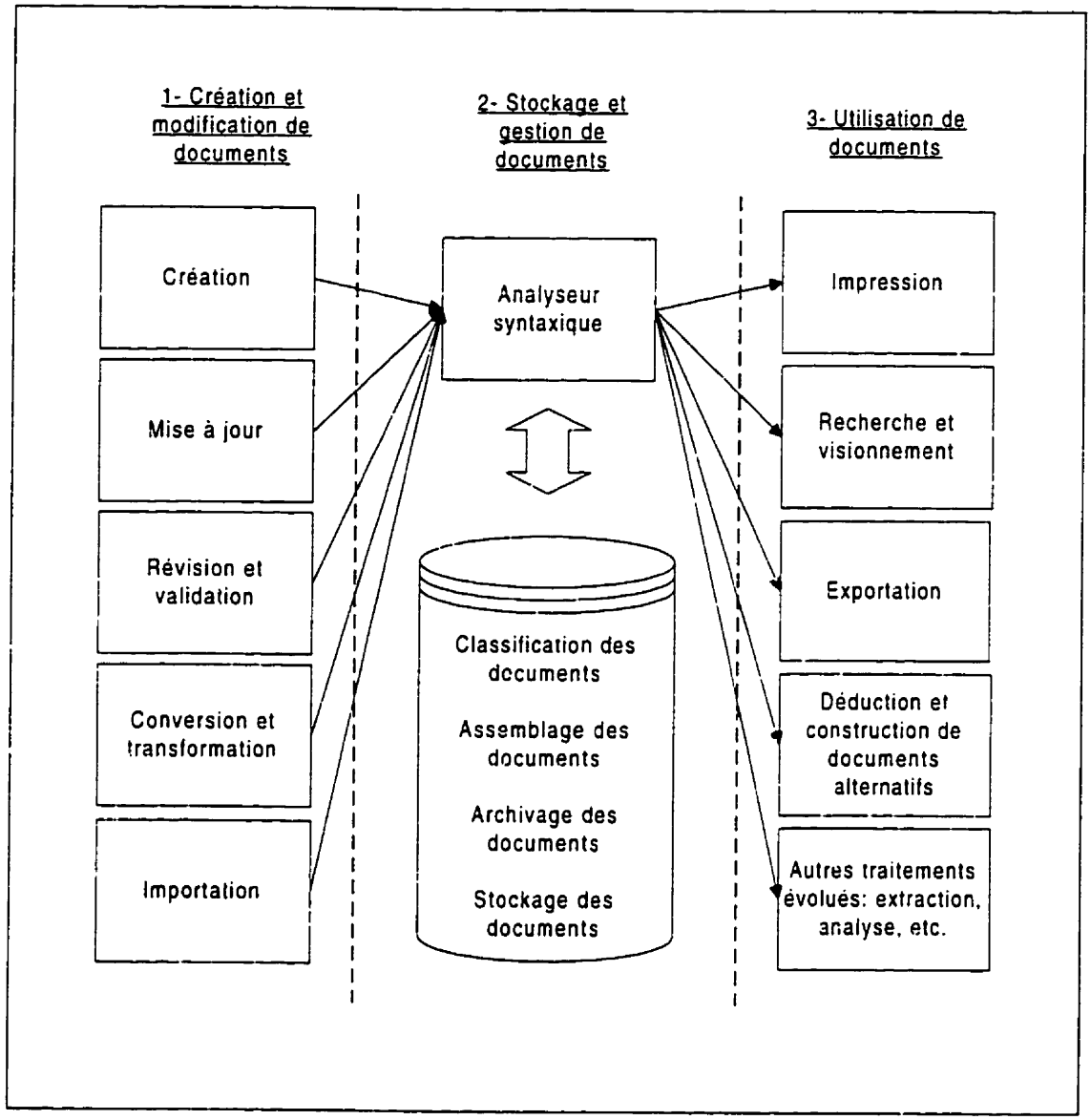


Figure 2.5 – Les types d'interaction avec le SGML

Gestion, stockage et archive (2)

Ce type d'interaction implique les activités de vérification, d'identification, de classification et d'indexation dans le but de pouvoir récupérer un document SGML parmi un ensemble de documents. L'analyseur syntaxique vérifie chaque document qui entre ou sort afin de s'assurer qu'il est conforme à son DTD[20]. Celui-ci constitue l'outil de base qui est intégré à tout logiciel qui traite des fichiers SGML, comme les éditeurs, les traducteurs, etc.

Création et modification (1)

Ce type d'interaction touche toutes les activités qui ont un lien avec l'entrée d'informations dans le système, comme la création, la modification, la révision, la validation, etc. Pour ce qui est des activités de conversion et de transformation, elles englobent la tâche de traduction[21] d'un document non-SGML à un document SGML. Il peut s'agir aussi de la traduction[22] d'un fichier SGML qui respecte un DTD à un autre fichier SGML qui respecte un autre DTD'.

Utilisation (3)

Le type utilisation touche aux activités d'impression, de recherche, de visualisation, d'exportation de documents dans un format non-SGML[23]. Une activité très importante est la recherche et l'extraction de l'information afin de créer un document alternatif ou

---

[20] Une base de documentation des documents de plusieurs classes (DTD) différentes.
[21] Ce processus est appelé "Up-Translate".
[22] Ce processus est appelé "Cross-Translate".
[23] Ce processus est appelé "Down-Translate".

un sous-document. Par exemple, si on a l'ensemble des documents de design d'un projet logiciel dans la base de données et que leur structure respecte un DTD qui est approprié, il serait possible de générer une partie du code du projet directement à partir de ces documents.

2.1.6 Les coûts et les bénéfices à l'utilisation du SGML

Les avantages de l'utilisation du SGML sont plus évidents lorsque l'on considère un ensemble volumineux de documents. Par exemple, on peut alors imaginer un nombre imposant de documents si on considère les documents produits durant les divers projets de développement d'une importante firme informatique.

L'utilisation du SGML permet de s'assurer la conformité de la représentation des documents. On peut ainsi changer rapidement le style ou la mise en page d'un ou de plusieurs éléments en modifiant seulement le fichier de présentation. L'utilisation d'éditeurs conventionnels, comme MSWord^{TM}, pour faire l'édition pourrait se solder par l'obligation de modifier manuellement chacun des documents, ce qui serait très dispendieux et sujet à l'erreur.

De plus, le SGML permet d'implanter le concept de vue. On peut adapter ou restreindre la présentation de l'information en fonction des intervenants (par exemple, les concepteurs, les chargés de maintenance, les programmeurs, les gestionnaires, le client, etc.). On peut choisir de montrer ou non certains éléments ou les organiser autrement afin de mettre en évidence différents aspects.

On peut élaborer des logiciels d'édition qui intègrent automatiquement une foule d'informations complémentaires aux documents lors de l'édition. Par exemple, on peut saisir la date ainsi que l'identité d'une personne qui vient de créer un élément de design dans un document. Ceci peut être intéressant pour des fins de gestion ou de traçabilité de l'information.

Cependant, il y a certaines difficultés reliées à l'utilisation de la technologie SGML, notamment, la définition des DTD pour représenter les documents. Il a été précisé plus haut que le SGML permet d'identifier les éléments d'un document. Il ne permet pas de savoir s'ils sont adéquats (sémantiquement ou autre). Alors, que doit-on représenter? Avec quelle granularité doit-on présenter un document?

De plus, un problème complexe concerne la traduction des documents SGML d'une structure (DTD) à une autre, par exemple, dans le cas où l'on disposerait de deux DTD différents pour représenter un courrier électronique. Il faudrait établir la correspondance entre chacun des éléments des DTD, ce qui n'est souvent pas simple voire impossible, dans certains cas. Si l'on considère le DTD de la Figure 2.2 et un autre DTD de courrier électronique, on a un problème de traduction dans le cas où il n'y a pas d'équivalent pour l'élément *citation* dans le second DTD.

2.2 Les modèles alternatifs de textes électroniques

Il existe plusieurs structures couramment utilisées, autres que le SGML, dans le monde informatique afin de représenter les documents sous format électronique. On présente dans les sections qui suivent les divers modèles tels que répertoriés par DeRose et al. (1990) ainsi que les limites de chacun en ce qui à trait aux traitements possibles.

2.2.1 Le texte comme une image

La Figure 2.6 illustre un extrait de texte représenté à l'aide d'une image. Cette représentation est surtout utilisée pour la conservation électronique des documents manuscrits ainsi que pour les télécopies. L'oeil est en mesure de repérer les éléments du message (lettres, mots, phrases, ...) et ainsi de décoder le message. Cependant, les traitements automatiques sur cette représentation sont très limités. Par exemple, la recherche de mots est impossible, étant donné qu'on ne peut pas identifier les lettres composant l'image.

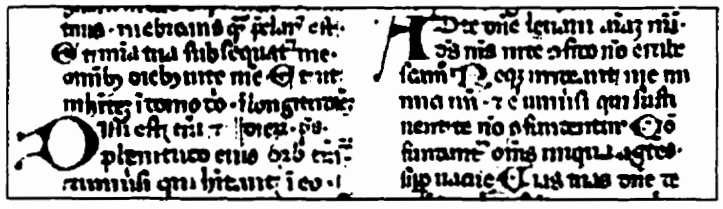


Figure 2.6 – Le texte comme une image

2.2.2 Le texte comme un flot de caractères

Le texte est représenté par des caractères et la structure du texte est explicitée, grâce aux espaces entre les mots, à la ponctuation et aux retours de chariot. Les seuls éléments que l'on peut aisément identifier sont les mots, les phrases et les paragraphes. Il est possible d'effectuer les opérations d'édition usuelles, par exemple : copier, coller, rechercher un mot, vérifier l'orthographe, etc. Cependant, il n'est pas possible d'effectuer des traitements plus complexes, comme obtenir automatiquement la liste de certains éléments du texte (liste des citations, table des matières, etc.). Cette structure est couramment utilisée pour les fichiers de type texte ou les courriers électroniques.

2.2.3 Le texte comme un flot de caractères et d'instructions de mise en page

Le format flot de caractères et instructions de mise en page est à la base des éditeurs de texte évolués. On ajoute à la structure précédente (i.e. : texte comme un flot de caractères) des codes de mise en page (polices, alignement du texte, etc.). Les mêmes types de traitements sont possibles qu'avec la structure précédente. On peut noter de légères améliorations, par exemple : il est possible d'identifier certains éléments (titre, citation, etc.) en leur attribuant une représentation graphique particulière. Cependant, il faut que cette représentation soit unique, sans quoi, il peut être difficile de distinguer la nature des éléments. De plus, l'auteur doit retenir les instructions de mise en page nécessaires afin de les représenter correctement. L'éditeur WordPad™, qui est livré avec le système Window95™, ainsi que le format de fichier RTF[24] qu'il implante, est un bon exemple.

La Figure 2.7 montre le fichier RTF qui permet de générer le courrier électronique tel que mis en forme précédemment[25].

---

[24] « Rich Text Format » qui se traduit en français par « Texte mis en forme ».
[25] Voir la Figure 2.4 – Un courrier électronique.

```
{\rtf1\ansi\ansicpg1252\uc1 \pard\plain \s15\qc\widctlpar\outlinelevel0\adjustright
\b\fs28\lang3084\cgrid {E-mail
\par }\pard\plain \qc\widctlpar\adjustright \fs20\lang3084\cgrid {\b\fs28
\par }\pard \widctlpar\adjustright {\b To:}{ \tab Secretariat du club
\par }\pard \widctlpar\outlinelevel0\adjustright {\b From:}{ \tab Membre senior
\par }\pard \widctlpar\adjustright {\b Subject:\tab }{L'apport du SGML?
\par }\pard \qc\widctlpar\adjustright {\f2 ----------
\par }\pard \widctlpar\adjustright {
\par Cher secretariat
\par
\par P. G. Wodehouse a dedie The Heart of a Goof \ldblquote }{\ul A ma fille Leonora dont la
sympathie et l'encouragement ont permis de realiser ce livre en un temps record.}{\rdblquote Est-ce
que vous croyez que le SGML y est pour quelque chose?

\par
\par }\pard \widctlpar\outlinelevel0\adjustright {Mes salutations
\par }\pard \widctlpar\adjustright {
\par }\pard \widctlpar\outlinelevel0\adjustright {Membre senior
\par }\pard \widctlpar\adjustright {
\par }}
```

Figure 2.7 – Le fichier RTF d'un courrier électronique

On y remarque les codes de mise en page suivants :

- *\b* : indique à l'éditeur de mettre en gras ce qui est entre accolades (i.e. A :, De :, Sujet :);

- *\ul* : indique à l'éditeur de souligner ce qui est entre accolades (i.e. la citation).

2.2.4 Le texte comme une structure de mise en page

La structure du texte respecte la façon usuelle de représenter un document. Elle se constitue d'éléments tels : livre, pages, en-tête, zone de texte principale, pieds de pages, et autres, qui sont organisés hiérarchiquement. L'exemple le plus représentatif de ce

modèle de document est le PostScript™. Cette structure de texte a les mêmes limites que les modèles précédents. Il ne serait pas possible de rechercher des structures plus évoluées comme des citations ou des équations mathématiques du texte. La Figure 2.8 montre un exemple de ce type de structure.

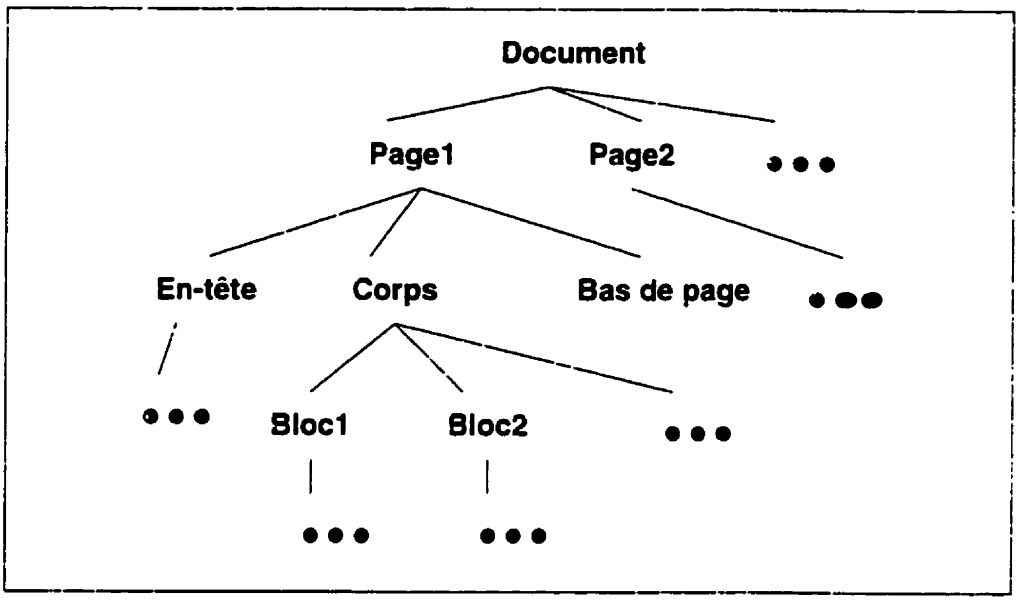


Figure 2.8 – Le texte comme une structure de mise en page

On remarque dans la figure que la division du document est fonction de sa représentation finale sur papier. Les pages contiennent une en-tête, un corps et un bas de page.

2.2.5 Le texte comme un flot d'objets de contenu

Le texte est composé d'éléments qui sont identifiés selon leurs types (date, citation, titre de chapitre, etc.). Ce modèle est implanté par les éditeurs de textes comme Microsoft Word™ ou Corel Word Perfect™. Ceux-ci utilisent la notion de « styles » afin d'identifier les éléments. Cependant, ils ne sont pas explicitement organisés

hiérarchiquement. En effet, ce modèle ne permet pas de définir des éléments de contenu composés (type d'élément qui contient d'autres éléments) qui sont la base de l'organisation hiérarchique. La hiérarchie du texte n'est visible qu'indirectement, via certains éléments[26] du texte. Par exemple, les types d'éléments *Chapitre*, *Premier Titre* et *Second Titre*, permettent de retrouver partiellement la hiérarchie.

Bien que ce modèle permette certains traitements évolués, comme la création d'une table des matières ou d'un index, il reste limité. Cette limite est d'autant plus marquée lorsque l'on travaille avec des documents volumineux qui ont une structure complexe et restrictive.

2.3 Le modèle de texte OHCO (Ordered Hierarchy of Content Object)

DeRose et al. (1990) proposent un modèle hiérarchique d'objets de contenu (OHCO), afin de repousser les limites au niveau du traitement automatique qui sont imposées par une structure inadéquate du texte. Ce modèle peut adéquatement être défini et appliqué à l'aide des technologies d'édition structurée, telle le SGML.

2.3.1 L'objet de contenu

Il y a une différence entre la forme et le contenu d'un texte. La partie essentielle de tout document est appelée « objets de contenu », il peut s'agir de paragraphes, de citations, etc. Chaque objet de contenu a certaines caractéristiques ou attributs comme un type et une façon particulière d'être représentée à l'écran. Selon DeRose et al., ce sont les

---

[26] On nomme ces éléments « Hierarchical Markup ».

éléments ainsi que leur contenu qui constituent l'essence du document. La présentation de ceux-ci est secondaire.

2.3.2 Le modèle OHCO

Les objets de contenu sont regroupés ensemble pour former une hiérarchie. Les plus gros comme les paragraphes contiennent les plus petits comme les citations. Selon DeRose et al., il existe suffisamment de preuves pour conclure sur le réalisme du modèle, notamment :

- le modèle reflète la structure linguistique du discours : les textes sont des objets linguistiques donc il s'agit d'un schème approprié;
- plusieurs textes publiés démontrent déjà implicitement une partie de cette structure, par exemple, via les tables de matières;
- il existe des manuels de styles qui décrivent les règles de mise en page en fonction des objets de contenu, par exemple, la façon de représenter une citation;
- on peut facilement identifier plusieurs éléments du modèle OHCO simplement en pointant des parties d'un texte et en demandant au lecteur quels sont leurs noms.

2.3.3 Les multiples hiérarchies structurelles d'un texte

Selon DeRose et al., beaucoup de documents ont plus d'une structure utile. Par exemple, la Bible a au moins trois hiérarchies disjointes :

- une hiérarchie basée sur les références : elle est constituée de testaments, livres, chapitres et verses;

- une hiérarchie thématique : elle est constituée de péricopes[27], de paragraphes et de phrases;

- une hiérarchie de dispositions (mise en page) : elle est propre à chaque édition et est constituée de pages, de colonnes et de lignes.

Par conséquent, il est possible de définir plus d'un type de structure OHCO qui soient valides pour un document. Toutefois, il est difficile de représenter à l'aide des mécanismes d'édition structurée (i.e. : SGML et autres) plus d'une hiérarchie simultanément. De plus, certains éléments, tels les liens hypertextes ou les références croisées, font qu'un texte n'a pas une structure purement hiérarchique mais plutôt une structure se rapprochant de celle d'un réseau (Renear et al., 1996).

2.4 Résumé du chapitre

Dans ce chapitre, on a défini l'essence du document technique à l'aide du standard SGML. On peut diviser un document selon trois niveaux d'information : sa structure générique (DTD), son contenu en information utile (fichier SGML) et sa présentation.

On a exposé les modèles de textes alternatifs tels que vus par DeRose et al.. On prend alors connaissance de la diversité des structures électroniques existantes mais surtout de

[27] Extrait d'un texte logiquement relié qui est sélectionné pour certaines considérations. Il peut s'agir d'une simple ligne, d'un proverbe, d'un sermon, etc..

la limite que ces modèles imposent à l'élaboration et à l'exécution des traitements automatiques évolués.

DeRose et al.propose le modèle OHCO, une hiérarchie ordonnée d'objets de contenu. Ce type de structure, qui peut être représenté à l'aide de la technologie SGML, offre les caractéristiques nécessaires à l'élaboration des traitements évolués.

Un point important soulevé par DeRose et al. concerne le fait qu'il existe plusieurs hiérarchies valides pour un texte donné. Cet aspect sur la structure d'un texte permet d'orienter et de raffiner la démarche des présentes recherches. En effet, afin d'atteindre les objectifs précisés dans la problématique, on doit pratiquement définir une structure valide, qui est l'une des multiples structures valides de type OHCO, reposant sur des bases sémantiques.

Dans le chapitre suivant, on présente les macrostructures de Van Dijk. Cette théorie du domaine de l'analyse de discours fournit les bases à l'élaboration de la structure cognitive de document.

CHAPITRE III –

LES MACROSTRUCTURES ET LA STRUCTURE GLOBALE D'UN TEXTE

Ce chapitre expose la notion de macrostructure qui a été introduite par Van Dijk (1980), dans un effort pour définir une grammaire du texte. Van Dijk s'est intéressé depuis plus de 25 ans à l'analyse du discours, qui peut être parlé ou écrit, et il possède une bibliographie impressionnante. La liste de ses livres les plus importants est présentée à l'Annexe I. La thèse sur laquelle repose ses travaux est qu'un texte n'a pas seulement une microstructure, qui exprime les relations entre les mots et les phrases, mais aussi une structure plus générale qui en définit la cohérence et l'organisation globale. Selon Van Dijk, la macrostructure a une base cognitive (traitement et représentation de l'information) importante. Il s'agit d'une structure qui est construite par une personne lors de la lecture afin d'organiser la représentation mentale d'un texte. Elle explique comment un lecteur comprend ce qui est le plus important dans un texte.

Dans ce chapitre on présente la macrostructure sémantique et la superstructure qui sont les notions centrales à la constitution de la structure sémantique globale d'un discours. Finalement, on approfondit l'aspect cognitif de la macrostructure.

3.1 La macrostructure sémantique

3.1.1 Définition

Selon Van Dijk, les macrostructures peuvent être obtenues suite à l'application de quatre macrorègles (suppression ou sélection, construction, généralisation et règle de dérivation nulle) sur la microstructure d'une séquence textuelle. Ces macrorègles permettent de relier le sens des mots et des phrases (structure locale) à une macrostructure qui représente le sujet (thème ou autre) du fragment de texte analysé. Étant donné que les macrorègles peuvent s'appliquer aux microstructures et aux macrostructures directement, on peut avoir plusieurs niveaux de macrostructures. Ceci implique une organisation hiérarchique de celles-ci.

