JAROSŁAW FRANCIK

SURVEILLANCE DU FLUX DES DONNEES DANS L'ANIMATION DES ALGORITHMES

RESUME D'UNE THESE EN CO-TUTELLE DIRIGEE PAR :

PROFESSEUR STEFAN WĘGRZYN
PROFESSEUR JEAN-MARC TOULOTTE

INSTITUT D'INFORMATIQUE DE L'ECOLE POLYTECHNIQUE SILESIENNE DE GLIWICE, POLOGNE
LABORATOIRE D'AUTOMATIQUE I^{3}D, UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE, FRANCE
1999

Pour Katherine, ma femme

ABSTRAIT

La visualisation de logiciel consiste à appliquer de techniques diverses multimédia pour l'amélioration significative de la compréhension humaine du logiciel. Si le sujet de telle visualisation est un algorithme, et animation produite par ordinateur est essentiel pour sa réalisation, nous parlons de l'animation des algorithmes.

Les visualisations des algorithmes réussies dépassent habituellement les applications isomorphes des structures des données ou code de programme à la représentation graphique de la sémantique de programme. Elles fournissent en soi un niveau d'abstraction élevé, fournissant une information supplémentaire sur la sémantique et la signification de cela qui est derrière le code. Le postulat de créer des visualisations fortement abstraites est par la plupart des auteurs indiqués en tant qu'étant en contradiction profonde avec le postulat de l'automation du travail du concepteur. Le but du travail présenté est partiel réconciliant ces deux postulats contredits. Nous pouvons prouver que quelques éléments qui augmentent de manière significative le niveau de l'abstraction peuvent être introduits à la visualisation en mode strictement automatique, sans effort additionnel du visualisateur. Pour obtenir ce résultat, on a proposé une méthode originale d'animation des algorithmes basée sur surveillance de flux des données. L'idée principale de la nouvelle méthode est d'acquérir l'information pour la visualisation en observant des opérations élémentaires des flux des données plutôt puis en examinant des valeurs temporaires de structures des données, comme dans les solutions traditionnelles. Dans des versions plus avançées de cette méthode, le formalisme du réseau de Pétri a été appliqué pour l'analyse dynamique des flux des données dans un rayon non-local. Il a laissé créer une machine pour la suppression automatique des données sans importance ou pas désiré.

La méthode de surveillance de flux des données ont été mise en application dans un système d'animation des algorithmes *Daphnis*, qui est le résultat pratique de la recherche. Les visualisations de niveau d'abstraction haute, créées relativement facilement avec le système *Daphnis*, dans les solutions existantes jusqu'ici, a exigé beaucoup d'effort impliquant précis, manuel concevant et l'accord. *Daphnis* dans sa version actuelle est un système facile à utiliser, avec un répertoire large des moyens graphiques. Ce répertoire peut être élargie grâce à l'architecture ouverte du système. Le système convient aux applications didactiques et de technologie.

Le texte de la redaction contient une introduction à la visualisation de logiciel avec un examen des solutions existantes jusqu'ici. Après, la méthode d'animation des algorithmes basée sur surveillance de flux des données est décrite en détail. Ensuite, le système *Daphnis* est présenté, du point de vue de l'utilisateur et du concepteur. Enfin, quelques visualisations exemplaires et remarques finales sont incluses.

MOTS-CLES : Visualisation de logiciel, animation des algorithmes, techniques multimédia, infographie, animation, réseaux de Pétri, flux des données, correction des programmes, systèmes de développement de logiciel.

ABSTRACT

Software visualisation consists of applying various multimedia techniques for significant improvement of human understanding of computer software. If the subject of such visualisation is an algorithm, and computer generated animation is essential for its realisation, we say of algorithm animation.

Successful algorithm visualisations usually go beyond isomorphic mappings of program data or code to graphical representation of program semantics. They inherently provide a high level of abstractness, supplying an extra information on the semantics and meaning that is behind the code. The postulate of designing highly abstract visualisations is by most authors stated as being in a deep contradiction with postulate of automation of the designer's work. The goal of the presented work is partial reconciling these two contradicted postulates. We can prove that some elements that significantly increase the level of abstractness may be introduced to the visualisation in a strictly automatic mode, without any additional effort of the visualiser. To obtain this result, an original method of algorithm animation based on data flow tracing has been proposed. The key idea of the new method is to acquire information for visualisation by observing elementary operations of data flow rather then by examining temporary data structure values used in traditional solutions. In more advanced versions of this method, Petri net formalism has been applied for dynamic analysis of data flow in a non-local range. It allowed creating an engine for automatic suppression of unimportant or unwanted data.

The data flow method has been implemented in a system for algorithm animation called *Daphnis*, which is the practical result of the research. The visualisations created relatively easily with the *Daphnis* system achieve level of abstractness, which, in the solutions existing so far, required much effort involving precise, manual designing and tuning. *Daphnis* in its current version is a general purpose, easy-to-use system, with wide repertoire of graphical means. This repertoire may be broadened thanks to the open architecture of the system. The system is suitable for both didactic and engineering applications.

The text of manual contains an introduction to the software visualisation with a review of solutions existing so far. Next, the method of algorithm animation based on data flow tracing is described in detail. Later, the *Daphnis* system is presented, both from the user's and designer's point of view. Finally, sample visualisations and final remarks are included.