La fonction de la macrostructure est double. Premièrement, elle organise l'information complexe en permettant de faire des liens entre des unités d'information à un niveau local afin de former des entités qui ont leur propre sens et fonction. Deuxièmement, la macrostructure permet de réduire l'information complexe, elle reflète l'importance, la signification, l'abstraction et l'information générale contenues dans un fragment de texte.

3.1.2 La macrostructure : outil d'analyse de la cohérence

Les macrostructures sont nécessaires pour traiter de la notion de cohérence. Un discours est cohérent non seulement au niveau de sa microstructure (i.e. : entre les phrases) mais aussi au niveau global. Le fragment de texte de la Figure 3.1 permet d'illustrer la notion de cohérence.

John était malade, il a donc appelé le docteur mais celui-ci ne pouvait venir car sa femme voulait aller au théâtre avec lui. Il jouait Othello et elle pensait qu'ils ne pouvaient pas la manquer parce que Shakespeare est l'un des rares auteurs dramatiques qui...

Figure 3.1 – Extrait d'un texte non cohérent (Van Dijk, 1980)

On voit dans cet exemple qu'un fait peut être une condition, une cause ou une raison pour un autre fait, mais l'ensemble de ce texte n'est pas cohérent, notamment :

- il passe d'un sujet à un autre sans orientation , sauf une orientation linéaire;
- le thème de l'extrait est "La maladie de John", mais Shakespeare n'a pas de lien avec celui-ci;
- il manque une organisation sémantique à l'extrait, on s'attend à ce que le discours soit organisé autour d'un noyau sémantique comme un thème ou un sujet.

Il est difficile de comprendre cet extrait étant donné l'absence de sens global. Intuitivement, afin de conserver un texte globalement cohérent, il faut que les extraits qui le composent aient une certaine invariance sémantique. La notion de thème ou de sujet impose cette invariance.

3.1.3 L'expression de la macrostructure dans le texte

Les macrostructures peuvent être exprimées explicitement dans le texte par des mots ou des phrases de nature thématique. Bien que l'extrait précédent ne soit pas cohérent, on note que la première phrase est une proposition de type thématique, elle annonce le

thème ou le sujet de l'extrait. Donc, la macrostructure de cet extrait pourrait s'exprimer avec la macroproposition (proposition déduite à l'aide des macrorègles) suivante : "La maladie de John".

Il existe d'autres moyens d'exprimer les macrostructures dans un texte, comme par exemple avec le résumé ou la conclusion. Le résumé permet d'annoncer les grands thèmes qui seront abordés dans le texte en question, tandis que la conclusion reprend les principaux thèmes de l'argumentation d'un texte.

3.2 La superstructure du texte

La superstructure est un schéma conventionnel qui détermine la forme globale d'un texte et, par conséquent, le contenu en macrostructures (type et organisation). La Figure 3.2, tirée du livre de Van Dijk (1980), expose la superstructure conventionnelle d'un texte de type argumentatif.

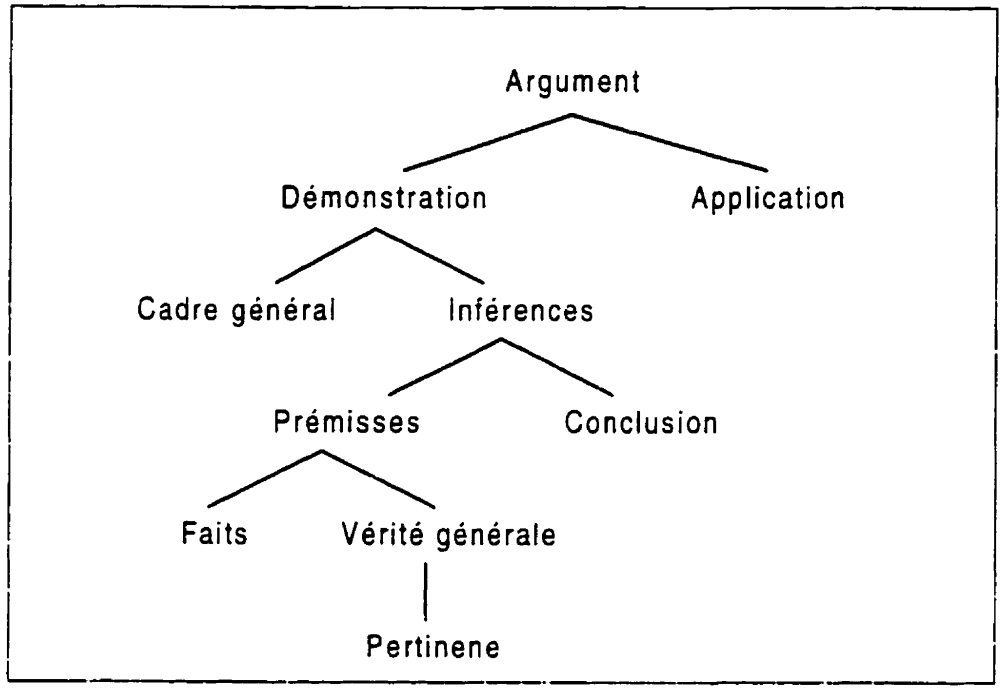


Figure 3.2 - Superstructure d'un texte de type argumentatif (Van Dijk, 1980)

La structure est constituée de catégories qui sont organisées hiérarchiquement. Ces catégories sont des endroits où l'on peut insérer le contenu du discours, soit les macrostructures. Il est évident que l'ordre de présentation peut être différent, il s'agit d'une convention. De même, il est possible que certaines catégories soient manquantes ou présentes plus d'une fois dans un texte.

3.3 La structure globale du texte

La structure globale permet de représenter le sens du texte, elle se compose en partie de la superstructure ainsi que d'un étage de macrostructures sémantiques. La Figure 3.3 :nontre la composition de celle-ci.

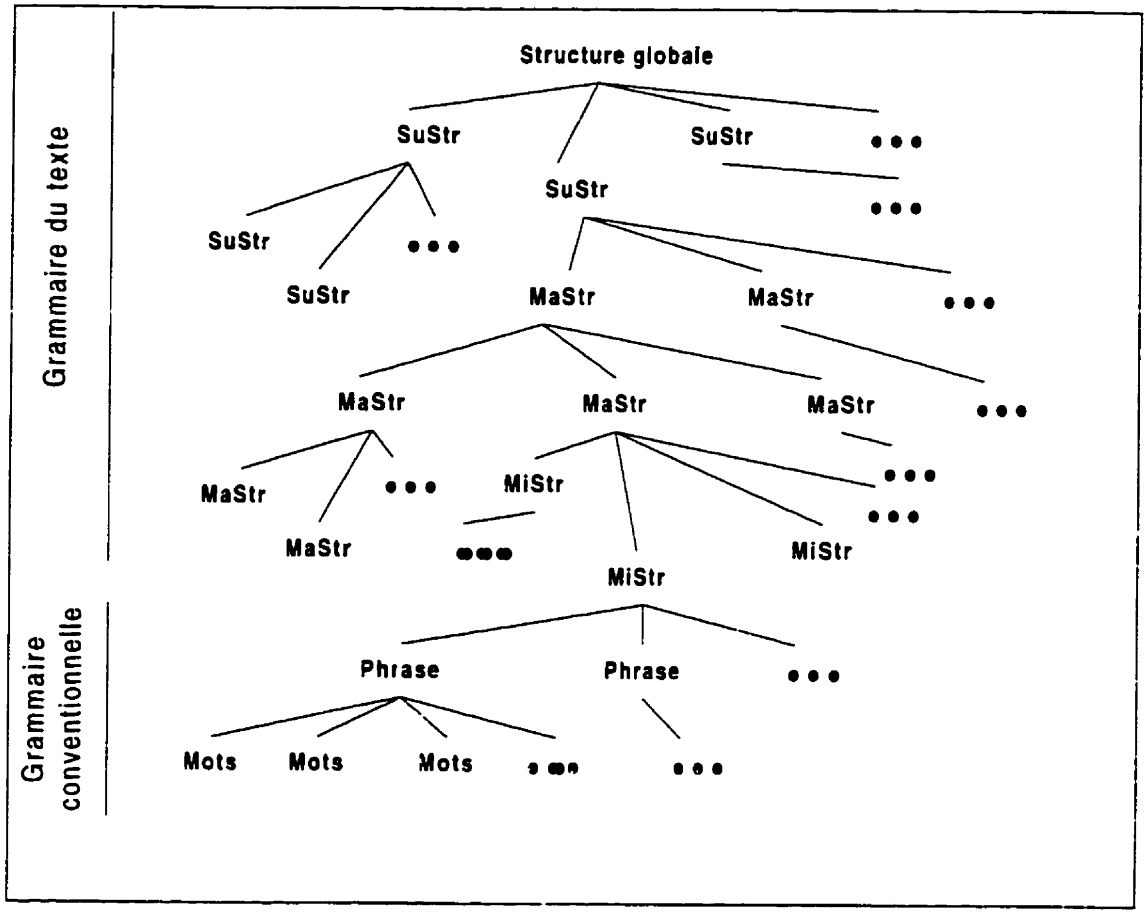


Figure 3.3 – La structure globale d'un texte

On remarque premièrement que la structure globale est divisée en deux catégories : la grammaire conventionnelle et la grammaire du texte. La grammaire conventionnelle comprend les éléments sémantiques de la microstructure (MiStr) ainsi que ceux qui lui sont subordonnés, comme les phrases et les mots. La grammaire du texte comprend les éléments sémantiques proposés par Van Dijk, soient la macrostructure (MaStr) et les catégories (SuStr) de la superstructure. On note qu'il peut avoir plusieurs étages de macrostructures et de catégories de superstructures.

3.4 La base cognitive de la macrostructure

Contrairement à la connaissance sur les processus de compréhension locaux (microstructure), la connaissance sur les processus cognitifs de la compréhension du discours au niveau macro est très peu développée. Selon Van Dijk, il existe quelques modèles qui sont plutôt simples et très théoriques. Afin d'expliquer la base cognitive de la macrostructure, Van Dijk utilise un modèle qui est explicité à la Figure 3.4.

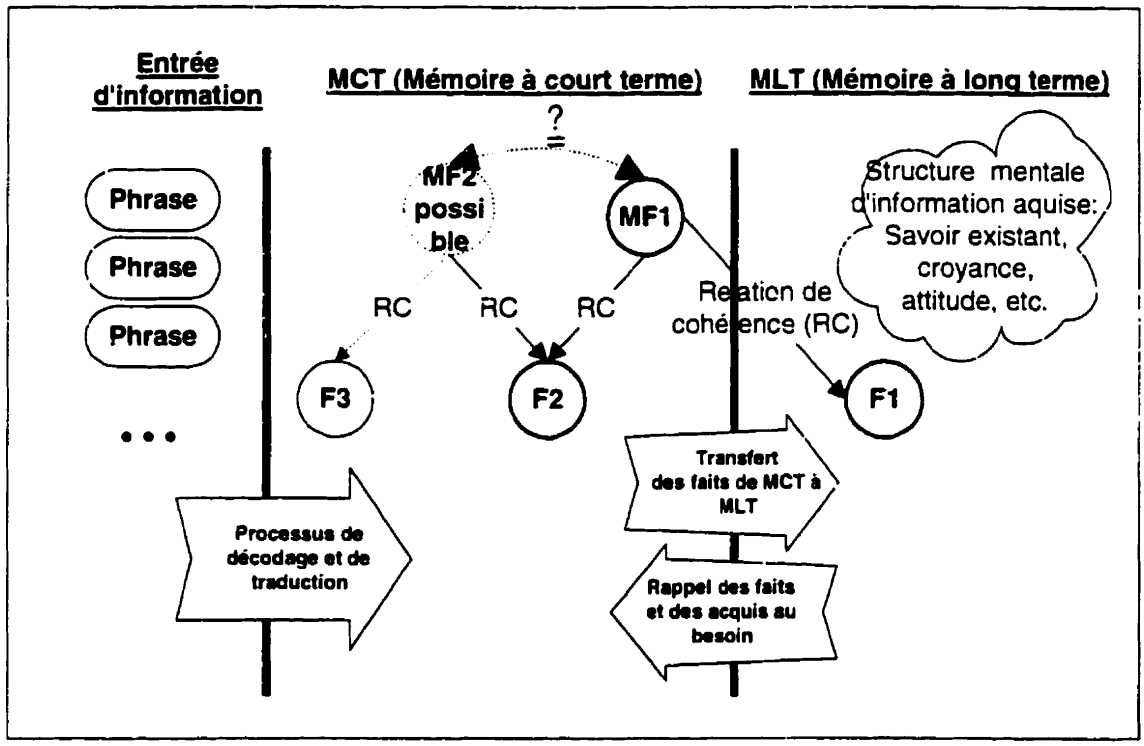


Figure 3.4 - Modèle d'un processus de traitement du discours

Le modèle est basé sur la notion de traitement sémantique de l'information[28]. On remarque les types de mémoire suivants :

- Mémoire à court terme (MCT) : Ceci consiste en la mémoire de travail, elle est très limitée en capacité de stockage.

- Mémoire à long terme (MLT) : Cette mémoire comporte deux fonctions. La première fonction est de conserver l'information qui a été traitée avec toutes les informations contextuelles (temps, endroit, circonstance, ...), ce qui constitue la mémoire épisodique. La deuxième fonction est de conserver le savoir conceptuel, les croyances, l'attitude, etc., ce qui constitue la mémoire topique.

L'élément atomique qui est véhiculé dans le modèle est le *Fait* (F). Celui-ci se définit comme la représentation cognitive d'un événement, d'une action, de l'état d'un événement qui prend place à un temps particulier, à un endroit particulier, sous des circonstances particulières et dans un univers possible particulier.

Les faits sont déduits suite au décodage des phrases de l'extrait textuel. Il est à noter que la correspondance entre une phrase et un fait n'est pas univoque. En effet, une phrase peut générer plusieurs faits ou inversement (plusieurs phrases peuvent générer un fait), ce qui explique la raison du processus de décodage[29]. Tel que montré, le système modélise un sujet qui lie un extrait où l'on peut voir que trois faits (F1, F2, F3) sont traités.

---

[28] *Semantic information processing.*
[29] L'explication de ce processus dépasse le cadre de cette recherche.

Selon Van Dijk, les faits doivent nécessairement être connectés et organisés ensemble, étant donné la capacité de stockage limitée de la MCT. Une fois les faits organisés, il est alors possible de les transférer dans la MLT (ce qui est le cas de F1 dans la figure). Van Dijk assume que les faits sont unis par une relation de cohérence (RC) qui se solde par la création d'un macrofait (MF1). En d'autres mots, la compréhension des phrases ainsi que leur séquence implique la production de propositions qui sont des structures conceptuelles nécessaires au traitement et à l'assimilation de l'information. La proposition que constitue le macrofait exprime le sujet ou le thème qui relie les deux faits.

Il s'ensuit que la macrostructure correspond une fois traduite aux macrofaits. Sa synthèse est nécessaire afin d'organiser la représentation mentale d'un texte. Elle explique la façon dont un lecteur perçoit et comprend ce qui est le plus important dans le texte.

3.5 Discussion et limite des travaux de Van Dijk

Dans son livre, Van Dijk (1980) présente une méthode semi-formelle et ad hoc qui permet d'extraire à l'aide des macrorègles les macrostructures d'un discours existant. Il est donc possible que l'on obtienne des structures globales différentes pour un même texte.

De plus, il démontre que la macrostructure est proche de la représentation mentale du lecteur, cependant il n'aborde pas les aspects touchant aux performances de traitement de l'information par le lecteur en fonction de la qualité de la structure globale.

Van Dijk n'a pas poussé son étude jusqu'à catégoriser les types de macrostructures que l'on peut avoir dans un texte. La macrostructure constitue avant tout un outil pour l'analyse de discours afin de pouvoir analyser la structure d'un texte.

3.6 Résumé du chapitre

Dans ce chapitre, on a présenté la théorie sur les macrostructures de Van Dijk et comment ces structures permettaient, à l'aide de la superstructure, de représenter la structure globale d'un texte. De plus, il est évident que la macrostructure est de nature sémantique. Elle a une base cognitive importante car elle représente le type de structures que le lecteur doit créer lorsqu'il lit et assimile le contenu d'un texte. Pratiquement, on a vu que les macrostructures représentaient le sujet ou le thème d'un extrait textuel.

CHAPITRE IV –

ÉLABORATION DU MODÈLE SMML DE TYPE OHCO BASÉ SUR LES MACROSTRUCTURES DE VAN DIJK

À la lumière des concepts qui ont été exposés dans les chapitres précédents, on propose maintenant le modèle SMML, qui signifie *Semantic Macrostructure Markup Language*, basé sur les macrostructures de Van Dijk. La structure du modèle proposée est du type OHCO, il s'agit donc d'une hiérarchie d'objets de contenu. On définit en premier lieu la représentation de l'élément de base du modèle, soit la macrostructure. En second lieu, on discute de l'organisation hiérarchique du modèle.

4.1 Définition d'une unité de base

Un point de réflexion important se pose maintenant : comment représenter la macrostructure sémantique dans le modèle? Selon Van Dijk (1980), il s'agit essentiellement d'une macroproposition qui exprime le sujet ou le thème d'un extrait. Cette définition n'est pas suffisante si l'on désire intégrer explicitement la notion de macrostructure à même le texte, et non seulement constituer une structure distincte comme cela se produit suite à l'application des macrorègles. On veut intégrer les éléments de la structure globale (superstructure et macrostructure) à même le texte afin de les voir explicitement.

4.1.1 Les travaux de Danes

Danes (1970, 1974) a fait une étude sur la notion de progression thématique dans les textes de type argumentatif. Ses travaux lui ont permis de définir certains types de progressions, par exemple : la progression à thème constant ou la progression linéaire. L'ensemble des progressions est présentée à l'Annexe III.

L'intérêt de ses travaux n'est pas directement lié aux résultats obtenus mais à la structure thématique qu'il a utilisée pour représenter les progressions. La structure thématique se décompose en deux éléments, le thème et le rhème. Le thème est essentiellement le sujet dont on parle tandis que le rhème est constitué par ce qui est dit sur le sujet ou le thème.

En utilisant cette structure pour définir une macrostructure, il est possible d'intégrer la structure globale au texte. La macroproposition représentant la macrostructure étant exprimée à l'aide de l'élément thème et le contenu du texte étant représenté avec l'élément rhème. La Figure 4.1 montre la définition de l'élément *MaStr* qui représente la notion de macrostructure, selon le formalisme SGML.

```
<!-- Définition de la MacroStructure >
<!ELEMENT MaStr - - (theme,rheme)>
<!-- Définition du thème >
<!ELEMENT theme - - (#PCDATA)>
<!-- Définition du rhème >
<!ELEMENT rheme - - (MaStr|MiStr)+>
```

Figure 4.1 – Définition de l'élément *MaStr*

Cette définition de types, qui constitue un DTD, définie un élément nommé *MaStr* qui est composé à l'aide des éléments *theme* et *rheme*. L'élément *theme* est une chaîne de

caractères qui contient la macroproposition[30]. Le *rheme* se compose d'éléments *MaStr* et/ou d'éléments *MiStr*. L'élément *MiStr* correspond à la microstructure. On considère les éléments suivants pour la microstructure : texte, image, graphique, lien hypertexte, etc. Bref, tous les éléments de base servant à l'édition d'un texte.

4.2 Le Modèle SMML (Semantic Macrostructure Markup Language)

La Figure 4.2 montre le modèle SMML de base qui respecte la théorie de Van Dijk sur les macrostructures. Notamment, un texte est représenté par une structure globale (élément *GlStr*) qui se compose d'une ou de plusieurs superstructures (élément *SuStr*). L'élément *SuStr* se définit un peu comme l'élément *MaStr*. Il se compose d'un thème (élément *theme*) suivi de superstructures[31] et/ou de macrostructures.

```
<!-- DTD du SMML(Semantic Macrostructure Markup Language)>
<!ELEMENT SMML - - (GlStr)>
<!-- Définition de la Structure Globale >
<!ELEMENT GlStr - - (SuStr)+>
<!-- Définition de la SuperStructure >
<!ELEMENT SuStr - - (theme, (SuStr |MaStr))+>
<!-- Définition de la MacroStructure >
<!ELEMENT MaStr - - (theme,rheme)>
<!-- Définition du thème >
<!ELEMENT theme - - (#PCDATA)>
<!-- Définition du rhème >
<!ELEMENT rheme - - (MaStr|MiStr)+>
<!-- Définition de la MicroStructure >
<!ELEMENT MiStr - - (texte|image|graphique|hyperlien)+>
```

Figure 4.2 – Le modèle SMML de base

[30] On rappelle que la macroproposition est déduite de l'extrait textuel suite à l'application des macrorègles et qu'elle correspond à la macrostructure.
[31] Ceci permet de faire une hiérarchie d'élément *superstructure*.

Le modèle SMML, tel que défini, permet de reproduire la structure globale montrée à la Figure 3.3 - La structure globale d'un texte. Il permet d'éditer et de représenter électroniquement un document en mettant l'accent sur sa structure sémantique et non sur sa présentation graphique.

Cette déclaration de type DTD reste très générale. Dans les sous-sections suivantes, on propose quelques modifications à la structure de base afin d'ajouter des caractéristiques souhaitables au modèle SMML.

4.2.1 Définition de la notion de classes pour les macrostructures

Les deux extraits suivants (Figure 4.3 et Figure 4.4) permettent d'expliquer la notion de classe d'une macrostructure. Les thèmes respectifs de ces extraits pourraient être dans l'ordre : « Le logiciel » et « Le génie logiciel ».

> Selon le Petit Robert (1985), un logiciel est l'ensemble des travaux de logique, d'analyse, de programmation, nécessaires au fonctionnement d'un ensemble de traitement de l'information.

Figure 4.3 – La définition du logiciel

> Le génie logiciel est un domaine de l'informatique qui concerne la réalisation de systèmes logiciels qui sont suffisamment grands et complexes pour être réalisés par des équipes d'ingénieurs.

Figure 4.4 – La définition du génie logiciel

À un niveau d'abstraction plus grand, soit en généralisant, on note qu'il s'agit de définitions. Dans la mesure où ces extraits sont contigus dans un texte, on peut définir un élément *MaStr* dont le thème est « Définitions » et dont le rhème englobe les deux

thèmes précédents (il s'agit d'un second niveau de macrostructure). Cependant, si les extraits sont distants dans le texte, cette approche ne peut pas s'appliquer. Afin d'expliciter le lien entre ces macrostructures, on peut identifier la nature de celles-ci à l'aide d'un attribut qui indique la classe. Dans l'exemple, les éléments *MaStr* auraient un attribut *classe* de type caractère qui se définirait à l'aide de la règle suivante:

```
<!ATTLIST MaStr classe CDATA>
```

Dans l'exemple, l'attribut classe aurait la valeur « Définition » ou autre chose d'équivalent. La Figure 4.5 montre le DTD du SMML avec les modifications.

```
<!-- DTD du SMML(Semantic Macrostructure Markup Language)>
<!ELEMENT SMML - - (GlStr)>
<!-- Définition de la Structure Globale >
<!ELEMENT GlStr - - (SuStr)+>
<!-- Définition de la SuperStructure >
<!ELEMENT SuStr - - (theme, (SuStr |MaStr))+>
<!-- Définition de l'attribut classe pour la superstructure >
<!ATTLIST SuStr classe CDATA>
<!-- Définition de la MacroStructure >
<!ELEMENT MaStr - - (theme,rheme)>
<!-- Définition de l'attribut classe pour la macrostructure >
<!ATTLIST MaStr classe CDATA>
<!-- Définition du thème >
<!ELEMENT theme - - (#PCDATA)>
<!-- Définition du rhème >
<!ELEMENT rheme - - (MaStr|MiStr)+>
<!-- Définition de la MicroStructure >
<!ELEMENT MiStr - - (texte|image|graphique|hyperlien)+>
```

Figure 4.5 – Le modèle SMML avec la notion de classe

On voit que l'attribut *classe*, qui est une chaîne de caractères, a été ajouté aux éléments *MaStr* et *SuStr*.

Cette notion de classe n'est pas présente dans les études de Van Dijk sur la macrostructure. Elle comporte des avantages intéressants, notamment :

- elle permet d'annoncer et de juger du contenu textuel d'une macrostructure;
- elle offre un classement de second niveau qui permet de comparer les macrostructures entre elles;
- elle permet l'élaboration de certains traitements évolués, par exemple : on peut composer une liste de définitions en parcourant le document à la recherche des macrostructures de classe « définition ».

4.3 Validation de la structure du modèle comme OHCO

Avec la définition donnée par DeRose et al. (1980), le modèle est du type OHCO pour les raisons suivantes :

- il s'agit d'une hiérarchie;
- l'objet de contenu est la macrostructure[32];
- il existe plusieurs hiérarchies possibles pour représenter la structure du document (DeRose et al., 1980). L'utilisation de la structure globale au sens de Van Dijk en est une. Le SMML permet d'expliciter la structure de discours d'un texte.

---

[32] Étant donné que l'organisation d'une superstructure est conforme à celle d'une macrostructure, on traitera seulement de la macrostructure par la suite. L'inclusion de la superstructure est donc implicite.

4.4 Discussion du modèle de documentation SMML

La définition de la notion de classes de macrostructure ouvre à de nombreuses considérations qui seront maintenant discutées. Notamment, quelles classes de macrostructures est-il possible de définir? Quels sont leur organisation et leur relation? Est-il souhaitable de définir un ensemble de macrostructures pour une classe de documents en particulier, par exemple, un document de requis ou de design? Existe-t-il un catalogue de macrostructures qui pourrait fournir un ensemble de classes de base pour l'édition? etc.

Parmi les questions posées, la dernière a été étudiée en premier lieu. En effet, disposer d'un tel catalogue de macrostructures serait très intéressant.

4.4.1 Compte rendu de recherche sur le catalogue de macrostructures

Dans le cadre de la recherche, on a d'abord tenté de faire l'inventaire des structures ou des macrostructures susceptibles d'être directement intégrées au modèle SMML.

Suite à plusieurs recherches et discussions avec des experts[33] du domaine, on en est venu à la conclusion qu'il n'existait pas de catalogue de telles structures. Pratiquement, on sait que les macrostructures existent, cependant on ne possède pas actuellement d'inventaire.

---

[33] Notamment avec M. Natan Ménard, professeur en analyse de textes du département de linguistique et de traduction de l'Université de Montréal.

Par conséquent, la solution envisageable consiste à concevoir un modèle ouvert qui soit évolutif. Celui-ci devrait permettre à l'utilisateur de définir lui-même les classes de macrostructures qu'il juge significatives et pertinentes.

4.4.2 Définition d'un catalogue de macrostructures

On présente dans cette section la notion de patron qui sera utilisée afin de définir le catalogue de macrostructures.

4.4.2.1 Les patrons : pour structurer l'information évolutive

En 1977, Christopher Alexander (1977) publie le livre « A Pattern Language – Towns, Buildings, Construction » un livre qui décrit une nouvelle approche de l'architecture, la construction et la planification. Dans son livre, il présente un langage pratique de patrons issu de son expérience en architecture. Les éléments de son langage sont des entités nommées patron. Il en donne la définition suivante :

> « Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. »

Pour des raisons de commodité et de clarté, Alexander propose un format unique afin de décrire tous les patrons[34] qui constituent son catalogue.

De son coté, Coplien (1994) croit que la notion de patron est particulièrement intéressante pour analyser et concevoir des entités organisationnelles et évolutives. Il présente dans son article un catalogue de patrons[35] qui peuvent être utilisés pour construire une organisation ou une entreprise dans le domaine du logiciel, et pour guider son processus de développement (formation de l'entreprise). Selon l'auteur, les patrons aident non seulement à comprendre les organisations existantes mais aussi à en construire de nouvelles.

La notion de patron a aussi trouvé sa place dans les techniques de conception orientée objet avec les « Design Pattern ». Selon Coplien (1994), elle offre une nouvelle façon de comprendre et de créer des programmes informatiques. Par exemple, Gamma et al. (1995) proposent un catalogue de 23 patrons qui nomme, abstrait, motive, explique et identifie les aspects clés de certaines structures de design qui sont utiles afin de concevoir un design orienté-objet réutilisable. Les « Design Pattern » adressent un problème de design qui revient souvent dans les systèmes orientés-objets.

Selon l'auteur, les patrons de conception ont les quatre composantes essentielles suivantes:

---

[34] Il définit 253 patrons au total.

Un nom : il identifie le patron. Les noms offrent une abstraction de haut niveau et constituent un vocabulaire de design qui peut être utilisé par les ingénieurs pour communiquer, documenter ou réfléchir.

Un problème : cette composante décrit quand on doit utiliser le patron de conception. il explique le problème et le contexte.

Une solution : cette composante décrit les éléments et les relations (diagramme de classes) qui constituent le design.

Une conséquence : le coût et les conséquences associés à l'utilisation de la solution.

Tout comme Alexander et Coplein, ils proposent une description unique afin de représenter chacun de ces patrons[36]. Pourquoi utiliser une description complète et non seulement une définition simple? Pour faciliter la réutilisation des patrons, en faciliter la compréhension, la comparaison, la communication et l'utilisation.

4.4.2.2 Définition du catalogue de macrostructures à l'aide de la notion de patrons

Le contexte d'utilisation et de définition des classes de macrostructure est compatible avec la notion de patron. En effet, la classe représente la composante récurrente d'un certain ensemble de macrostructures.

---

[35] Un langage qui comporte 39 entités.
[36] Cette description est incluse en annexe IV.

On présente dans cette section une interface afin de décrire une classe de macrostructures. Il s'agit d'une proposition de base théorique. Cette description est appelée à évoluer au même titre que le catalogue qu'elle constitue.

Afin de décrire une classe de macrostructures, il faut compléter les aspects présentés à la Figure 4.6.

NOM DE LA CLASSE DE LA MACROSTRUCTURE

Il identifie le patron. Les noms offrent une abstraction de haut niveau et constituent un vocabulaire de design qui peut être utilisé par les ingénieurs pour communiquer, documenter ou réfléchir.

Type

Il s'agit de son classement dans le catalogue. On propose une classification par type de document, par exemple : design, requis, etc.

Définition

Que représente la classe de macrostructure, quels sont l'intention, le but ou le rationnel de son élaboration?

Structure

S'il s'agit d'une classe de macrostructures composite, qui se définit par un arrangement complexe d'autres classes de macrostructures, quelles sont la composition de la classe ainsi que les relations d'inclusion et d'exclusion des autres classes?

Exemples d'utilisation

Les cas typiques dans lesquels il est possible d'utiliser cette classe de macrostructures.

Macrostructure associée

La liste des classes de macrostructures qui ressemblent à celle-ci en terme de structure ou autre.

Figure 4.6 – Canevas de présentation d'une classe de macrostructure[37]

[37] La présentation graphique est inspirée des travaux de Trudel (1995)

4.4.3 Limites et aspects complémentaires à l'utilisation du modèle SMML et du catalogue

Le modèle SMML, tel que défini, constitue un modèle générique et évolutif pour les documents. Le développement et l'utilisation d'un catalogue basé sur la théorie des patrons permettent cette évolution. Par conséquent, la conception d'un éditeur doit supporter ces deux composantes. La Figure 4.7 modélise un éditeur implantant les deux composantes.

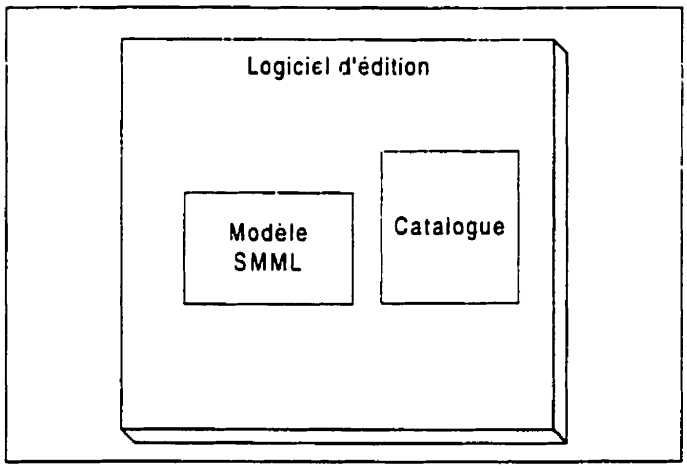


Figure 4.7 – Structure d'un logiciel implantant le modèle SMML avec le catalogue

De plus, l'implantation de la notion de classe comme un attribut ne permet pas de spécifier l'organisation des classes entre elles. Une classe *définition* peut-elle contenir une autre classe *définition*, même si cela n'a pas de sens? Il faut que le catalogue offre une structure permettant la définition de l'organisation des classes et que le logiciel d'édition l'applique. La gestion de l'organisation est normalement prise en compte par le standard SGML via le DTD.

Le moyen de pallier à ce problème est de définir des modèles SMML statiques spécifiques à un document en particulier, par exemple pour un document de requis. Dans la mesure où l'on dispose d'un ensemble de classes suffisant et adapté à un type de document il est possible de concevoir un modèle SMML de document spécifique[38]. Il suffit d'intégrer le catalogue directement dans la définition des types (DTD) du modèle. Les classes de macrostructures possibles ainsi que leur organisation sont alors statiques, il est impossible de les modifier. La Figure 4.8 modélise le logiciel qui implante un modèle pour un document de design.

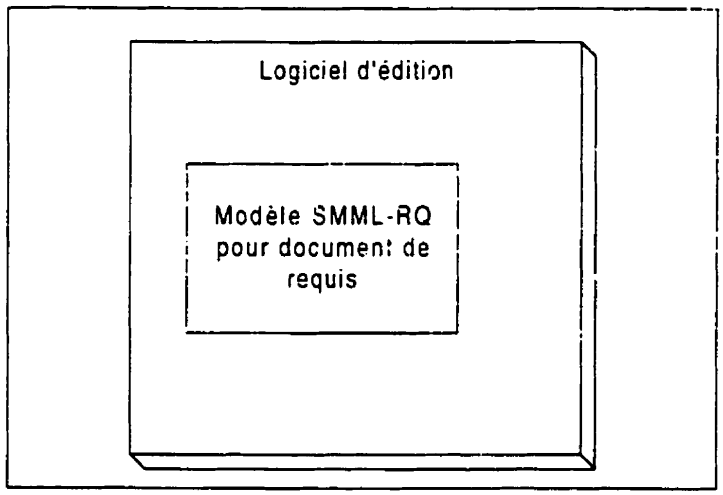


Figure 4.8 – Structure d'un logiciel implantant le modèle SMML-RQ

On remarque que le logiciel supporte le SMML-RQ, un langage qui permet d'éditer des documents de requis.

[38] Ce qui signifie que l'on possède un langage de macrostructure avec un nombre d'entités suffisants.

4.4.4 Intégration des standards et traduction de documents techniques de génie logiciel

Que peut-on faire avec les millions de pages de documents existants? Peut-on les convertir au modèle SMML facilement? Quels sont les possibilités et obstacles d'une telle entreprise?

L'extrait suivant, tiré de l'article de Bernhardt (1986) permet de mettre en relief la problématique sous-jacente à la traduction d'un document:

> « At all levels of structure, texts which are highly informative visually share features not characteristic of texts which do not exploit the graphic potential of written language. We can assume that visually informative texts achieve rhetorical organization, just as do texts which are relatively non-informative visually. Both types must provide direction to the reader as to how the text is to be read : what transaction is intended, what the major divisions are, what is considered important, and what relations exist amongst the various subpoints. But the manner in which visually informative texts achieve rhetorical control differs in important ways from that in the non-visually informative text, at all levels of organization: in the whole discourse, in the paragraph, and in the sentence. »

Le fait que la rhétorique utilisée change profondément la composition et l'organisation du discours en fonction du degré d'information qui est transmis visuellement, apporte un point de réflexion important. On peut s'attendre à ce que les auteurs changent leurs techniques de rédaction. Ceci va se traduire par une organisation différente des

informations que l'on trouve actuellement dans les documents écrits à l'aide des éditeurs usuels.

Par conséquent, afin de pouvoir traduire les documents existants, il faudrait connaître ces différences. Par la suite, il serait possible de concevoir un schéma de traduction afin de faire le lien entre un document exprimé à l'aide d'une structure « flot d'objet de contenu » et le même document exprimé à l'aide de la structure SMML.

Dans un autre ordre d'idée, les standards, qui régissent la rédaction des documents du processus, définissent une structure générale de haut niveau. Ceci semble être en accord avec la notion de superstructure définit par Van Dijk. Par conséquent, il serait possible à première vue d'utiliser ces standards afin de définir les supercontextes là où cela s'applique. Il s'agit toutefois d'une hypothèse qui doit être validée pratiquement.

4.5 Résumé du chapitre

Dans ce chapitre, on a présenté le modèle SMML (Semantic Macrostructure Markup Language) à l'aide de la technologie du SGML. Le SMML permet d'expliciter la structure sémantique du discours d'un document en pointant les macrostructures du texte. Ceci contraste avec les autres langages par exemple comme le HTML qui pointe principalement des constructions de mise en page.

Le SMML utilise les notions de macrostructure, de superstructure et de structure globale telles que développées par Van Dijk et les intègre explicitement au texte grâce aux notions de thème et de rhème. On a défini la notion de classe afin de fournir un degré d'abstraction supplémentaire aux macrostructures. Ceci permet l'élaboration de traitements évolués génériques sur les documents SMML comme la composition d'une liste de définitions, par exemple.

Du fait qu'il n'existe pas de répertoire de macrostructure, on a utilisé la notion de patron et proposé une interface afin de décrire les classes de macrostructures et ainsi constituer un catalogue. La notion de patron est particulièrement bien adaptée pour supporter un contenu d'information évolutif.

On a discuté de la pertinence de concevoir un langage SMML spécifique pour un type de document, comme le SMML-RQ pour une structure de document de requis.

Finalement, on a exposé les problèmes inhérents à la traduction des documents déjà existants et conçus à l'aide d'éditeurs conventionnels. Notamment, le fait de changer la structure et l'organisation visuelle d'un document devrait amener une évolution de la rhétorique, ce qui veut dire qu'un document SMML ne serait pas composé de la même manière qu'un document conventionnel.

Dans le chapitre suivant, on propose une présentation graphique afin de représenter un document SMML à l'écran ou sur papier.

CHAPITRE V –

PRÉSENTATION DU MODÈLE DE DOCUMENT SMML

L'objectif de ce chapitre est de proposer une présentation pour un document SMML. Pour ce faire, on se base principalement sur les considérations apportées par Bernhardt.

5.1 Seeing the Text (Bernhardt, 1986)

Pour un lecteur, le fait que le texte soit physiquement représenté, avec son organisation spatiale sur une page, requiert une approche visuelle. Le texte doit être vu, ce qui implique un processus de décodage différent de celui de la perception de la parole. Dans le cas d'un auteur, le mode de communication écrit demande de concevoir un arrangement spatial, un processus d'encodage par lequel l'auteur donne des indices visuels sur l'organisation du texte.

On peut classer les textes selon un continuum, d'un côté les textes qui communiquent relativement peu d'information visuellement et, de l'autre, ceux qui révèlent une information substantielle sur la structure du texte via certains artifices visuels (indentation, marge, ponctuation, etc.). Selon Bernhardt l'extrême du continuum contient les textes qui explicitent clairement leur structure organisationnelle :

> « At the other extreme of the continuum would be texts which display their structure, providing the reader/viewer

with a schematic representation of the divisions and hierarchies which organize the text. »

5.2 Définition de la présentation du modèle SMML

Suite aux considérations amenées par Bernardt(1986), on relève les points importants suivants concernant la représentation graphique d'un document :

- Hiérarchie : comme le texte est une structure hiérarchique, la représentation du modèle SMML devrait permettre de retrouver cette hiérarchie plus facilement qu'avec les documents basés sur les modèles « flot d'objets de contenu ». Par exemple, les documents conçus avec l'éditeur Microsoft Word™ n'indiquent la hiérarchie d'un document qu'indirectement, via les marqueurs de type hiérarchique[39].

- Délimitation visuelle des éléments du texte : il faut distinguer la limite spatiale des éléments du texte. Dans le texte technique traditionnel, toutes les macrostructures ne sont pas délimitées explicitement dans le texte. Par exemple : il est possible qu'un extrait ait plusieurs définitions contenues dans un même paragraphe. Dans ce cas, les macrostructures ne sont pas visibles directement, seule la lecture de l'extrait nous permettra d'extraire la structure du texte.

En tenant compte de ces considérations, on propose la présentation de la Figure 5.1 afin de rendre à l'écran l'élément *MaStr* du modèle SMML.

---

[39] Hierarchical Markup.

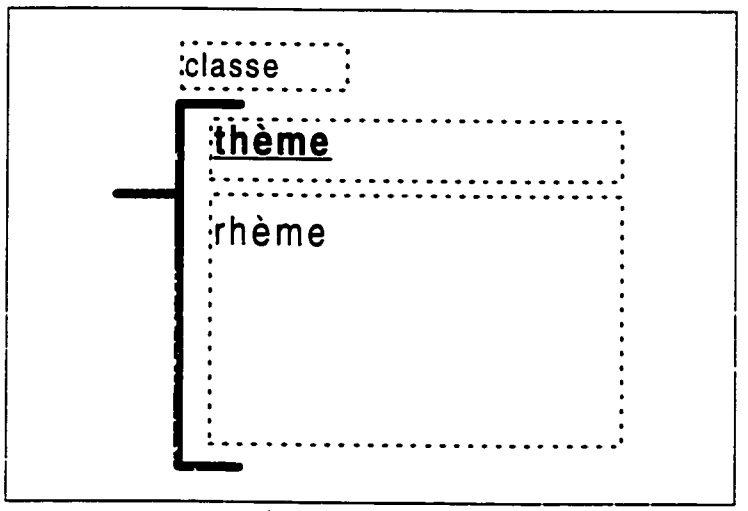


Figure 5.1 – Présentation de l'élément *MaStr* du modèle SMML

On remarque les trois éléments qui composent la macrostructure, soit la classe en haut, le thème et le rhème. Le thème est représenté en gras et est souligné afin que l'on puisse facilement déterminer la limite spatiale entre celui-ci et le rhème qui suit.

La Figure 5.2 montre l'organisation hiérarchique d'un document contenant plusieurs éléments *MaStr*.

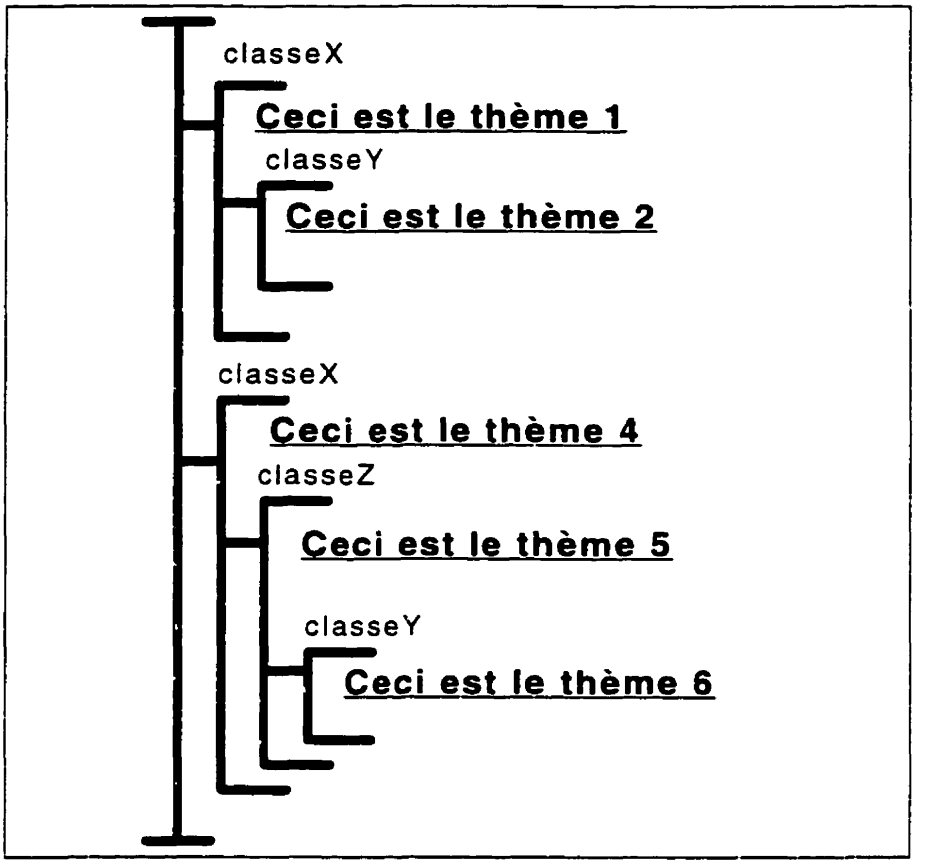


Figure 5.2 – Présentation de la structure d'un document SMML

Cette figure permet de mettre en évidence l'organisation hiérarchique d'un document SMML. Il est à noter que les éléments rhème ont été volontairement retirés, question de bien montrer l'organisation de l'ensemble. De plus, telles que présentées, il y a deux macrostructures de classe X et deux de classe Y. Cette structure pourrait représenter l'organisation d'un extrait de document de design dans lequel il y a deux composantes de design (classe X) qui possèdent chacune une définition (classe Y).

5.3 Résumé du chapitre

Dans ce chapitre, on a brièvement proposé une façon de représenter l'élément *MaStr* et ainsi représenter à l'écran ou sur papier un document SMML. Cette représentation est en accord avec les considérations apportées par Bernhardt qui considère à l'extrême de son continuum les textes qui explicitent schématiquement leur structure interne.

Dans le chapitre suivant, on valide le modèle SMML. On utilise d'une part la théorie de Van Dijk concernant l'aspect cognitif des macrostructures et d'autre part, on utilise la notion d'organisateur avancé afin de prouver théoriquement que le modèle a le potentiel nécessaire afin d'atteindre les objectifs pour lesquels il a été conçu.

CHAPITRE VI –

VALIDATION DU MODÈLE SMML

Dans ce chapitre, on démontre que le modèle SMML, tel que défini au chapitre précédent, nous permet d'atteindre les objectifs qui ont été fixés dans la problématique. Essentiellement, on doit démontrer au niveau cognitif que le modèle SMML, s'il est utilisé correctement, peut améliorer la performance du canal de communication qu'offre le document.

Premièrement, on procède à une validation théorique du modèle à l'aide des travaux de Van Dijk sur la nature cognitive de la macrostructure. Ceci permet d'inférer l'impact cognitif de l'utilisation du modèle.

Deuxièmement, on démontre que l'élément *MaStr* respecte la définition d'un AO (Advance organizers) et peut donc être traité comme tel. Plusieurs expérimentations ont été faites, notamment par Mayer et al. (1979), afin de déterminer le contexte dans lequel les AO avaient un impact cognitif positif. Il en découle un ensemble de résultats intéressants, grâce auxquels il sera possible d'approfondir l'impact du SMML.

Finalement, on discute du travail à réaliser afin de conduire une validation pratique du modèle.

6.1 Validation théorique

6.1.1 Validation du modèle selon la théorie des macrostructures de Van Dijk

Dans le Chapitre III, on a présenté le rôle de la macrostructure dans la synthèse des macrofaits. La macrostructure correspond, une fois traduite, aux macrofaits. Sa synthèse est nécessaire afin d'organiser la représentation mentale d'un texte.

Dans les textes conventionnels, les macrostructures ne sont pas directement visibles. Elles correspondent aux thèmes d'un discours. On peut les retrouver partiellement via certaines phrases à nature thématique et via les constructions comme la table des matières, l'introduction, la conclusion, le résumé, etc.

Van Dijk assume que le lecteur, durant la lecture, conduit un processus complexe d'analyse-synthèse des structures sémantiques (macrofaits), souvent à plusieurs niveaux, simultanément et/ou parallèlement, afin de construire une structure conceptuelle[40]. Durant ce processus, le lecteur doit faire des hypothèses sur les thèmes (macrofaits) qui sont pertinents, aussitôt qu'une ou plusieurs phrases fournissent assez d'informations pour faire une telle hypothèse[41].

Comme il a été mentionné précédemment, des indices très importants peuvent être donnés quant au thème ou à la macrostructure courante via les titres et les phrases de nature thématique. Les hypothèses sur les thèmes du discours sont établies plus

---

[40] Le "Résultat d'apprentissage" au sens de Mayer (1985).

[41] Il est à noter que l'assignation des thèmes par hypothèse n'est pas déterminée seulement à l'aide des informations du texte. Les acquis et le contexte permettent de générer des hypothèses sur le thème probable de l'extrait courant.

spécifiquement par l'interprétation de ce type d'expression. Les titres peuvent fournir de l'information sur le thème du texte dans son ensemble et les phrases thématiques, sur le thème des paragraphes qui suivent.

Donc, avant même d'avoir lu la première phrase d'un extrait, le lecteur peut déjà avoir un ensemble de thèmes (macrofaits) possibles à assigner. Il ne s'agit pas d'un processus a posteriori qui déduirait le macrofait une fois que tous les faits seraient présents dans la mémoire à court terme (STM) mais d'un processus dynamique dont les résultats sont validés, infirmés et raffinés suite à la traduction de chaque phrase.

Avec ces informations, il est raisonnable de croire que le modèle SMML, s'il est utilisé correctement, est en mesure de faciliter la tâche de synthèse des macrofaits et, par conséquent, de faciliter le processus de compréhension de l'extrait en entier.

Tel qu'il a été défini, le modèle SMML, ou plus précisément l'élément *MaStr*, spécifie systématiquement le sujet de son contenu via l'élément *theme*. Ceci amène deux conséquences importantes sur l'utilisation du modèle SMML, soit :

- le lecteur n'a plus à faire d'hypothèse quant au thème de l'extrait courant, la probabilité de faire une erreur lors de l'établissement de la relation de cohérence (macrofait) entre deux faits est alors grandement réduite.

- le processus d'assimilation est plus linéaire : les retours en arrière ne sont plus nécessaires afin de dégager le macrofait de l'extrait. Le fait que le thème soit explicité permet au lecteur de comprendre la première phrase du discours directement en tenant compte des thèmes plus globaux ou subordonnés.

6.1.2 Validation du modèle selon la théorie des Advance Organizer

6.1.2.1 Définition de l'Advance Organizer

Tels que rapportés par Mayer (1979), les travaux d'Ausubel (Ausubel, 1960, 1963; Ausubel & Fitzgerald, 1961, 1962; Asubel & Youssef, 1963; Fitzgerald & Ausubel, 1963) ont permis de définir l'AO. Celui-ci en a donné la définition suivante : « appropriately relevant and inclusive introductory materials ... introduced in advance of learning ... and presented at a higher level of abstraction, generality, and inclusiveness ». Sa fonction étant: « to provide ideational scaffolding for the stable incorporation and retention of the more detailed and differentiated material that follows ». Cette fonction est accomplie en manipulant : « the availability to the learner of relevant and proximately inclusive subsumers[42] ».

La notion d'AO est large, à un niveau *macro*, on peut considérer un vidéo ou un texte explicatif donné avant la lecture d'un texte cible. Par exemple, dans certaines de ses expériences, Mayer a présenté un texte explicatif sur l'ordinateur et la programmation, avant de présenter un texte de 10 pages concernant un langage de programmation simple. À un niveau *micro*, une phrase ou une image, avant un extrait textuel plus ou moins volumineux, peut être considérée comme un AO.

Il existe divers types d'AO et dans son article Mayer présente les suivants :

---

[42] Soient T, le concept « supérieur », ⊥, le concept « inférieur », C, un concept quelconque et la relation $\perp < C < T$. L'opérateur « < » est appelé « subsumption ». Par exemple : une personne (concept plus général) « subsumes » un travailleur (un concept plus spécifique), donc Travailleur < Personne (Poullet et al., 1997).

- expository organizer : ce type d'AO permet de générer la ou les relations logiques qui unissent le matériel qui doit être appris (to-be-learned material);

- comparative organizer : ce type d'AO permet de faire le lien entre une information non-familière (le matériel à apprendre) et une information familière (savoir existant). Par exemple : Ausubel expérimentait l'effet de ce type d'AO en présentant un texte sur le Bouddhisme expliqué avec des comparaisons au Christianisme.

Les AO ont normalement les caractéristiques suivantes :

- petit ensemble de mots ou d'informations visuelles;

- présenté en prélude à un extrait d'information qui doit être assimilé;

- ne contient pas d'idée spécifique appartenant à l'extrait qui doit être assimilé;

- fournit un moyen de générer le lien logique qui unit les idées de l'extrait qui doit être assimilé;

- influence le processus d'encodage du lecteur.

6.1.2.2 Validation de l'élément *MaStr* comme AO

La question qui se pose est à savoir si l'on peut considérer l'élément *theme* de l'élément *MaStr* comme un AO. Voici la définition qui a été donnée concernant la macrostructure sémantique :

> Selon Van Dijk, les macrostructures peuvent être obtenues suite à l'application de quatre macrorègles sur la microstructure d'une séquence textuelle. Ces macrorègles permettent de relier le sens des mots et des phrases (structure locale) à une macrostructure qui représente le sujet (thème ou autre) du fragment de texte analysé.

L'élément *MaStr*, tel que défini dans le modèle SMML, est composé des éléments *theme* et *rheme*. L'élément *theme* représente la macroproposition qui est déduite à l'aide des macrorègles de Van Dijk et l'élément *rheme* est le contenu de l'extrait textuel.

Suite aux définitions précédentes, on peut affirmer avec certitude que l'élément *MaStr* renferme les caractéristiques d'un AO de type *expository*. Son élément *theme* rencontre les deux premières caractéristiques. En effet, il s'agit d'un petit ensemble de mots présentés en prélude à l'extrait textuel. La définition donnée par Van Dijk sur la macrostructure permet de rencontrer les deux caractéristiques suivantes. La macrostructure fait le lien sémantique entre les éléments de l'extrait et, étant une généralisation de l'extrait, elle ne contient pas d'idée spécifique lui appartenant.

La dernière caractéristique constitue essentiellement l'impact de l'utilisation d'un AO, ce qui est expliqué à la section suivante.

6.1.2.3 Impact de l'utilisation des AO

Selon Mayer, il existe des situations dans lesquelles il y a des évidences suffisantes de l'effet positif des AO sur l'apprentissage. Par exemple, si le sujet n'avait pas, d'une autre façon, eu accès à une information préalablement assimilée (i.e. : manque de connaissances) ou s'il ne l'aurait pas naturellement sollicitée (i.e. : connaissance

existante mais non utilisée pour l'assimilation). En d'autres mots, les AO peuvent influencer l'encodage de l'information en :

- fournissant une nouvelle organisation plus générale comme contexte d'assimilation qui n'aurait pas été présentée normalement;

- activant une structure générale de savoir existant qui n'aurait pas été normalement sollicitée afin d'assimiler l'extrait.

Les meilleurs résultats sont obtenus lorsque le matériel est non-familier, technique ou qu'il est difficile pour le lecteur de relier l'information aux connaissances déjà acquises.

De plus, l'effet d'un AO sur l'assimilation d'un extrait est relatif aux caractéristiques du lecteur ainsi qu'à l'extrait donné. En effet, un lecteur qui utilise une stratégie efficace afin d'utiliser ses connaissances acquises durant la lecture ou un professionnel qui maîtrise le domaine pourrait trouver peu d'avantages à l'utilisation des AO, contrairement à un novice ou à un lecteur moins expérimenté.

6.1.2.4 Discussion

Avec les informations données, il est clair que l'élément *MaStr* peut être considéré comme un AO. La différence importante avec les travaux de Mayer consiste essentiellement dans le fait qu'avec le SMML on utilise la notion d'AO à un niveau micro. Dans le cas du SMML, l'AO permet d'introduire le contenu de quelques phrases, alors que dans les expériences de Mayer l'AO est un extrait de 10 pages qui est donné avant un texte entier. Par conséquent, un document SMML contient donc autant d'AO qu'il y a d'éléments *MaStr* dans le texte.

Néanmoins, étant donné que les fonctions sont les mêmes, on peut supposer des effets semblables. Suite aux recherches de Mayer, il apparaît que les AO seraient surtout efficaces au niveau de l'apprentissage ou lors des premiers contacts avec un document. De plus, l'efficacité ou l'influence de telles structures peut être relative d'une personne à l'autre, elle est fonction des caractéristiques individuelles. L'efficacité est aussi fonction des caractéristiques du document, notamment, sa cohérence et la bonne utilisation des macrostructures.

6.2 Vers une validation pratique du modèle

Que reste-t-il à faire comme travail afin de pouvoir quantifier les bénéfices réels de l'utilisation du SMML? En fait, avec tout ce qui a été exposé jusqu'à présent, il semble que le SMML devra être implanté dans un éditeur supportant l'édition structurée (SGML ou autres) afin d'être expérimenté pratiquement. L'efficacité du modèle dépend de plusieurs facteurs qui ne peuvent pas être évalués du fait que le SMML est pour le moment un concept théorique qui n'a pas été expérimenté.

Notamment, il faut définir un catalogue de macrostructures. On a suggéré un support théorique à son développement mais il existe probablement des règles pour l'édification de nouvelles macrostructures afin d'en maximiser l'impact positif, ce qui n'a pas été discuté ici. Il est évident que la qualité du catalogue influencera l'efficacité du SMML.

De plus, on a supposé qu'il serait souhaitable de concevoir un langage SMML spécifique pour chaque type de document. Ceci constitue une nouvelle avenue qui devrait être explorée avant de mesurer l'impact réel du modèle SMML.

6.3 Résumé du chapitre

Dans ce chapitre, on a fait une validation théorique du modèle SMML. Selon les théories de Van Dijk et de Mayer sur les Advance Organizer (AO). Selon les travaux de Van Dijk, il est raisonnable de croire que le modèle SMML, s'il est utilisé correctement, est en mesure de faciliter la tâche compréhension d'un extrait textuel. Notamment, en lui spécifiant systématiquement le thème de l'extrait courant, lui évitant ainsi de faire des hypothèses sur celui-ci. On allège donc la tâche cognitive de création des macrofaits (moins de travail, d'erreurs, de retours arrières afin de trouver le thème), ce qui permet au lecteur de se concentrer sur le contenu d'informations.

On a par la suite démontré que l'élément *MaStr* répond à la définition d'un Advance Organizer (AO). Il apparaît que la structure serait surtout efficace au niveau de l'apprentissage ou lors des premiers contacts avec un document. De plus, l'efficacité ou l'influence d'une telle structure peut être relative d'une personne à l'autre, elle est fonction des caractéristiques individuelles et des caractéristiques du document.

Considérant tout ceci on peut s'attendre, en théorie, à ce que le SMML puisse améliorer l'efficacité du canal de communication. D'autant plus que les ingénieurs sont très souvent confrontés à du matériel nouveau qui doit être lu, compris et validé rapidement afin de poursuivre avec le document suivant.

Dans le chapitre suivant on présente une application pratique du SMML ainsi qu'un exemple de définition d'un catalogue et d'une structure spécifique nommé SMML-AR.

CHAPITRE VII –

APPLICATION PRATIQUE DU SMML ET ÉTUDE COMPARATIVE

L'objectif de ce chapitre est de présenter un exemple de document SMML, de catalogue de macrostructures et d'un modèle spécifique, nommé SMML-AR.

Les résultats de ce chapitre sont basés sur l'analyse de l'article de Robillard (1999) « The Role of Knowledge in Software Development» paru dans la revue « Communication of the ACM » en janvier 1999.

Dans un premier temps, on expose certaines considérations quant au choix de l'article ainsi qu'à la validité des résultats obtenus étant donné la démarche utilisée. Par la suite, on présente le catalogue de macrostructures qui a été développé sur les bases de l'article. On fait une analyse comparative de la version SMML et de la version conventionnelle de l'article afin de relever les points forts et les coûts de l'utilisation du SMML. Finalement, on présente le modèle SMML-AR pour l'édition d'articles scientifiques.

7.1 Contexte et rationnel de l'approche

Les travaux effectués et les résultats présentés dans ce chapitre reposent sur l'analyse d'un seul article, soit « The Role of Knowledge in Software Development » (Robillard,

1999), qui est présenté en annexe IV. Ce texte a été choisi préférablement à d'autres pour les des raisons suivantes :

- Le texte est de qualité compte tenu qu'il s'agit d'une publication officielle. Son organisation, sa structure interne et son contenu ont été révisés et corrigés par des professionnels du milieu. Il se démarque donc sur ces points comparativement à d'autres documents non-officiels, comme des documents internes d'un projet par exemple.

- La structure du texte est simple et régulière tout en étant suffisamment complexe, permettant d'obtenir les résultats escomptés. Le texte choisi présente cinq concepts les uns à la suite des autres en les discutant individuellement. Il est parsemé d'exemples et de définitions simples. On peut donc définir certaines classes de macrostructures basées sur les éléments récurents du texte.

Pour réaliser le document SMML (document cible), on a utilisé un processus de réingénierie de type opportuniste consistant essentiellement à analyser le document conventionnel (document source) afin de reconstituer au mieux sa structure sémantique. Le document SMML obtenu est présentés à l'annexe V. Cette analyse a permis de définir un ensemble de classes de macrostructures compatibles avec à la structure sémantique du document source. À l'aide des classes de macrostructures définies, on a réalisé le catalogue. Finalement, on a défini le modèle SMML-AR qui permet de reproduire la structure sémantique de l'article analysée.

Il est à noter que l'utilisation du SMML dans un processus de réingénierie n'est pas usuel. En effet, le SMML a été présenté avant tout pour être utilisé lors de la conception d'un texte et non a posteriori. Il doit permettre à l'auteur de saisir la structure sémantique de son texte lors de l'édition. La structure sémantique n'est

qu'implicitement conservée dans les textes conventionnels. Il s'en suit que sa reconstruction à partir d'un texte déjà écrit est un processus pouvant demander l'établissement de certaines hypothèses et est, par conséquent, sujet à l'erreur. Il est impossible de vérifier la conformité de la structure reconstituée avec celle initialement formulée par l'auteur.

De plus, selon Bernhardt (1986), la rhétorique utilisée change profondément au niveau de la composition et de l'organisation du discours en fonction du degré d'information qui est transmis visuellement (voir la section 4.4.4 pour plus de détails). On peut donc s'attendre a ce que les auteurs changent leur technique de rédaction. Il s'en suit que l'article aurait probablement eu une structure et une formulation différentes s'il avait été édité avec le SMML. Par conséquent, le document SMML réalisé à partir de l'article est probablement différent du document qu'on aurait obtenu si l'auteur avait utilisé le SMML lors de la rédaction.

Toutefois, étant donné que les objectifs fixés consistent essentiellement à montrer des exemples d'application d'un document SMML, d'un catalogue et d'une structure spécifique, les résultats présentés restent globalement valides. Quant à l'analyse comparative, les observations touchent essentiellement à la structure et non au contenu d'information du document.

7.2 Définition d'un catalogue de macrostructures

On présente maintenant le catalogue de macrostructures. Celui-ci comporte treize classes de macrostructures que l'on retrouve dans le document SMML de l'article (Annexe V). Chacune des classes respecte le format présenté à la Figure 4.6.

On rappelle que les entités du catalogue représentent les éléments d'un langage grâce auquel il est possible de communiquer de l'information. Par conséquent, il est essentiel de connaître le contenu du catalogue avant de lire un document SMML. Le lecteur sera alors en mesure de décoder l'information qui est véhiculée via les structures graphiques. Il comprendra la signification d'une macrostructure de classe *Definition* ou *Constituant* et connaîtra le type d'information qui doit y être associé ainsi que l'organisation structurelle de la classe. Il saura donc qu'une classe *Definition* contient la définition d'un concept et qu'une classe *Constituant* peut être composée de classes *Notion* et *Discussion*.

7.2.1 Patrons de SuperStructures

Le catalogue contient trois classes de superstructures. Celles-ci représentent la structure conventionnelle d'un texte, c'est-à-dire l'introduction, le développement et la conclusion. Les classes sont présentées en ordre alphabétique.

Conclusion

Type

Article. SuperStructure.

Définition

Un élément de classe *Conclusion* contient la fin de l'article dans laquelle l'auteur reprend les notions importantes en les concluant.

Structure

Élément composé d'éléments de classe *Terminaison* et d'autres classes de type *MacroStructure* . La déclaration SGML en définit la structure :

```
<!-- Définition de l'élément Conclusion >
<!ELEMENT Conclusion - - (Theme, (Terminaison|MaStr)+)>
```

Exemple d 'utilisation

Non disponible.

Macrostructures associées

Dans la définition d'un article, l'élément *Conclusion* est normalement précédé des éléments *Introduction* et *Developpement.*

Terminaison : Élément constituant.

Developpement

Type

Article, SuperStructure.

Définition

Un élément de classe *Developpement* contient le corps de l'article.

Structure

Élément composé, pouvant contenir sans restriction toutes les classes du catalogue de type *MacroStructure*.

Exemple d 'utilisation

Non disponible.

Macrostructures associées

Dans la définition d'un article, l'élément *Developpement* est normalement précédé d'un élément *Introduction* et suivi d'un élément *Conclusion*.

Introduction

Type

Article, SuperStructure.

Définition

Un élément de classe *Introduction* contient la première partie de l'article dans laquelle l'auteur introduit le sujet traité et expose le but et les objectifs de son discours.

Structure

Élément composé d'éléments de classe *But, Objectif* et d'autres classes de type *MacroStructure* . La déclaration SGML en définit la structure :

```
<!-- Définition de l'élément Introduction >
<!ELEMENT Introduction - - (Theme, (Objectif|But|MaStr)+)>
```

Exemple d 'utilisation

Non disponible.

Macrostructures associées

Dans la définition d'un article, l'élément *Introduction* est normalement suivi des éléments *Developpement* et *Conclusion*.

But : Élément constituant.

Objectif : Élément constituant.

7.2.2 Patrons de MacroStructures

Le catalogue contient dix classes de macrostructures. Les classes *But* et *Objectif* sont des classes simples présentes dans l'introduction d'un texte. Les classes *Definition*, *Exemplification* et *Figure* sont des classes simples représentant des définitions de concepts, des exemples et des figures. La classe *Liant* représente les entités sémantiques d'un texte qui ne sont pas catégorisées dans les autres classes. La classe *Terminaison* est présente dans la conclusion d'un texte et contient une conclusion sur une notion du texte introduite par l'auteur.

La classe *Constituant* est une classe composée des classes *Notion* et *Discussion*. Elle permet de représenter les sujets importants traités dans le texte qui implique souvent la présentation de théories (classe *Notion*) ainsi que d'explications, d'observations et de réflexions de l'auteur (classe *Discussion*).

But

Type

Article, MacroStructure.

Définition

Un élément de classe *But* représente le but qui est visé par l'auteur par la rédaction de l'article. La classe se compose essentiellement d'éléments de microstructure.

Structure

La déclaration SGML suivante permet d'expliciter la structure de l'élément *But* en définissant sa structure :

```
<!-- Définition de l'élément But >
<!ELEMENT But - - (Theme, (MiStr)+)>
```

Exemple d 'utilisation

Non disponible.

Macrostructure associées

Introduction : Élément englobant.

Objectif : Élément associé.

Constituant

Type

Article, MacroStructure.

Définition

Un élément de classe *Constituant* contient un sujet important qui est traité dans un article.

Structure

Élément composé d'éléments de classe *Notion* et de classe *Discussion*. La déclaration SGML en définit la structure :

```
<!-- Définition de l'élément Constituant >
<!ELEMENT Constituant - - (Theme, (Notion|Discussion)+)>
```

Exemple d 'utilisation

L'article *The Role of Knowledge in Software Development* contient cinq éléments de classe *Constituant*. Ceux-ci représentent les divers concepts abordés soit : *Procedural/Declarative Knowledge*, *Schema*, *Proposition*, *Chunking* et *Planning*.

Macrostructures associées

Notion : Élément constituant .

Discussion : Élément constituant.

Definition

Type

Article, MacroStructure.

Définition

Un élément de classe *Definition* représente une définition selon le sens usuel du terme. Il s'agit d'une classe élémentaire de macrostructure. Elle se compose essentiellement d'éléments de microstructure comme du texte ou des images.

Structure

La déclaration SGML suivante permet d'expliciter la structure de l'élément *Definition* en définissant sa structure :

```
<!-- Définition de l'élément Definition >
<!ELEMENT Definition - - (Theme, MiStr*)>
```

Exemple d 'utilisation

L'article *The Role of Knowledge in Software Development (Robillard, 1999)* contient plusieurs éléments de classe *Definition*. La Figure 7.1 montre la définition du savoir telle que présentée dans l'article.

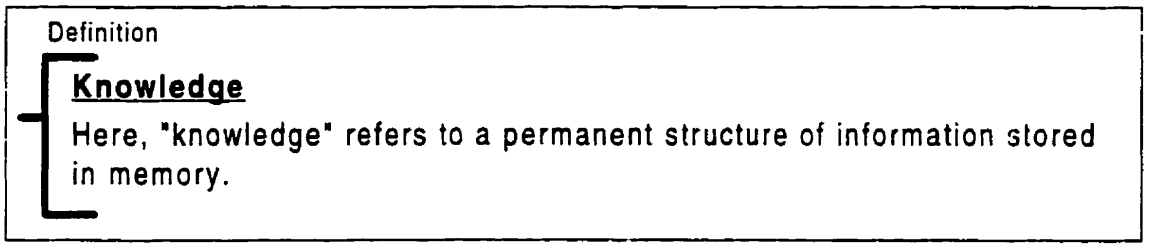


Figure 7.1 – La définition du savoir (Robillard, 1999)

Macrostructure associées

Aucune.

Discussion

Type

Article, MacroStructure.

Définition

Un élément de classe *Discussion* peut représenter les observations, les conclusions, les réflexions et les considérations de l'auteur sur un sujet. L'élément *Discussion* englobe par conséquent le matériel nouveau qui est introduit dans l'article par l'auteur.

Structure

Élément qui compose un *Constituant*. Celui-ci peut contenir sans restriction toutes les classes du catalogue de type *MacroStructure*.

Exemple d 'utilisation

L'article *The Role of Knowledge in Software Development* contient plusieurs éléments de classe *Discussion*. Ceux-ci sont toujours englobés par un élément de classe *Constituant*. Cette classe a été utilisée afin d'expliquer les conclusions de l'auteur sur le sujet des spécifications formelles et des types de problèmes. Cette discussion a été introduite suite à la présentation des notions (élément *Notion*) sur les problèmes bien définis et mal définis.

Macrostructures associées

Constituant : Élément englobant .

Notion : Élément associé.

Exemplification

Type

Article, MacroStructure.

Définition

Un élément de classe *Exemplification* représente un exemple selon le sens usuel du terme. Il s'agit d'une classe élémentaire de macrostructure. Elle se compose essentiellement d'éléments de microstructure comme du texte ou des images.

Structure

```
<!-- Définition de l'élément Definition >
<!ELEMENT Exemplification - - (Theme, MiStr*)>
```

Exemple d 'utilisation

L'article *The Role of Knowledge in Software Development* contient plusieurs éléments de classe *Definition*. La Figure 7.2 montre un élément *Exemplification*.

Exemplification

Scenario of schema

The power of the schemas and their default values are illustrated by the following scenario (which includes many schemas):

Daniel opens his computer, and edits his homework while listening to music by Mozart. He e-mails one chapter to his classmate for revision, then plays a game called ABC while waiting for her reply.

Your understand this scenario based on your schemas, which are filled with default values based on your personal topic and episodic knowledge related to them. For example, the user's schema of a computer has a default value which corresponding to the operating system for the editing schema; the default value could be any one of the many text editors (WORD, WORDPERFECT, FRAMEWORD, and LaTex). Your schema of Mozart's music could be just some classical music or specific Mozart masterpieces. However, you are likely to lack a default value for the schema of the electronic game called ABC.

Figure 7.2 – Exemple d'utilisation d'une classe *Exemplification* (Robillard, 1999)

Macrostructure associées

Aucune.

Figure

Type

Article, MacroStructure.

Définition

Un élément de classe *Figure* représente une figure, un graphique ou un tableau ainsi que le texte de présentation et d'explication qui lui est relié. Il s'agit d'une classe élémentaire de macrostructure. Elle se compose essentiellement d'éléments de microstructure comme du texte ou des images.

Structure

```
<!-- Définition de l'élément Definition >
<!ELEMENT Image - - (Theme,MiStr+)>
```

Exemple d 'utilisation

L'article *The Role of Knowledge in Software Development* contient un élément de classe *Figure*. La Figure 7.3 montre un élément *Figure*.

Figure

Knowledge concept in cognitive sciences

Table 1 list the viewpoint corresponding to each key knowledge concept in the cognitive sciences.

Key knowledge concept	Viewpoint
Procedural/Declarative knowledge	Nature of knowledge content
Schema	Internal structure of knowledge
Proposition	Formal representation of knowledge
Chunking	Representation of knowledge unit
Planning	Management of knowledge structures

Table 1. Key knowledge concepts in the cognitive sciences.

Figure 7.3 – Exemple de Figure (Robillard, 1999)

Macrostructures associées

Aucune.

Liant

Type

Article. MacroStructure.

Définition

Un élément de classe *Liant* permet de représenter un extrait de texte ayant une invariance sémantique constante (thème constant) dont la nature intrinsèque ne peut pas être associée à une autre classe du catalogue. Il s'agit en quelque sorte d'une classe générique qui permet de représenter des éléments non-classés.

Structure

Celui-ci peut contenir sans restriction toutes les classes du catalogue de type *MacroStructure*.

Exemple d 'utilisation

Non disponible.

Macrostructure associées

Aucune.

Notion

Type

Article. MacroStructure.

Définition

Un élément de classe *Notion* représente une théorie introduite et expliquée qui a été développée par un tiers. Celle-ci servira de base pour la discussion ou les observations de l'auteur.

Structure

Élément qui compose un *Constituant*. Celui-ci peut contenir sans restriction toutes les classes du catalogue de type *MacroStructure*.

Exemple d 'utilisation

L'article *The Role of Knowledge in Software Development* contient plusieurs éléments de classe *Notion*. Ceux-ci sont toujours englobés par un élément de classe *Constituant*. Cette classe a été utilisée afin de présenter certaines notions utiles à l'auteur, par exemple : *Declarative Knowledge* et *Topic Knowledge*. La présentation de ces notions permet à l'auteur d'expliquer leur rôle dans le développement de logiciels.

Macrostructures associées

Constituant : Élément englobant.

Discussion : Élément associé.

Objectif

Type

Article, MacroStructure.

Définition

Un élément de classe *Objectif* représente les objectifs de l'article au sens usuel du terme. Cette classe est uniquement présente dans l'introduction du texte. Elle se compose essentiellement d'éléments de microstructure.

Structure

La déclaration SGML suivante permet d'expliciter la structure de l'élément *Objectif* en définissant sa structure :

```
<!-- Définition de l'élément Objectif >
<!ELEMENT Objectif - - (Theme, MiStr+)>
```

Exemple d 'utilisation

Non disponible.

Macrostructures associées

Introduction : Élément englobant .

But : Élément associé.

Terminaison

Type

Article, MacroStructure.

Définition

Un élément de classe *Terminaison* est présent dans la conclusion d'un article. Il permet de présenter le rappel des observations et des réflexions de l'auteur concernant un sujet qui a été traité dans le développement. Cet élément permet à l'auteur de conclure chacun des sujets en particulier.

Structure

Élément qui compose une classe *Conclusion*. Celui-ci peut contenir sans restriction toutes les classes du catalogue de type *MacroStructure*.

Exemple d 'utilisation

Non disponible.

Macrostructures associées

Aucune.

7.3 Analyse comparative : Avantage et coût de l'utilisation du SMML

On présente maintenant une analyse basée principalement sur la comparaison du texte conventionnel (texte source) et du texte SMML (texte cible). Cette analyse a pour objectif de mettre en évidence certains des avantages du SMML et des coûts reliés à son utilisation.

La structure sémantique du document est explicitée

Le texte SMML (document cible) permet d'illustrer clairement la structure sémantique du discours. Cette structure sémantique n'est qu'implicitement conservée et représentée dans le texte conventionnel. Par conséquent, le lecteur doit lire le document conventionnel d'une façon plus ou moins complète afin de saisir son organisation sémantique. La Figure 7.4 illustre une partie de la structure sémantique du texte SMML.

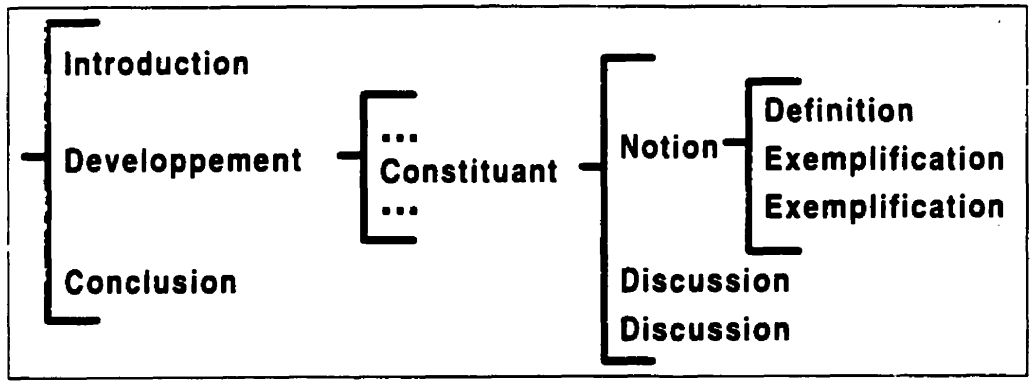


Figure 7.4 – Extrait de la structure sémantique du document cible

La figure est constituée des classes de macrostructures qui sont instanciées dans le document SMML. On note la présence des trois classes de superstructures :

Introduction, *Developpement* et *Conclusion*. La classe *Constituant* et les sous-classes qui la composent représentent la structure sémantique de l'extrait textuel où l'auteur présente le sujet des schémas.

Bien que la figure soit différente visuellement du document SMML, il s'agit du même document. En effet, comme on utilise la technologie SGML, il suffit de faire une vue[43] sur le document SMML dans laquelle on présente hiérarchiquement les classes de macrostructures en ne montrant pas le reste des informations du texte (thème, rhème, etc.).

Les éléments sémantiques du texte sont délimités visuellement

Dans un texte conventionnel, l'auteur a recours à la section et aux paragraphes afin de délimiter ou de séparer les éléments du texte. Cependant, ces outils d'organisation structurelle peuvent être mal adaptés pour faire ressortir la structure de certains extraits textuels. À titre d'exemple, les Figure 7.5 et la Figure 7.6 représentent le même extrait textuel constituant un seul paragraphe dans le texte conventionnel.

[43] Voir la Section 2.1.6 sur les coûts et les bénéfices à l'utilisation du SGML pour plus de détails sur les vues.

The mental processing and representation of knowledge are complex activities, and our understanding is still rudimentary and subject to debate [10]. A general concept to describe knowledge is as elusive as ever, though various key concepts been developed from specific viewpoints in the cognitive sciences. Some of theme are derived from the content or structure of knowledge, others from its representation. Here, "knowledge" refers to a permanent structure of information stored in memory. "Knowledge representation" refers to a transitory construction built up in memory for the processing of a specific situation.

Figure 7.5 – Extrait textuel du document source

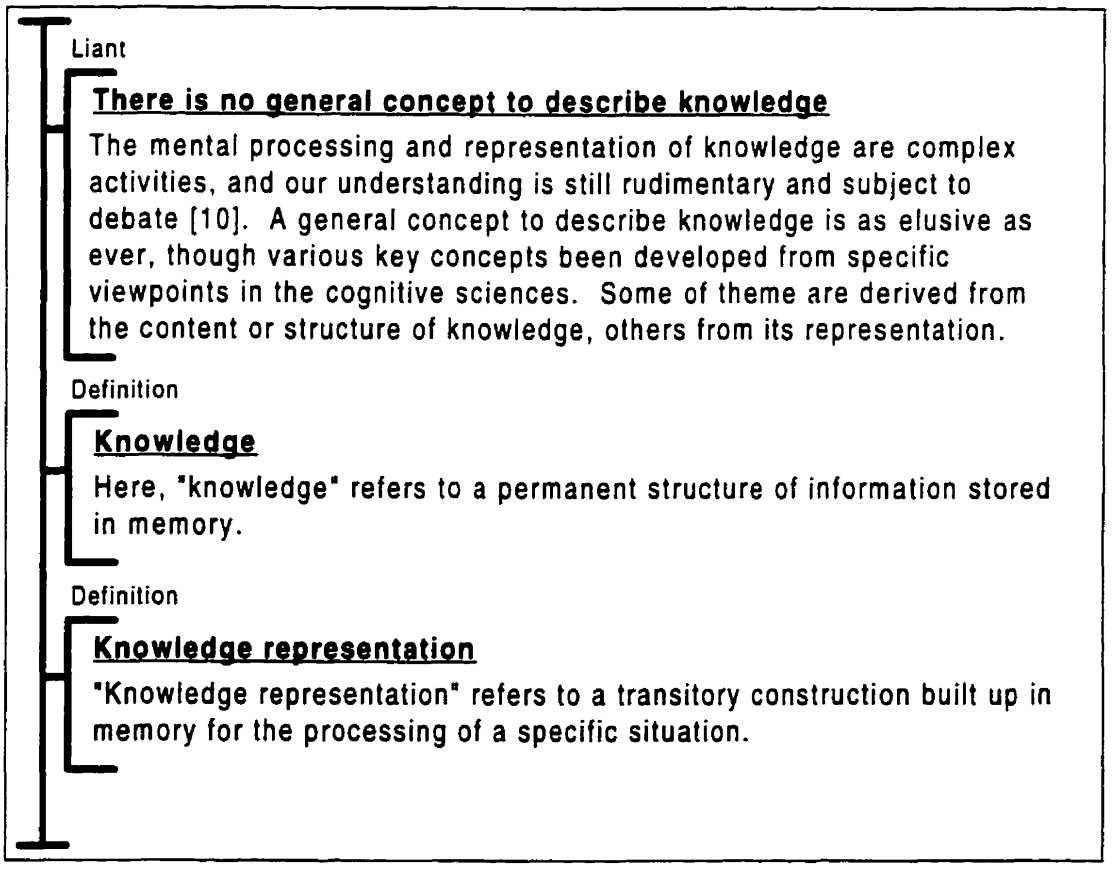


Figure 7.6 – Extrait textuel du document SMML

La seconde figure permet de voir immédiatement la présence de deux définitions dans le paragraphe. Celles-ci ne sont pas délimitées ou mises en évidence visuellement dans le texte conventionnel. Par conséquent, le lecteur doit lire le paragraphe en entier afin de se rendre compte qu'il contient deux définitions. La macrostructure semble offrir une granularité plus fine que le paragraphe ce qui s'avère utile dans l'exemple présenté.

Validation possible du contenu du discours à l'aide du catalogue

Le catalogue fournit une définition pour chacune des classes de macrostructures qui peut être utilisée dans un type de texte. Lors d'une révision du document, ces définitions peuvent être utilisées afin de valider le contenu d'information des macrostructures du texte. Par exemple, dans le cas d'une macrostructure de classe *Notion*, on peut vérifier que le contenu d'information est cohérent avec le thème et la classe de celle-ci. On peut donc détecter une erreur de mauvaise utilisation d'une classe de macrostructures si l'auteur utilise une classe *Notion* afin de communiquer ses réflexions sur un sujet au lieu d'introduire des théories d'un tiers. L'analogie la plus significative consiste à considérer le texte SMML comme un formulaire dans lequel on vérifie si les informations sont valides et inscrites aux bons endroits. La validation à l'aide du catalogue devient plus intéressante encore lorsque le document est plus technique avec une structure SMML plus complexe, comme par exemple un document de design.

Traitements évolués sur le document SMML

Étant donné que le SMML est basé sur le SGML et qu'on identifie certains éléments du document, il est possible de faire des recherches selon les types d'éléments ou de

composer des documents automatiquement (document déduit)[44]. Par exemple, la liste hiérarchique des classes (Figure 7.4) permet de visualiser l'organisation structurelle du document. La liste hiérarchique des thèmes d'un document permet d'obtenir un condensé du contenu en information. De plus, il est possible de composer une liste de définitions du texte. Dans l'exemple, on obtiendrait la liste des neuf définitions suivantes : « Knowledge », « Knowledge Representation», « Procedural Knowledge », « Declarative Knowledge », « Topic Knowledge », « Episodic Knowledge », « Schema » et « Plans ». Cette information ne peut pas être automatiquement récupérée dans le document conventionnel.

Considérations sur l'utilisation du SMML

Un point important à considérer concerne la justesse et l'utilisation judicieuse des macrostructures afin de représenter le plus efficacement possible le contenu sémantique d'un texte. Il serait possible de définir beaucoup plus de macrostructures dans le texte cible et ainsi obtenir une structure sémantique avec une granularité plus fine. On peut cependant s'attendre à ce que l'utilisation d'une structure avec une granularité trop fine alourdisse inutilement un texte et nuise éventuellement à la compréhension de celui-ci. À l'opposé, on peut s'interroger sur les avantages d'utiliser une structure non suffisamment détaillée. Le contenu d'information communiqué par les macrostructures est alors faible. Les règles afin de définir une structure SMML optimale constituent un problème ouvert qui n'est pas abordé ici.

Qu'en est-il de la surcharge de travail associée à l'utilisation du SMML? Il est actuellement impossible de répondre directement à cette question étant donné l'absence

---

[44] Voir la Section 2.1.5 sur le SGML et les types d'interactions pour plus de détails sur les traitements évolués.

de données sur le sujet. On sait cependant que l'auteur doit définir les classes de macrostructures qu'il désire utiliser dans le cas où celles-ci n'existent pas. De plus, il doit, lors de l'édition, identifier la classe de macrostructures adéquate pour le contenu d'information qu'il désire exprimer.

Dans le même ordre d'idée, lors de la rédaction d'un texte, l'auteur doit dans un premier temps penser à l'organisation structurelle de son discours afin de déterminer les sujets qui seront abordés ainsi que leurs articulations dans le texte. Le SMML peut le supporter dans cette tâche. On peut faire une analogie entre le SMML et le Pseudocode schématique (Robillard, 1995). Le Pseudocode schématique, avec la notion de raffinement successif qu'il implante, permet de concevoir l'algorithme d'un programme. Celui-ci exprime la structure et l'organisation du programme tout comme le SMML exprime la structure sémantique d'un texte. Une fois l'algorithme conçu et validé, on peut réaliser le programme et on s'assure ainsi d'avoir un programme qui soit bien structuré. La même approche pourrait probablement s'appliquer avec le SMML.

Finalement, la représentation graphique actuelle semble plus appropriée pour un environnement informatique comme un éditeur SMML que pour un environnement conventionnel comme une représentation sur papier. Notamment, il peut être difficile de saisir rapidement la hiérarchie d'un document SMML volumineux représenté sur papier. Par exemple, si une définition est à une profondeur importante dans le document elle sera précédée de plusieurs lignes pouvant s'étendre sur plusieurs pages. Il peut être difficile de différencier une ligne du lot en particulier. Ce problème peut être facilement résolu dans un environnement informatique. Il suffit de définir une vue permettant de voir l'arbre thématique du document dans laquelle on indique clairement la position du curseur. Le lecteur pourra alors facilement identifier sa position.

7.4 Définition d'un modèle spécifique pour les articles scientifiques : le modèle SMML-AR

On présente maintenant le modèle SMML-AR pour la rédaction des articles scientifiques. Celui-ci regroupe et organise toutes les classes définies dans le catalogue de macrostructures. De plus, le modèle doit pouvoir représenter la structure de tous les documents qui ont servi à la construction du catalogue (voir la section 4.4.3 - Limites et aspects complémentaires à l'utilisation du modèle SMML et du catalogue pour plus de détails). La Figure 7.7 présente la définition SGML du modèle SMML-AR.

```
<!--0 DTD du SMML_AR(Semantic Macrostructure Markup Language)>
<!--1> <!ELEMENT SMML_AR - - GlStr>
<!--2> <!ELEMENT GlStr - -
                  (Introduction, Developpement, Conclusion)>
<!--3> <!ELEMENT Introduction - -
                  (Theme,(But|Objectif|MaStrCmp|MaStr)+)>
<!--4> <!ELEMENT But - - (Theme,MiStr)>
<!--5> <!ELEMENT Objectif - - (Theme,MiStr)>

<!--6> <!ELEMENT Developpement - - (Theme, (MaStrCmp|MaStr)+>
<!--7> <!ELEMENT Conclusion - -
                  (Theme, (Terminaison|MaStrCmp|MaStr)+)>
<!--8> <!ELEMENT Terminaison - - (Theme, (MaStrCmp|MaStr)+>

<!--9 ><!ELEMENT MaStrCmp - -(Constituant|Liant)>
<!--10><!ELEMENT Constituant - - (Theme,(Notion|Discussion)+)>
<!--11><!ELEMENT Liant - - (Theme,(MaStrCmp|MaStr)+)>
<!--12><!ELEMENT Notion - - (Theme,(MaStrCmp|MaStr)+)>
<!--13><!ELEMENT Discussion - - (Theme,(MaStrCmp|MaStr)+)>

<!--14><!ELEMENT MaStr - -(Definition|Exemplification|Figure|MiStr)>
<!--15><!ELEMENT Definition - - (Theme,MiStr)>
<!--16><!ELEMENT Exemplification - - (Theme,MiStr)>
<!--17><!ELEMENT Figure - - (Theme,MiStr)>


<!--18><!ELEMENT Theme - - (#PCDATA)>
<!--19><!ELEMENT MiStr - - (texte|image|graphique|hyperlien)+>
```

Figure 7.7 – Définition du modèle SMML-AR

On remarque que la structure globale (ligne 2, élément *GlStr*) est composée des éléments de superstructure *Introduction, Developpement* et *Conclusion.* Ceux-ci sont notamment composés (ligne 3,6,7) d'un thème et d'une série mixte de macrostructures composées (élément *MaStrCmp*) et de macrostructures simples (élément *MaStr*). L'élément *Introduction* (ligne 3) peut aussi contenir des éléments *But* et *Objectif.* L'élément *Conclusion* (ligne 7) peut aussi contenir des éléments *Terminaison.* Les éléments *But, Objectif,* et *Terminaison*, sont composés essentiellement d'éléments de microstructures (ligne 4,5,8).

On considère les éléments *Constituant* et *Liant* (ligne7) pour les classes de macrostructures composées. Conformément à ce qui a été défini dans le catalogue, l'élément *Constituant* (ligne 8) est composé d'une liste mixte d'éléments *Notion* et *Discussion*. Les éléments composés peuvent contenir d'autres éléments composés ou des éléments simples (ligne 9 à 11). Les élément *Definition*, *Exemplification* et *Figure* sont considérés comme étant les éléments de macrostructures simples (ligne 12). Ils sont composés essentiellement d'éléments de microstructures.

On rappelle que la création d'un modèle spécifique pour un type de documents est souhaitable lorsqu'on considère que le catalogue est complet. Il s'en suit qu'il faudrait idéalement considérer beaucoup plus qu'un seul article.

7.5 Résumé du chapitre

Ce chapitre a permis essentiellement de présenter un exemple de document SMML et d'en faire une analyse comparative. On a d'ailleurs construit sur cet exemple un catalogue de macrostructures ainsi que la structure spécifique SMML-AR pour la rédaction d'articles scientifiques.

Le processus de réingénierie employé n'est pas conforme à l'utilisation usuele du SMML. Cependant, étant donné que l'objectif principal du chapitre est de montrer un exemple pratique, les résultats restent valides.

CONCLUSION

Ce mémoire est avant tout un travail de recherche théorique de nature multidisciplinaire. Il met en valeur des théories issues du domaine de l'analyse de discours, de la cognition et de l'informatique. Il s'agit essentiellement d'un transfert et d'une adaptation, pour de nouvelles fins, de théories du domaine de l'analyse de textes au génie logiciel. Il y a beaucoup à gagner à faire ce type de mariage.

Les présentes recherches ont pour sujet les formats électroniques de document. Bien que ce sujet appartient à un domaine commun et relativement accessible, soit celui de l'édition, elles ont permis de mettre en relief la problématique de l'efficacité de la communication et proposent une solution simple et originale.

Le modèle SMML (Semantic Macrostructure Markup Language) est défini à l'aide du SGML qui a été présenté au chapitre II. On a aussi présenté dans ce chapitre un répertoire des formats électroniques existants. Il est clair que ceux-ci comportent des limites importantes au niveau du traitement informatique. Le SMML respecte la définition du modèle OHCO qui a été proposée par DeRose et al. afin de repousser ces limites de traitement. Un apport fondamental au présent travail consiste en l'explication de DeRose sur les multiples hiérarchies possibles d'un document. Le SMML est donc une hiérarchie possible, parmi les autres, qui permet de relever la structure sémantique d'un document contrairement à sa structure de mise en page.

Le modèle est basé sur les travaux de Teun A. Van Dijk concernant les macrostructures, qui ont été présentés au chapitre III. Initialement, la macrostructure est un concept outil pour l'analyse de discours qui permet de faire le lien entre la structure d'un texte et sa représentation cognitive. Son adaptation au SMML permet d'utiliser le concept de macrostructure à de nouvelles fins, soit pour la formalisation de l'information et la communication.

Au chapitre IV, on a présenté le modèle SMML. Notamment, on a défini l'élément *MaStr* de telle sorte que la structure globale de Van Dijk soit intégrée directement avec le texte. On a introduit la notion de classes de macrostructure. Celle-ci fournit un degré d'abstraction intéressant et ouvre la porte à la définition de traitements informatiques évolués. On a proposé l'utilisation des patrons pour la définition d'un catalogue de macrostructures, étant donné qu'il n'existe pas d'inventaire actuellement disponible.

On propose une façon de représenter graphiquement un document SMML à l'écran ou sur papier au Chapitre V. Cette proposition tient compte des travaux de Bernardt. Elle permet notamment d'expliciter la structure hiérarchique et de délimiter graphiquement les macrostructures dans le texte.

On a fait une validation théorique du modèle SMML au chapitre VI, à l'aide des travaux de Van Dijk sur la macrostructure et la cognition. Il est ressorti que le modèle SMML devrait permettre d'alléger la tâche de compréhension du lecteur et donc de réduire le coût d'utilisation du canal de communication qu'est le document.

On a poussé plus loin la validation théorique afin de raffiner les attentes quant aux qualités cognitives du SMML, en démontrant que la macrostructure respectait la notion d'Advance Organizer (AO). Les recherches faites sur les AO montrent que les

bénéfices de l'utilisation d'une telle construction peuvent varier en fonction de la connaissance, de l'expertise et de la technique de lecture du lecteur.

Finalement on a présenté dans le chapitre VII un exemple pratique d'un document SMML, d'un catalogue de macrostructures et d'une structure spécifique nommée SMML-AR pour l'édition d'articles scientifiques. On a aussi présenté une analyse comparative afin de montrer certains avantages reliés à l'utilisation du SMML.

La poursuite des recherches devrait toucher les aspects suivants :

- construction du catalogue;
- construction d'un environnement informatique d'édition pour supporter le SMML et le catalogue;
- définition de modèles SMML pour des types de document précis, par exemple, le modèle SMML-RQ pour les documents de requis, etc.;
- validation pratique du modèle;
- implication pour l'auteur de l'utilisation du SMML;
- problématique de traduction des documents et standards existants.

Considérant le travail qui a été fait et les résultats obtenus, on peut supposer que le SMML trouvera sa place parmi les outils informatiques de l'ingénieur et, en fait, pour toute personne dont l'objectif est de communiquer le plus efficacement possible une

information technique via un document, plutôt que de charmer l'oeil du lecteur par la présentation de celui-ci.

BIBLIOGRAPHIE

BASILI V.R., GREEN S., LAITENBERGER O., LANUBILE F., SHULL F., ORUMGARD S., AND ZELKOWITZ M.V. (1996). The Empirical Investigation of Perspective-Based Reading, Tech. Rep. ISERN-96-06, University of Maryland.

BERNHARDT, S. A. (1986). Seeing the text. *College Composition and Communication*, Vol. 37, No. 1, February , pp. 66-78.

BERNHARDT. S. A. (1992). Revisions : Seeing the Text, Again. *The Journal of Computer Documentation*, ACM Press, Vol. 16. No 3. pp. 48-54.

BROOKS, F.P. (1987), No Silver Bullet : Essences and Accidents of Software Engineering., *Computer*, Vol. 20(4), 10-19.

BURNARD L. (1995), What Is SGML and How Does It Help?, *Computer and the Humanities*, Vol. 29. pp. 41-50.

CHRISTOPHER A., SARA I., MURRAY S., MAX J., INGRID F. SHLOMO A. (1977), A Pattern Language, Oxford University Press, New York, 1171 p.

COPLIEN O. J. (1994), A Development Process Generative Pattern Language, *Proceedings of PloP/94*, Monticello, Août , 34 p.

DANES, F. (1970), One instance of Prague School Methodology : functional analysis of utterance and text, *Method and theory in linguistics*, P.L. Garvin (ed.), The Hague : Mouton, pp. 132-140.

DANES, F. (1974), Functional Sentence Perspective and the organization of text, *Papers on functional sentence perspective*, F. Danes (ed.), The Hague : Mouton, pp. 106-128

DEROSE, DURAND, MYLONAS, AND RENEAR (1990). What Is Text, Really?. *Journal of Computing in Higher Education*, I (Winter), pp. 3-26.

GAMMA E., HELM R., JOHNSON R., VLISSIDES J., (1997). Design Patterns. Elements of Reusabie Object-Oriented Software. Addison-Wesley, 394 p.

GIRILL, T. R. (1992), The Principles of Visual Rhetoric and the Rhetoric of Visual Principles. *The journal of computer documentation*, ACM Press, Volume 16, no 3, 1992, pp. 17-26.

GOLDFARB C. F. (1990), The SGML HandBook, Clarendon Press, Oxford, 1990, 663 p.

HANEEF N. J. (1998), Software Documentation and Readability : A Proposed Process Improvement. *Software Engineering Notes*, Vol 23 no 3, (Mai). Page 75-77.

HILERA J. R., GONZALEZ L. A., GUTIERREZ J. A., MARTINEZ J.M. (1998). Software Documentation as an Engineering Process. *Software Engineering Notes*, Vol 23, No 5, (Septembre), Page 61-63.

IEEE Standards Collection : Software Engineering (1997). *IEEE Standards Collection.* Institute of Electrical and Electronics Engineers, Inc.

KLINE-SIMON K. (1992), Seeing the Text from the User's Point of View, *The journal of computer documentation*, ACM Press. Vol. 16, no 3, pp. 41-42.

LE E. (1996). Structure discursive comparée d'écrits argumentatifs en français et en anglais. De leur linéarité. *Thèse du département de linguistique et de traduction, Faculté des arts et des sciences, Université de Montréal*, pp. 1-74.

MALER E., EL ANDALOUSSI J. (1996). Developing SGML DTDs. From Text to Model to Markup. Prentice Hall PTR, Upper Saddle River. New Jersey, 532 p.

MAYER R. E. (1985). Structural Analysis of Science Prose : Can We Increase Problem-Solving Performance? *Understanding expository text*, Hillsdalez. pp. 55 - 87.

MAYER R. E. (1979). Can Advance organizers influence meaningful learning? *UReview of Educational Research*, Vol. 45, pp. 371 - 383.

POULLET L. PINON J-M, CALABRETTO S, (1997). Semantic Structuring of Document. *Proceeding of IEEE Conderence on Data Management Systems, BITWIT'97*, Biarritz, pp. 118 – 124.

PRESSMAN R. S. (1997), Software engineering, apractitioner'a approach. The McGraw-Hill Companies, Inc., New York, 852 p.

RENEAR A., MYLONAS E., DURAND D. (1996). Refining our Notion of What Text Really Is : The Problem of Overlapping Hierarchies. *Research in Humanities Computing 4*, Clarendon press, Oxford 1997, pp. 263-280.

ROBERT P. (1985). Le Petit Robert, Dictionnaires LE ROBERT, Paris, 1985, 2171p.

ROBILLARD P. N. (1999), The Role of Knowledge in Software Development, *Communication of the ACM*, Vol. 42, No. 1, pp. 87-92.

ROBILLARD, P.N. (1986). Schematic pseudocode for program constructs and its computer automation by Schemacode. *Communications of the ACM*, 29 (11), 1072-1089.

ROBILLARD P. N. (1985), Le logiciel: de sa conception à sa maintenance, Gaëtan Morin editeur, Chicoutimi, 275 p.

ROBILLARD, P.N. et THALMANN, D. (1980). *Complex problem solving using schematic pseudocode (SPC)*. Publication #373, Université de Montréal, Montréal (Québec).

SOMMERVILLE I.(1995), Software engineering, Addison Wesley, 1995, 741 p.

TRUDEL F. (1995), L'essence et l'objet : Les mécanismes orientés-objet pour la gestion de la complexité logicielle., *Mémoire de maîtrise ès science appliquées du département de génie électrique et de génie informatique, Polytechnique, Université de Montréal*, pp. 139.

VAN DIJK, T.A (1980). Macrostructures (An Interdisciplinary Study of Global Structures in discourse, interaction, and cognition), Lawrence Erlbaum, Hillsdale New Jersey, 317 p.

VIRALLY M.(1966) , Réflexions sur le *jus cogens*, AFDI, 1 : pp. 5-29

ANNEXE I: Van Dijk, Teun : Principales publications en anglais

-Some Aspects of Text Grammars (The Hague: Mouton, 1972)

-Text and Context (London: Longman, 1977)

-Macrostructures (Hillsdale, N.J.: Erlbaum, 1980)

-Studies in the Pragmatics of Discourse (The Hague: Mouton, 1981)

-Strategies of Discourse Comprehension (with W. Kintsch; New York: Academic Press, 1983)

-Prejudice in Discourse (Amsterdam: Benjamins, 1984)

-Discourse and Communication (Ed.)(Berlin: de Gruyter, 1985)

-Handbook of Discourse Analysis (Ed.)(4 vols., London: Academic Press, 1985)

-Communicating Racism (Newbury Park, CA: Sage, 1987)

-News as Discourse (Hillsdale, NJ: Erlbaum, 1988)

-News Analysis (Hillsdale, NJ: Erlbaum, 1988)

-Discourse and Discrimination (Detroit: Wayne State U.P, 1988) (with Geneva Smitherman, Eds.).

-Racism and the Press (London: Routledge, 1991)

-Elite Discourse and Racism (Newbury Park, CA: Sage, 1993).

-Discourse. A Multidisciplinary Introduction. 2 vols. (Ed.). (London: Sage, 1997).

-Discourse and ideology (London: Sage, 1998, in press)

ANNEXE II : Macrostructure : Exemple pratique d'application des macrorègles

Dans cette annexe, on présente la théorie de van Dijk (1980) concernant les macrorègles et leur application. Il est à noter que l'ensemble du matériel présenté ici est extrait (section 1.1.2.3.1 et 1.2.1) de la thèse d'Élisabeth Le (1996) intitulé : « Structure discursive comparée d'écrits argumentatifs en français et en anglais ». On présente en détail les quatre macrorègles ainsi qu'un exemple de leur application.

Présentation des macrorègles

Pour van Dijk, les macrostructures sont des unités sémantiques supérieures à la SVM[45] (ou microstructure), c'est-à-dire que chacune d'entre elles va inclure au moins deux SVM (sauf cas exceptionnel où il y aurait uniquement application de la macrorègle de dérivation zéro).

Ces macrostructures s'obtiennent à la suite de l'application de quatre macrorègles : l'effacement ou la sélection, la généralisation, la construction et la règle de dérivation nulle.

L'effacement (E) permet de ne pas tenir compte de SVM qui ne sont pas pertinentes pour l'interprétation des autres SVM du texte, dans la mesure où elles représentent des faits qui peuvent être compris comme des propriétés normales d'un fait plus global représenté dans le texte.

La sélection (S) fait retenir seulement les SVM pertinentes à l'interprétation d'autres SVM dans le texte, c'est-à-dire qui sont présupposées par d'autres SVM. Cette règle est la formulation positive de la règle de l'effacement.

La généralisation (G) s'opère en laissant de côté les détails sémantiques dans les SVM retenues, de façon à arriver à un niveau conceptuel plus général.

La construction (C), enfin, regroupe en une SVM l'information contenue dans deux ou plus SVM, de façon à ce que la structure ainsi obtenue ne contienne que les éléments de bas, les conditions et les conséquences d'un fait global.

La dérivation nulle (D_0) fait passer directement une SVM au niveau des macrostructures.

Les règles d'effacement, de généralisation et de construction peuvent se définir de façon formelle de la façon suivante (van Dijk, 1980, pp.82-83) :

Effacement : Soit une suite E de propositions $<p_i, p_{i+1}, \ldots p_k>$ d'un texte T, remplissant les conditions de cohérence linéaire.

> E est effacées et remplacées par un suite E', tel que chaque p_{i+j} de E qui n'est pas une condition d'interprétation (n'est pas présupposée) pour au moins une proposition de T, n'apparaisse pas dans E', E et E' étant par ailleurs similaires.

---

[45] Séquence Verbale Maximale : unité de base correspondant à la « phrase » (Le, 1996).

Généralisation : Soit une suite E de propositions $<p_i, p_{i+1}, \ldots, p_k>$ d'un texte T remplissant les conditions de cohérence linéaire.

> E est remplacée par une proposition q, tel que chaque proposition p_{i+j} de E entraîne q et que q soit la plus petite généralisation possible de E.

Construction : Soit une suite E de propositions $<p_i, p_{i+1}, \ldots, p_k>$ d'un texte T remplissant les conditions de cohérence linéaire.

> E est remplacée par une proposition q telle que q entraîne l'ensemble de la suite E dans le même ensemble C, C étant un ensemble de sous-ensembles cognitifs, tes que la connaissance. les croyances, les intérêts, les devoirs, etc..

La formulation des ces règles entraîne plusieurs remarques :

1. Les macrorègles sont des règles de transformation sémantique, qui à des microstructures (ici, des SVM) font correspondre une ou des macrostructures.

2. Il n'y a pas d'équivalence entre les microstructures et la macrostructure qui leur correspond, mais cette dernière « englobe » les premières.

3. Les macrorègles ne peuvent s'appliquer que sur une suite de SVM remplissant les conditions de cohérence.

4. Les macrorègles de généralisation et de construction ne peuvent s'appliquer que sur un ensemble d'au moins deux SVM, et ces dernières ne doivent pas nécessairement être contiguë.

5. L'ordre d'application des règles n'est pas immuable; il dépend des connaissances du monde de la personne qui les applique.

6. Enfin, il n'y a pas de limite fixe d'application aux macrorègles; c'est à la personne qui les applique de choisir où s'arrêter.

Ainsi, si les macrorègles nous permettent de définir des unités sémantiques plus large que la SVM, et à partir desquelles nous pourrions effectuer notre analyse, l'absence de limites claires pour leur application nous rend la notion de macrostructure malaisée à utiliser.

Exemple d'application

De nombreux auteurs ont reconnu l'existence d'une unité d'analyse dont les limites dépasseraient celles de la phrase. Parmi les définitions données, c'est la macrostructrures de van Dijk qui est probablement la plus connue. Elle présente le double avantage d'être intégrée dans un modèle de traitement des informations et d'être définie par l'application de quatre règles.

Considérons l'extrait de l'article de Michel Virally, présenté à la Figure A.1, « Réflexions sur le jus cogens », et plus particulièrement le paragraphe 7 (c'est-à-dire le 7^{e} depuis le début de l'article), et numérotons les SVM constituant ce paragraphe.

7.16 En vue de donner une idée plus frappante de ce que serait le jus cogens, certains auteurs l'ont associé à des notions voisines, empruntées au droit interne, telle que celles d'ordre public, de droit public (au sens étroit du droit romain) ou encore de droit constitutionnel.
7.17 Une telle démarche fait appel au raisonnement par analogie, suivant une habitude constante chez les auteurs lorsqu'ils sont mis en présence d'une notion nouvelle apparue dans le droit international.
7.18 Il est évidemment très tentant, dans une telle circonstance, de se référer à des institutions de droit étatique, qui existent souvent depuis des siècles et dont les traits, par conséquent, ont pu se former avec précisions.
7.19 Dans beaucoup de cas, cette méthode a pu être utilisée avec succès.
7.20 Elle ne peut, cependant, être maniée qu'avec une extrême prudence.
7.21 Dans le cas présent, on peut se demander si les analogies indiquées ci-dessus sont justifiées.
7.22 Il existe, en effet, des différences considérables de situations entre la société internationale et la société étatique
7.23 et ces différences sont particulièrement marquées à propos, précisément, des notions que l'on invoque ici.
7.24 Le droit constitutionnel ne s'est formé dans l'ordre interne que par l'édification d'une structure d'organes extrêmement complexe, qui n'existe évidemment pas dans la société internationale.
7.25 Pour la même raison, les autorités chargées, dans la société étatique, de définir l'ordre public n'ont pas d'équivalent au plan des rapports entre les États.
7.26 Enfin, la distinction du droit public et du droit privé semble avoir une portée très différente suivant qu'on la considère dans la perspective de l'ordre juridique étatique ou dans celle de l'ordre international.

Figure A.1 - Paragraphe 7 de Virally, M « Réflexions sur le jus cogens » (1966)

Soient : E(i,..., n), l'application de la macrorègle d'effacement aux SVM i à n; G(I) : l'application de la macrorègle de généralisation à la SVM I; et C(i, ..., n), l'application de la macrorègle de construction aux SVM i à n. Nous nommerons M(x), la macrostructure du paragraphe x.

Nous obtenons alors :

- par l'application de la règle d'effacement sur les SVM 16 à 26, un ensemble de quatre SVM :

E(16, ..., 26) = (16, 17,19, 20)

En effet, la SVM 18 est présupposée par la SVM 17, car elle en est une explication. Il en est de même avec les SVM 21 à 26, par rapport à la SVM 20. Donc, les SVM 18, 21, 22, 23, 24, 25, 26 sont effacées.

- par l'application de la règle de généralisation (et dérivation zéro) sur les quatre SVM obtenues après effacement, les quatre énoncés suivants :

G(16) = En vue de donner une idée plus frappante de ce que serait le jus cogens, certains auteurs l'ont associé à des notions voisines.

G(17)= Une telle démarche fait appel au raisonnement par analogie.

G(19) = Dans beaucoup de cas, cette méthode a été utilisée avec succès.

G(20) = Elle ne peut, cependant, être maniée qu'avec une extrême prudence.

Dans la SVM 16, la généralisation a eu pour effet de laisser de côté le type de notions voisines auxquelles a été associé le jus cogens, c'est-à-dire une explication; et dans la SVM 17, une comparaison.

Remarquons que G(19) = 19 et G(20) = 20. Il y a donc eu application de la macrorègle de dérivation zéro.

- et enfin, par l'application de la règle de construction sur les quatre énoncés obtenus après généralisation, la macrostructure du paragraphe 7 :

C[G(16), G(17), G(19), G(20)] = M(7)

c'est-à-dire :

M(7) = Le raisonnement par analogie utilisé pour donner une idée du jus cogens, doit être manié avec une extrême prudence, même si cette méthode a souvent été utilisée avec succès.

En effet, l' « association à des notions voisines » (SVM 16) est un « raisonnement par analogie » (SVM 17).

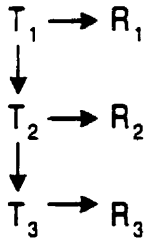
Cet exemple a ainsi montré comment les macrorègles de van Dijk (1980) s'appliquent dans les limites d'un paragraphe typographique.

ANNEXE III : Les progressions thématiques de Danes

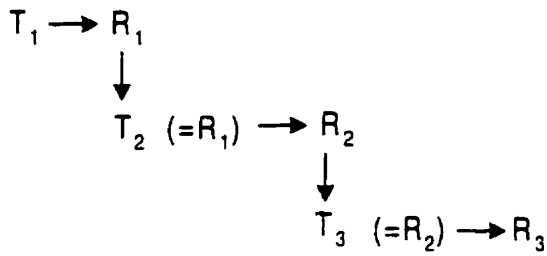
Dans cette annexe, on présente les progressions thématiques de Danes (1970, 1974) Il est à noter que l'ensemble du matériel présenté ici est extrait (section 1.1.3.3.1) de la thèse d'Élisabeth Le (1996) intitulée : « Structure discursive comparée d'écrits argumentatifs en français et en anglais ». On présente les principales progressions thématiques.

De l'école linguistique de Prague, Danes (1970,1974) a identifié quatre types de développement du discours en se basant sur des textes en tchèque, allemand et anglais. Son analyse repose sur la distinction entre le thème T (ce dont il s'agit) et le rhème R (ce qui est dit sur le thème) dans la phrase.

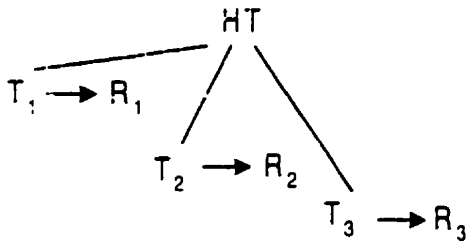
Dans une progression à thème constant (Figure C.1), le même thème est répété (sémantiquement), mais les rhèmes sont différents. Dans une progression linéaire (Figure C.2), un rhème devient le thème de la phrase suivante. Dans le troisième type de développement, la progression à thème dérivé (Figure C.3), les différents thèmes sont reliés à un thème central, l'hyperthème, et vont du particulier au général ou vice-versa. Enfin, il peut également y avoir « éclatement » du rhème (Figure C.4), chaque partie de ce dernier devenant le thème d'une phrase suivante.



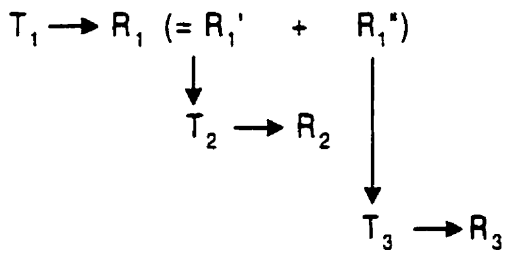
C.1 - Progression à thème constant (Danes)



C.2 - Progression linéaire (Danes)



C.3 - Progression à thème dérivé (Danes)



C.4 - Éclatement du rhème (Danes)

ANNEXE IV: The Role of Knowledge in Software Development, version conventionnelle.

Dans cette annexe on présente l'article « The Role of Knowledge in Software Development » (Robillard, 1999) en version conventionnelle, telle que parue dans la revu *Communication of the ACM* en janvier 1999.

THE ROLE OF KNOWLEDGE IN SOFTWARE DEVELOPMENT

Before the program can be written, humans have to describe and organize the knowledge it represents according to specific knowledge structures.

SOFTWARE DEVELOPMENT IS KNOWLEDGE-INTENSIVE. MANY CONCEPTS HAVE BEEN DEVELOPED to ease or guide the processing of knowledge in software development, including information hiding, modularity, objects, functions and procedures, patterns, and more. These concepts are supported by various methods, approaches, and tools using symbols, graphics, and languages. Some are formal; others are semiformal or simply made up of key practices. Methods and approaches in software engineering are often based on the results of empirical observations or on individual success stories.

Key concepts have been developed by researchers in the cognitive sciences to account for the various aspects of knowledge processing. My goal here is to bridge the gaps between the viewpoints of cognitive scientists and software scientists and practitioners regarding knowledge and outline the characteristics of the related concepts in software methodologies and approaches. The benefits are twofold: a better understanding of the cognitive processes involved in software development and an additional scheme for developing new software practices, methods, and tools [1].

The mental processing and representation of knowledge are complex activities, and our understanding is still rudimentary and subject to debate [10]. A general concept for describing knowledge is as elusive as ever, though various key concepts been developed from specific viewpoints in the cognitive sciences. Some of them are derived from the content or structure of knowledge, others from its representation. Here, "knowledge" refers to a permanent structure of information stored in memory. "Knowledge representation" refers to a transitory construction built up in memory for the processing of a specific situation. Table 1 lists the viewpoint corresponding to each key knowledge concept in the cognitive sciences.

There are many ways to define knowledge. One is to consider the way it is stored in human memory. Related studies have identified two types of knowledge—procedural and declarative—and their corresponding memory contents.

Procedural knowledge, including psychomotor ability, is dynamic. Procedural memory stores all the information related to the skills developed to interact with

PIERRE N. ROBILLARD

our environments (such as walking, talking, typing, and mouse clicking). Knowledge acquisition is based mainly on practice. Procedural knowledge never requires verbal support and is very difficult to describe but, once learned, is rarely forgotten. Such knowledge includes what we call know-how, or knowledge built up through experience. Early designers of expert systems underestimated the complexity of this knowledge concept.

Declarative knowledge, based on facts, is static and concerned with the properties of objects, persons, and events and their relationships. Declarative memory contains all the information that is consciously and directly accessible. Declarative knowledge is easy to describe and to communicate. Declarative memory consists of two types of knowledge—topic, or semantic, and episodic.

Key Knowledge Concept	Viewpoint
Procedural/Declarative	Knowledge nature of content
Schema	Knowledge internal structure
Proposition	Formal knowledge representation
Chunking	Representing unit of knowledge
Planning	Managing knowledge structures

Table 1. Key knowledge concepts in the cognitive sciences

Topic knowledge refers to the meaning of words, such as definitions in dictionaries and textbooks. Topic memory is made up of all the cultural structures of an environment and supports the organization of knowledge related to an environment. Such environments exist at various levels, including social, personal, professional, and technical (such as structured analysis and object-oriented).

Episodic knowledge consists of one's experience with knowledge. Examples include reusing a function, decomposing data-flow diagrams, defining objects from specification requirements, building entity-relation graphs, and documenting programs. Most of these activities are learned through experience once the topic knowledge is obtained from textbooks or courses.

Software development requires topic and episodic knowledge. Difficulties may arise when software developers have only topic knowledge of the application domain, so experience with the knowledge of the application domain may be left out of the software being developed. An example is a well-designed but inappropriate software application. At the coding level, lack of episodic knowledge in the programming language sometimes results in an unduly complex program. Novice programmers have only a limited store of episodic knowledge.

The quality of the software design derived from a methodology can vary according to the designer's episodic knowledge of the methodology. A methodology learned from a book or a crash course is essentially based on topic knowledge. Some methodologies may require more episodic knowledge than others, and the level of episodic knowledge required should be measured or accounted for in some way when evaluating a methodology or the quality of a software design.

The notion of "schema" was first proposed for artificial intelligence by Minsky in 1975 and for psychological studies by Bower et al. in 1979 for describing knowledge structure. The schema concept assumes that knowledge is stored in a human's memory in a preorganized way. Schemas describe specific links among various knowledge elements. A schema is a generic structure built up from an undefined variety of topics and from episodic knowledge. The topic part of the schema represents objects or events; the episodic part represents temporal or causal links between objects or events [9].

According to the schema concept, our understanding of the world is based on the structure of the knowledge organization. A schema is a knowledge structure made up of variables, or slots, that capture the regularities among objects and events. These slots take into account the properties of objects, typical sequences, or any typical knowledge related to the schema. They often have default values. Schemas have predefined or assumed values for some of their variables. Schemas are rarely fully specified, and values for objects or relations are often assumed. Schemas are also context-dependent [10]. For example, your schema of your operating system represents your memory organization of the related items of topical knowledge, including icons, setup, layout, and menu structure. It also comprises episodic knowledge built up from users' experience with the operating system, including how to run a program, open a file, use a spreadsheet, listen to CD music, and use email.

The power of the schemas and their default values are illustrated by the following scenario (which includes many schemas):

> Daniel opens his computer and edits his homework while listening to music by Mozart. He emails a chapter to his classmate for revision, then plays a game called ABC while waiting for her reply.

You understand this scenario based on your schemas, which are filled with default values based on your personal topic and episodic knowledge related to them. For example, the user's schema of a computer

has a default value corresponding to the operating system for the editing schema; the default value could be any of the many text editors (such as Word, Wordperfect, Frameword, and LaTex). Your schema of Mozart's music could be just some classical music or specific Mozart masterpieces. However, you are likely to lack a default value for the schema of the electronic game ABC.

Schema default values can be unexpected yet major components in software development activities, because they are based on the developer's personal experience with a particular area of knowledge. Schema default values need to be validated. Walkthroughs, reviews, and inspection meetings help validate or define the default values of the various schemas used. Although schema validation is carried out implicitly most of the time, it might be rewarding to base these meetings on explicit schema default-value validation. In such cases, each schema and its corresponding validated variables are identified.

WE HUMANS INTUITIVELY UNDERSTAND THAT GOOD DESIGN EMERGES FROM THE SPECIFICATION OF A WELL-DEFINED PROBLEM.

Schemas have been used to study text understanding and software comprehension [2]. Some software methodologies explicitly promote the use or creation of schema; an example is software patterns.

AI also uses the notion of schema as a data structure representing a concept stored in computer memory. The theory behind the schema in AI uses the notions of frames, scripts, and plans. Natural schemas (not those used in AI) are fuzzy concepts. Schemas are all embedded and usually a mix of subschemas, which are also mixtures of subschemas. At some point, we reach basic, or primitive, schemas, leading to the notion of the "proposition."

Knowledge formulation is based on atomic components described in terms of propositions and predicates. A proposition is the smallest unit of knowledge constituting an affirmation as well as the smallest unit that can be true or false. Propositions are discrete representations of knowledge elements and seem to be interconnected in memory based on their shared arguments. According to some authors in AI, mental models are defined by propositional representations [3].

The theoretical hypothesis concerning the cognitive structure of human information systems states that, at a certain level, information is organized in propositional form. Many psychologists studying the representation of meaning in memory believe that the propositional representation is the dominant representation in the human brain and that propositions have three principal functions:

- They can represent any well-specified information from which it follows that propositions form a general mechanism for representing knowledge.
- They preserve the meaning but not the form of a statement or sentence.
- They naturally support reasoning and inferences.

Based on a model of text comprehension, it has been estimated that the working human memory can handle from one to four propositions. A proposition is a formal representation of knowledge. Software development based on formal specifications relies on the propositional representation of knowledge, which is applicable mainly to well-defined problems.

A problem is well-defined if the initial state, the goal, and a set of possible operations are available to reach the goal from the initial state. An academic problem, for example, is a typical well-defined problem, formulated from defined knowledge and requiring students to select the right set of operations for its solution. It might ask you to: Write a program in C to implement a first-in-first-out data structure. Solutions to such problems are characterized by a search in the memory for existing algorithms that may provide the answer. The ability to solve well-defined problems is acquired through study. These problems are intellectual exercises and formalized easily.

An ill-defined problem does not have a well-specified goal because many goals may be acceptable. In the same way, the cognitive approaches used to solve ill-defined problems cannot be defined clearly because there might be many ways to solve them [12].

Software design problems usually belong to the family of ill-structured problems. Their solutions are acceptable in varying degrees and are rarely either absolutely correct or absolutely incorrect. The design task structures a problem by finding the missing information or creating new information and using it to specify new goals within the problem-knowledge space [4]. Software design is generally a mixture of ill- and well-defined problems. The specification and the

design of the algorithms or the system architecture often constitute an ill-defined problem type; translation of the detailed design into programming code is more of a well-defined problem type.

The nature of the problem is not defined in an absolute way, depending somehow instead on the solver's level of experience. A novice may find a problem ill-defined, while an experienced designer considers it well-defined, because a well-defined goal for reaching the solution is available. A problem is perceived as being more or less complex, depending on the existing goal definition in the mind of the solver. The perception of complexity raises the problem of software complexities in the planning activity.

We humans intuitively understand that good design emerges naturally from the formal specification of a well-defined problem. However, less obvious is that formal specifications are appropriate for ill-defined problems. Formal specifications are based on propositions. Recall that propositions have three functions (listed earlier); of these, the first and the third do not apply to ill-defined problems. The information is clearly well-specified, and reasoning and inference are not the dominant mental activities required to solve ill-defined problems. The answer to the everlasting debate on the appropriateness of formal specifications for software development may lie in the nature of the problems to be solved. Formal specification seems to be more appropriate for well-defined problems.

THE MENTAL MECHANISMS INVOLVED IN PLANNING ARE NOT FULLY UNDERSTOOD AND CANNOT BE FULLY AUTOMATED.

Another component of the mental process is the amount of knowledge available for immediate processing. Psychologists use the concept of chunks to account for the limited amount of knowledge that can be handled by the human mind at any given time. Chunks are general and do not refer to the information content of the knowledge. It is well known (since Miller's classic experiment in 1957) that short-term memory, or working memory, has a limited capacity and can typically process only 7±2 chunks at a time.

For example, it is more difficult to memorize the letter combination BMI LMU than IBM UML. The second sequence is, however, no more difficult to memorize than computer/Unified Modeling Language. In the first sequence, each letter is a chunk (six chunks, for a total of six letters); in the second, each group of letters is a chunk (two chunks, for a total of six letters); and in the third sequence, each group of words is a chunk (two chunks for a total of 31 letters). A chunk is a unit of information whose significance varies with the individual reader. For example, UML may not be easier to remember than LMU for someone unfamiliar with the UML object-oriented methodology; in such a case, IBM UML is composed of four chunks (1+3).

Software methodologies based on encapsulation, information hiding, modularization, abstraction, and even the divide-to-conquer approach all deal with the chunking phenomenon. Successful methodologies based on icons, graphic symbols, and reserved words are naturally limited to the chunk number for the simultaneous use of elements in working memory. Extensive computer-aided software engineering (CASE) tools or methodologies requiring overchunking by users (too many features to remember simultaneously) are likely to meet with a lack of approval. Since chunks do not refer to information content, they are a measure of the unrelated knowledge that can be processed naturally.

Making Plans

Software development involves processing a large amount of information distributed over many knowledge domains that are more or less contiguous with fuzzy frontiers that are intertwined most of the time. The limited capacity of the human mind's working memory cannot keep track of all the information from all the knowledge domains visited. Plans are therefore needed to manage the knowledge. Plans are knowledge representations used to organize knowledge based on various criteria and to guide the tasks to be done by the mind. The properties of plans are anticipation and simplification. Anticipation accounts for the expected results associated with a plan and are based on experience. Plans can be a set of subgoals defining the main steps to be reached before a final goal is achieved. Plans are not necessarily procedural, and each subgoal does not necessarily correspond to a well-defined activity. Plans have three main characteristics [6]:

- A heuristic nature. Plans efficiently guide mental activity toward the most promising avenue based

on the knowledge available without a detailed analysis of the situation.
- Optimal use of memory. Plans keep only critical properties of objects or events by making abstractions of all nonsalient details associated with the activity being carried out.
- Higher control level. Plans enable the emergence of an activity that cannot be derived from the detail of the activity being processed.

The following scenario illustrates how plans guide the design process. The mental structure of the designer is in a "local" knowledge state; local means that the amount of knowledge available at any given time for immediate processing by the brain is limited to the capacity of the working memory. The designer must therefore move continuously from the local state to another state of knowledge. This move can be done in a completely arbitrary fashion or can be based on plans. Arbitrary or heuristic approaches can be associated with a dreaming activity or limited self control in the state of mind entered into. Software designers would rather (hopefully) rely on planned activities that can be either rigorous and systematic or opportunistic [5].

A systematic planning approach is when designers believe they have access to all the knowledge required to do the task. It has been observed in studies related to the psychology of programming that, for example, experts adopt a planning mechanism based on a breadth-first approach, while novices, who often rely on their understanding of programming languages, adopt a depth-first approach. Designers actually follow well-structured plans as long as they find nothing better to do. When knowledge is not readily available, some explanatory mechanisms are required. The explanatory process is called "opportunistic," because at various points in the process the designer makes a decision or takes action depending on the opportunities presented. The decisions are motivated by earlier decisions and are not the product of a well-planned process [4].

Designers progress from a systematic planning activity to an opportunistic one with the evolution of the design, a process that is not always balanced [11]. Design rationales are especially useful when opportunistic planning has occurred [7], because they capture the information on which decisions are based.

Serendipitous planning occurs when designers try to group together or integrate a set of decisions or plans into a single coherent plan. Grouping together means that partial solutions are recognized at various levels of detail and are combined [8]. Software reuse is suitable for serendipitous planning.

Developing plans depends on designers' experience with the design solution and their ability to associate existing plans. It has been suggested by researchers in the psychology of programming that lack of experience increases a design's variability and then contributes to the software's complexity. Expert knowledge is organized in a more abstract and deeper way. Resulting plans are based on situations already seen, rather than on trial-and-error exploration. Studies on planning have shown that expert plan structures have four abstract characteristics [6]:

- Hierarchical with multiple levels;
- Explicit relationships between levels;
- Based on basic schema recognition; and
- Well connected internally.

Planning is one of the human brain's most powerful natural activities, although the mental mechanisms involved are not fully understood and cannot be fully automated. Early methodologies and software tools have sought to define and enforce some planning activities based mainly on hierarchical top-down development, an ingenious approach that generated great expectations but little success.

Some CASE tools are artificial guides for planning activities. The following partial list of desired methodologies or CASE tool features is based on our current understanding of mental processes and is divided into two sections—the first helpful in planning activities, the second helpful in representational activities [12]:

- Helps organize mental activity;
- Enables deviation from or even abandonment of plans; never imposes fixed hierarchical planned activities;
- Supports a return to an original plan, never assuming or imposing it;
- Enables work at various levels of detail and abstraction;
- Helps manage the limits of human memory by making various levels of knowledge available simultaneously;
- Maintains traces of abandoned or interrupted tasks or plans for easy, spontaneous return.

- Generates visual representations adapted to the designer's level of experience and to the various viewpoints expressed;
- Presents the solution's constraints;
- Enables easy change in the representation level;
- Helps build representations;
- Helps create the design;
- Captures the design rationale; and
- Outlines plan structures and strategies.

Conclusions

Software development is the processing of knowledge in a very focused way. We can say it is the progressive crystallization of knowledge into a language that can be read and executed by a computer. The knowledge-crystallization process is directional, moving from the knowledge application domain to software architectural and algorithmic design knowledge, and ending in programming language statements.

Software engineers have developed methods, practices, and tools to ease the knowledge-crystallization process. And cognitive scientists have studied the properties of knowledge from various points of view. Our purpose is to merge these two approaches.

New directions in software engineering may result by considering established views of knowledge structures and representations from the cognitive sciences. Each knowledge concept presented here illustrates a feature of the mental processing of knowledge. These concepts are derived from observations and are applicable to any mental activity, including software development.

Cognitive scientists derive their understanding of knowledge through their observation of experts and novices at work and from their controlled experiments. Software engineers have made little use of these approaches, however, and few methodologies or CASE tools are derived from documented observations or controlled experiments.

An immediate benefit for software engineering is to account for the known characteristics of mental knowledge processing. Some methodologies or CASE tools could be improved, or at least be made to not work in a counterproductive way by interfering with the brain's natural processing of knowledge. It should then be easy to identify the knowledge viewpoints targeted by a component of a method or a function of a tool.

Experience plays a major role in any knowledge-related activity. Psychologists recognize a distinct structure, called "episodic," in human memory that accounts for experience. Any project leader knows the value of experience. Psychologists also know that knowledge processing by an expert is quite different from a novice's knowledge processing. Software engineers rarely define the level of experience required to use a methodology or a tool, and some software advertisements claim no experience is needed. Is such a tool useful to an expert?

It is important to distinguish between the knowledge structures supporting understanding (schemas) and the mechanisms used to organize that knowledge (plans). Software engineers have placed a great deal of emphasis on documenting the final representation of the knowledge structure, or the source code. But the documentation of the plan has only recently been introduced through the design rationale, which documents the process through which the knowledge is structured or crystallized.

Software complexity, software quality, and software metrics may find common ground if the level of opportunistic planning in a given task can be measured. Such a measure would be a sign of the stability, or the quality, of the design and reflect the designer's experience in a particular knowledge domain.

Software development can be improved by recognizing the related knowledge structure or representations, including building schemas, validating schema default values, acquiring topic knowledge, requiring appropriate episodic knowledge, performing planning activities, applying formal specifications (encoding knowledge into a propositional form) to define problems, and having the appropriate tools to manage the chunking phenomenon. ■

References

1. Curtis, B. Objects of our desire: Empirical research on object-oriented development. *Hum.-Comput. Interact. 10* (1995) 337–344.
2. Detienne, F. Design strategies and knowledge in object-oriented programming: Effects of experience. *Hum.-Comput. Interact. 10* (1995), 129–169.
3. Fagin, R., Halpern, J., Moses, Y., and Vardi M. A model for knowledge. In *Reasoning About Knowledge*. MIT Press, Cambridge, Mass., 1995, pp. 15–45.
4. Guindon, R. Knowledge exploited by experts during software system design. *Int. J. Man-Mach. Stud. 33* (1990), 279–304.
5. Hayes-Roth, B., and Hayes-Roth, F. A cognitive model of planning. *Cog Sci. 3* (1979), 275–310.
6. Hoc, J. *Psychologie Cognitive de la Planification*. Presse Universitaire de Grenoble, Grenoble, France, 1987.
7. Lee, J., Lai K.-Y., What's in design rationale? In *Design Rationale*. Chapt. 2. T. Moran and J. Caroll, J., Eds. Lawrence Erlbaum Associates, Mahwah, N.J., 1996, pp. 21–51.
8. Rist, R. Variability in program design: The interaction of process with knowledge. *Int. J. Man-Mach. Stud. 33* (1990), 305–322.
9. Simon, H. Information processing models of cognition. *Ann. Rev. Psych. 30* (1979), 363–396.
10. Sternberg, R. Intelligence. In *Thinking and Problem Solving*. R. Sternberg, Ed. Academic Press, San Diego, Calif., 1994, pp. 263–288.
11. Visser, W. Organization of design activities: Opportunistic with hierarchical episodes. *Interact. Comput. 6*, 3 (1994), 239–274.
12. Visser, W., and Hoc, J. Expert software design strategies. In *Psychology of Programming*. Academic Press, San Diego, Calif., 1990, pp. 235–247

PIERRE N. ROBILLARD (robillard@rgl.polymtl.ca) is a professor in the Department of Electrical and Computer Engineering at the Ecole Polytechnique de Montréal in Montréal, Canada.

This work was supported in part by grant A0141 from the Natural Sciences and Engineering Research Council of Canada. Additional support was provided by the Applied Software Engineering Center

ANNEXE V: The Role of Knowledge in Software Development, version SMML.

Dans cette annexe on présente l'article « The Role of Knowledge in Software Development » (Robillard, 1999) en version SMML.

Introduction

Software developpement and knowledge

Software development is knowledge-intensive. Many concepts have been developed to ease or guide the processing of knowledge in software development, such as information hiding, modularity, objects, functions and procedures, patterns, and more. These concepts are supported by various methods, approaches, and tools using symbols, graphics, languages. Some are formal; others are semi-formal or simply made up of key practices. Methods and approaches in software engineering are often based on the results of empirical observations or on individual success stories.

Key concepts have been developed by researchers in cognitive sciences to account for the various aspects of knowledge processing.

But

Main goal of article

My goal here is to bridge the gap between the viewpoints of cognitive scientists and software practitioners and scientists regarding concepts related to knowledge and outline the characteristics of the knowledge and approaches.

Objectif

Objectives of article

The benefits are twofold: a better understanding of the cognitive processes involved in software development and an additional scheme for developing new software practices, methods, and tools [1].

Developpement

Main knowledge concepts and software engineering

Liant

There is no general concept to describe knowledge

The mental processing and representation of knowledge are complex activities, and our understanding is still rudimentary and subject to debate [10]. A general concept to describe knowledge is as elusive as ever, though various key concepts been developed from specific viewpoints in the cognitive sciences. Some of theme are derived from the content or structure of knowledge, others from its representation.

Definition

Knowledge

Here, "knowledge" refers to a permanent structure of information stored in memory.

Definition

Knowledge representation

"Knowledge representation" refers to a transitory construction built up in memory for the processing of a specific situation.

E.1 – Document SMML, page 1

Figure

Knowledge concept in cognitive sciences

Table 1 list the viewpoint corresponding to each key knowledge concept in the cognitive sciences.

Key knowledge concept	Viewpoint
Procedural/Declarative knowledge	Nature of knowledge content
Schema	Internal structure of knowledge
Proposition	Formal representation of knowledge
Chunking	Representation of knowledge unit
Planning	Management of knowledge structures

Table 1. Key knowledge concepts in the cognitive sciences.

Constituant

Knowledge Type

Notion

Knowledge Type

There are many ways to define knowledge. One is to consider the way it is stored in human memory. These studies have identified two types of knowledge - procedural and declarative- and their corresponding memory contents.

Notion

Procedural knowledge

Definition

Procedural knowledge

Procedural knowledge, including psychomotor ability, is dynamic. Procedural memory stores all the information related to the skills developed to interact with our environment (such as walking, talking, typing, and mouse clicking).

Knowledge acquisition is based mainly on practice. Procedural knowledge never requires verbal support and is very difficult to describe, but once learned, it is rarely forgotten. Such knowledge includes what we call know-how, or knowledge built up through experience.

Discussion

Expert-Systems and procedural knowledge

Early designers of expert-systems have underestimated the complexity of this knowledge concept.

Notion

Declarative knowledge

Definition

Declarative knowledge

Declarative knowledge, based on facts, is static and concerned with the properties of objects, persons and events and their relationships.

Declarative memory contains all the information that is consciously and directly accessible. Declarative knowledge is easy to describe and to communicate. Declarative memory is made up of two types of knowledge -topic or semantic, and episodic.

Notion

Topic knowledge

Definition

Topic knowledge

Topic knowledge refers to the meaning of words, such as definitions in dictionaries or textbooks.

Topic memory is made up of all the cultural structures of an environment and supports the organization of knowledge related to an environment. Such environment can exist at various levels, including social, personal, professional, and technical (such as Structured Analysis or Object Oriented, etc.)

Notion

Episodic knowledge

Definition

Episodic knowledge

Episodic knowledge is made up of one's experience with knowledge.

Examples include reusing a function, decomposing data-flow diagrams, defining objects from specification requirements, building entity-relation graphs, and documenting programs. Most of these activities are learned through experience once the topic knowledge is obtained from textbooks or courses.

E.3 – Document SMML, page 3

Discussion

Software development and types of knowledge

Software development requires both topic and episodic knowledge. Difficulties may arise when software developers have only a topic knowledge of the application domain, so experience with the knowledge of the application domain may be left out of the software being developed.

Exemplification

inappropriate software application

An example is a well-designed but inappropriate software application. At the coding level, lack of episodic knowledge in the programming language sometimes results in an unduly complex program. Novice programmers have only a limited store of episodic knowledge.

Exemplification

Quality of software VS episodic knowledge

The quality of the software design derived from a methodology can vary according to the designer's level of episodic knowledge of the methodology. A methodology learned from a book or a crash course is essentially based on topic knowledge. Some methodologies may require more episodic knowledge than others, and the level of episodic knowledge required should be measured or accounted for in some way when evaluating a methodology or the quality of a software design.

Constituant

Schema

Notion

Notion of schema

The notion of the "schema" was first proposed for artificial intelligence by Minsky in 1975 and for psychological studies by Bower et al. in 1979 for describing knowledge structure. The schema concept assumes that knowledge is stored in a human's memory in a preorganized way.

Definition

Schema

Schemas describe specific links among various knowledge elements. A schema is a generic structure built up from an undefined variety of topics and grom episodic knowledge.

The topic part of the schema represents objects or events; the episodic part represents temporal or causal links between objects or events [9].
According to the schema concept, our understanding of the world is based on the structure of the knowledge organization. A schema is a knowledge structure made up of variables, or slots, that capture the regularities among objects and

E.4 – Document SMML, page 4

events. These slots take into account properties of objects, typical sequences, or any typical knowledge related to the schema. They often have default values. Schemas have predefined or assumed values for some of their variables. Schemas are rarely fully specified, and values for objects or relations are often assumed. Schemas are also context-dependent [10].

Exemplification

Schema of operating system

For example, your schema of operating system represents the memory organization of the related items of topical knowledge, including icons, setup, layout, and menu structure. It also comprises episodic knowledge built up from the experience the user with the operating system, including how to run a program, open a file, use a spreadsheet, listen to CD-music, use e-mail.

Exemplification

Scenario of schema

The power of the schemas and their default values are illustrated by the following scenario (which includes many schemas):

Daniel opens his computer, and edits his homework while listening to music by Mozart. He e-mails one chapter to his classmate for revision, then plays a game called ABC while waiting for her reply.

Your understand this scenario based on your schemas, which are filled with default values based on your personal topic and episodic knowledge related to them. For example, the user's schema of a computer has a default value which corresponding to the operating system for the editing schema; the default value could be any one of the many text editors (WORD, WORDPERFECT, FRAMEWORD, and LaTex). Your schema of Mozart's music could be just some classical music or specific Mozart masterpieces. However, you are likely to lack a default value for the schema of the electronic game called ABC.

Discussion

Software engineering and schema

Schema default values can be unexpected yet major components in software development activities, because they are based on the developer's personal experience with a particular area of knowledge. Schema default values need to be validated. Walk-throughs, reviews and inspections meeting help validate or define the default values of the various schemas used. Although schema validation is carried out implicitly most of the time, it might be rewarding to base these meetings on explicit schema default-value validation. In such cases, each schema and its corresponding validated variables are identified.
Schemas have been used to study text understanding and software comprehension [2]. Some software methodologies explicitly promote the use or the creation of schema; an example is software pattern.

Discussion

Artificial Intelligence and schema

AI also uses the notion of the schema as a data structure representing a concept stored in computer memory. The theory behind the schema in AI uses the notions of frames, scripts and plans. Natural schemas (not the ones used in AI) are fuzzy concepts. Schemas are all embedded and usually a mix of subschemas, which are also mixtures of subschemas. At some point, we reach basic, or primitive, schemas, leading to the notion of the "proposition". Knowledge formalization is based on atomic component described in terms of propositions and predicates. A proposition is the smallest unit of knowledge constituting an affirmation as well as the smallest unit that can be true or false. Propositions are discrete representations of knowledge elements and seem to be interconnected in memory based on their shared arguments. According to some authors in AI, mental models are defined by propositional representations [3].

Constituant

Proposition

Notion

Notion of proposition

The theoretical hypothesis concerning the cognitive structure of human information systems states that, at a certain level, information is organized in propositional form. Many psychologists studying the representation of meaning in memory believe that the propositional representation is the dominant representation in the human brain and that propositions have three principal functions:
- They can represent any well-specified information from which it follows that propositions form a general mechanism for representing knowledge.
- They preserve the meaning but not the form of a statement or sentence.
- They naturally support reasoning and inferences.

Based on a model of text comprehension, it has been estimated that the working memory can handle from one to four propositions. A proposition is a formal representation of knowledge.

Discussion

Formal specifications and propositional representation

Software development based on formal specifications relies on the propositional representation of knowledge, which is applicable mainly to well-defined problems.

Constituant

Notion of well define and ill-defined problem

Notion

Well-defined problem

A problem is well-defined if the initial state, the goal, and a set of possible operations are available to reach the goal from the initial state.

Exemplification

well-defined problem: academic problem

An academic problem, for example, is a typical well-defined problem, formulated from defined knowledge and requiring students to select the right set of operations for it solution. I might ask you to: Write a program in C to implement a first-in-first-out data structure. Solutions to such problems are characterized by a search in the memory for existing algorithms that may provide the answer.

The ability to solve well-defined problems is acquired through study. These problems are intellectual exercises and formalized easily.

Notion

Ill-defined problem

An ill-defined problem does not have a well-specified goal because many goals may be acceptable. In the same way, the cognitive approaches used to solve ill-defined problems cannot be defined clearly because there might be many ways to solve them [12].

Discussion

Software design problems usually ill-structured problems

Software design problems usually belong to the family of ill-structured problems. Their solutions are acceptable in varying degrees and are rarely either absolutely correct or absolutely incorrect. The design task structures a problem by finding the missing information or creating new information and using it to specify new goals within the problem knowledge space [4]. Software design in general is a mixture of ill- and well-defined problems. The specification and the design of the algorithms or the system architecture often constitute an ill-defined problem type; translation of the detailed design into programming code more of a well-defined problem type.

The nature of the problem is not defined in an absolute way, depending somehow instead on the solver's level of experience. A novice may find a problem ill-defined, while an experienced designer considers it to be well-defined, because a well-defined goal for reaching the solution is available. A problem is perceived as being more or less complex, depending on the existing goal definition in the mind of the solver. The perception of complexity raises the problem of software complexities in the planning activity.

E.7 – Document SMML, page 7

Discussion

Formal Specification and type of problem

We humains intuitively understand that good design naturally emerge from the formal specification of a well-defined problem. However, less obvious is that formal specifications are appropriate for ill-defined problems. Formal specifications are based on propositions. Recall that propositions have three functions (listed earlier); of these, the first and third do not apply to ill-defined problems. The information is not well-specified, and reasoning and inference are not the dominant mental activities required to solve ill-defined problems. The answer to the everlasting debate on the appropriateness of formal specifications for software development may lie in the nature of the problems to be solved. Formal specification seems to be more appropriate for well-defined problems.

Constituant

Chunk of information and processing

Notion

Chunk of information and processing

Another component of the mental process is the amount of knowledge available for immediate processing. Psychologists have been using the concept of chunks to account the limited amount of knowledge that can be handled by the mind at any given time. Chunks are general and do not refer to the information content of the knowledge. It is well know (since Miller's classic experiment in 1957) that short-term memory, or working memory, has a limited capacity and can typically only process 7± 2 chunks at a time.

Exemplification

BMI LMU/IBM UML example on chunk

For example, it is more difficult to memorize BMI LMU than IBM UML. This last sequence is not really more difficult to memorize than computer / Unified Modeling Language. In the first sequence, each letter is a chunk (six chunks for a total of six letters); in the second, each group of letters is a chunk (two chunks, for a total of six letters); finally, in the third sequence, each group of words is a chunk (two chunks for a total of 31 letters). A chunk is a unit of information whose significance varies with the individual reader. For example, UML may not be easier to remember than LMU for someone who is not familiar with the UML object-oriented methodology; in such a case, IBM UML is composed of four chunks (1+3).

Discussion

Software methodologie and chunk

Software methodologies based on encapsulation, information hiding, modularization, abstraction, and even the divide-to-conquer approach all deal with the chunking phenomenon. Successful methodologies based on icons, graphic symbols or reserved words are naturally limited to the chunk number for simultaneous use of elements in working memory. Extensive computer-aided

software engineering (CASE) tools or methodologies requiring over-chunking by users (too many features to remember simultaneously) are likely to meet a lack of approval. Since chunks do not refer to information content, they are a measure of the unrelated knowledge that can be processed naturally.

Constituant

Making plans

Notion

Notion of plans

Software development involves processing of a large amount of information distributed over many knowledge domains that are more or less contiguous with fuzzy frontiers that are intertwined most of the time. The limited capacity of the human mind's working memory cannot keep track of all the information from all the knowledge domains visited. Plans are therefore needed to manage the knowledge.

Definition

Plans

Plans are knowledge representations used to organize knowledge based on various criteria and to guide the tasks to be done by the mind.

Liant

Properties and caracteristics of plans

The properties of plans are anticipation and simplification. Anticipation accounts for the expected results associated with a plan and are based on experience. Plans can be a set of subgoals defining the main steps to be reached before the final goal is achieved. Plans are not necessarily procedural, and each subgoals does not necessarily correspond to a well-defined activity. Plans have three main characteristics[6]:

- A heuristic nature. Plans efficiently guide mental activity towards the most promising avenue based on the knowledge available without a detailed analysis of the situation.

- Optimal use of memory. Plans keep only critical properties of objects or events by making abstractions of all nonsalient details associated with the activity being carried out.

- Higher control level. Plans enable the emergence of an activity that cannot be derived from the level of detail of the activity being processed.

Discussion

Plans and design

Exemplification

Scenario of plans

The following scenario illustrates how plans guide the design process. The mental structure of the designer is in a "local" knowledge state; local means that the amount of knowledge available at any given time for immediate processing by the brain is limited to the capacity of the working memory. The designer must therefore move continuously from this local state to another state of knowledge. This move can be done in a completely arbitrary fashion or can be based on plans. Arbitrary or heuristic approaches can be associated with a dreaming activity or limited self control in the state of mind entered into. Software designers would rather (hopefully) rely on planned activity that can be either rigorous and systematic or opportunistic [5].

Liant

Planning approach of novice and expert

A systematic planning approach is when designers believes they have access to all the knowledge required to do the task. It has been observed in studies related to the psychology of programming that, for example, that experts adopt a planning mechanism that is based on a breadth-first approach, while novices, who often rely on their understanding of programming languages, adopt a depth-first approach. Designers actually follow well-structured plans as long as they find nothing better to do. When knowledge is not readily available, some explanatory mechanisms are required. The explanatory process is called "opportunistic", because at various points in the process the designer makes a decision or action depending on the opportunities that presented. The decisions are motivated by earlier decisions, and are not the product of a well-planned process [4].
Designers progress from a systematic planning activity to an opportunistic one with the evolution of the design, a process that is not always well balanced [12]. Design rationales are especially useful when opportunistic planning has occurred [7], because they capture the information on which the decisions are based.

Liant

Serendipitous planning

Serendipitous planning occurs when designers try to group together or integrate a set of decisions or plans in a single coherent plan. Grouping together means that partial solutions are recognized at various levels of detail and are combined [8]. Software reuse is suitable for serendipitous planning.

Liant

Design experience and plans

Developing plans depends on designer's experience with the design solution and their ability to associate existing plans. It has been suggested by researchers in the psychology of programming that lack of experience increases a design's variability and then contributes to the software's complexity. Expert knowledge is organized in a more abstract and deeper way. Resulting plans are based on situations already seen, rather than on trial-and-error exploration. Studies on planning have shown that expert plan structures have four abstract characteristics[6]:

- Hierarchical with multiple levels;
- Explicit relationships between levels;
- Based on basic schema recognition; and
- Well connected internally.

Planning is one of the human brain's most powerful natural activities, although the mental mechanisms involved are not yet fully understood and cannot be fully automated. Early methodologies and software tools have sought to define and enforce some planning activities based mainly on hierarchical top-down development, an ingenuous approach that generated great expectations but little success.

Liant

CASE and plans

Some CASE tools are artificial guides for planning activities. The following partial list of desired methodologies or CASE tool features is based on our current understanding of the mental processes and is divided into two sections - the first helpful in planning activities[12]:

- Help to organize mental activity;
- Enables deviation from or even abandonment of plans; never impose fixed hierarchical planned activities;
- Support a return to an original plan, never assuming it or imposing it;
- Enable work at various levels of detail and abstraction;
- Help manage the limits of mind memory by making various levels of knowledge available simultaneously;
- Maintain traces of abandoned or interrupted tasks or plans for easy, spontaneous return.
- Generates visual representations adapted to the designer's level of experience and to the various viewpoints expressed;
- Presents the solution's constraints;
- Enables easy change of representation level;
- Helps build representations;
- Help create the design;
- Captures design rationale; and
- Outlines plan structures and strategies.

Conclusion

Conclusion of article

Terminaison

Software development is crystallization of knowledge

Software development is the processing of knowledge in a very focused way. We can say it is the progressive crystallization of knowledge into a language that can be read and executed by a computer. The knowledge crystallization process is directional, moving from the knowledge application domain to software architectural and algorithmic design knowledge, and ending in programming language statements.

Software engineers have developed methods, practices and tools to ease the knowledge-crystallization process. And cognitive scientists have been studied the properties of knowledge from various point of view. Our purpose is to merge these two approaches.

New directions in software engineering may result by considering established views on knowledge structures and representations from the cognitive sciences. Each knowledge concept presented illustrates a feature of the mental processing of knowledge. These concepts are derived from observations and are applicable to any mental activity, including software development.

Terminaison

Software engineers should use cognitive scientists approaches

Cognitive scientists derive their understanding of knowledge through their observation of experts and novices at work and from their controlled experiments. Software engineers have made little use of these approaches, however, and few methodologies or CASE tools are derived from documented observations or controlled experiments.

An immediate benefit for software engineering is to account for the known characteristics of mental knowledge processing. Some methodologies or CASE tools could be improved, or at least be made to not work in a counterproductive way by interfering with the brain's natural processing of knowledge. It should then be easy to identify the knowledge viewpoints targeted by a component of a method or a function of a tool.

Terminaison

Experience and knowledge-related activity.

Experience plays a major role in any knowledge-related activity. Psychologists recognize a distinct structure, called "episodic", in human memory that accounts for experience. Any project leader knows the value of experience. Psychologists also known that knowledge processing by an expert is quite different from a novice's knowledge processing . Software engineers rarely define the level of experience required to use a methodology or a tool, and sometimes advertisements claim no experience is needed. Is such a tool useful to an expert?

Terminaison

Documentation of plans and software engineering

It is important to distinguish between the knowledge structures supporting understanding (schemas) and the mechanisms used to organize that knowledge (plans). Software engineers have placed a great deal of emphasis on documenting the final representation of the knowledge structure, or the source code. But the documentation of the plan has only recently been introduced through the design rationale, which will document the process through which the knowledge is structured or crystallized.

Terminaison

Mesuring the level of opportunistic planning

Software complexity, software quality, and software metrics may find common ground if the level of opportunistic planning in a given task can be measured. Such a measure would be a sign of the stability, or the quality, of the design and reflect the designer's experience in a particular knowledge domain.

Software development can be improved by recognizing the related knowledge structure or representations, including building schemas, validating schema default values, acquiring topic knowledge, requiring appropriate episodic knowledge, performing planning activities, applying formal specifications (encoding knowledge into a propositional form) to define problems, and having the appropriate tools to manage the chunking phenomenon.