KEYWORDS: Software visualisation, algorithm animation, multimedia techniques, computer graphics and animation, Petri nets, data flow, debugging, software development systems.

SOMMAIRE

DEFINITIONS PRINCIPIALES

ALGORITHME – suite d'opérations permettant d'obtenir un résultat bien spécifié sous une hypothèse de conditions initiales déterminées. Les opération d'algorithme doivent être effectuées dans un ordre défini par des règles précises.

REALISATION ou IMPLEMENTATION D'ALGORITHME – quelconque procedure ou programme qui utilise un algorithme pour obtenir solution d'un probléme déterminé.

VISUALISATION DE LOGICIEL – utilisation de techniques diverses avancées informatiques pour augmenter significativement la compréhension humaine de logiciel. Les techniques utilisées comportent entre d'autres infographie, typographie, conception graphique, photographie, animation, génération et reproduction de son, musique et parole, aussi bien que la cinématographie. Ces techniques sont généralement connues comme techniques de multimédia.

VISUALISATION DES PROGRAMMES – un type de visualisation de logiciel qui focalise ces aspects du logiciel visualisé, qui sont joint avec une réalisation concrète, par exemple un programme.

VISUALISATION DES ALGORITHMES – un type de visualisation de logiciel qui focalise description de ce logiciel fait à niveau élevé, abstrait.

ABSTRACTION – opération de l'esprit, qui isole d'une notion un élément en négligeant les autres. Dans la visualisation de logiciel, l'abstraction consiste à ignorer quelques aspects d'un programme ou d'un algorithme étant visualisé, s' ils ne constituent pas un moyen efficace pour la compréhension d'algorithme. D'autre part, il consist également dans introduction à la visualisation quelques éléments, qui ne sont présents ni dans le programme exécuté ni dans ses données, mais ils aident à comprendre le logiciel.

CONTENU D'INTENTION – information supplémentaire sur la sémantique et la signification qui est derrière le code ; l'information orientée pour l'utilisateur de ce qui est important au sujet du programme.

ANIMATION – affichage séquentiel rapide des images, avec les images changeant graduellement au-dessus du temps, de sorte que les changements de vue de cadre à cadre sont assez petits, et la vitesse d'affichage est assez rapide pour obtenir une illusion du mouvement continu, comme dans dessin conventionnel animé.

ANIMATION DES ALGORITHMES – un type de visualisation de logiciel qui utilise à la manière significant la technique d'animation.

SYSTEM DE VISUALISATION DE LOGICIEL – système de logiciel spécialisé consacré pour créer et développer la visualisation de logiciel et pour conduire des projections.

SYSTEM D'ANIMATION DES ALGORITHMES – system de visualisation de logiciel consacré pour créer et développer les animations des algorithmes.

PROJECTION – processus de rendre la visualisation à un humain.

MACHINE PROJECTIVE – une partie d'un système de visualisation de logiciel laquelle conduit une projection.

VISUALISATEUR ou ANIMATEUR – utilisateur d'un système de visualisation (animation) de logiciel, qui prépare une projection en utilisant le système.

OBSERVATEUR – utilisateur d'un système de visualisation de logiciel, qui l'emploie pour observer une projection.

CONTRIBUTIONS

La contribution primaire de cette thèse est un modèle original d'animation des algorithmes basé sur **surveillance de flux des données**. Ce modèle a été réalisé dans le système d'animation des algorithmes *Daphnis*.

La méthode de visualisation des algorithmes proposée est une étape vers la création automatique des visualisations des algorithmes caractérisées avec un niveau élevé d'abstraction. Ces deux aspects, à savoir, abstraction et automation dans la visualisation de logiciel, sont par la plupart des auteurs indiqués comme profondément contradictoires : si la visualisation est davantagement abstrait, l'utilisateur doit mettre plus d'effort pour créer cette visualisation, et *vice versa*. Les vues abstraites de visualisation impliquent beaucoup d'information orienté pour l'utilisateur – le contenu d'intention – et un appui *a priori* de conception. La méthode proposée est un moyen pour partiel réconciliant les deux postulats contredits. Nous pouvons prouver, qu'elle introduit **certain niveau d'abstraction sans effort additionnel du visualisateur**. Dans les systèmes de visualisation de logiciel existant jusqu'ici, au moins ceux-qui nous sommes connus, ce niveau d'abstraction impliqué a exigé beaucoup d'effort impliquant précis, manuel concevant et l'accord de visualisation. Malgre cela, pour réaliser un niveau d'abstraction encore plus élevé, les équipements automatiques de notre solution ne sont pas suffisants.

Maintenant formulons notre thèse principale :

> En utilisant une méthode d'animation des algorithmes basée sur surveillance de flux des données, il est possible d'introduire automatiquement dans une visualisation de logiciel des éléments, qui augmentent de manière significative le niveau d'abstraction de cette visualisation.

La contribution originale est également une proposition d'une **architecture d'une chaîne des transformations**, par lesquelles les données obtenues à partir du programme visualisé

passent avant l'affichage. Cette architecture orientée objet, basée sur le modèle POST, peut être facilement configuré en utilisant un fichier manuscrit textuel ou des outils interactifs. Les manuscrits de configuration impliquent un niveau de contenu d'intention très élevé ; l'utilisateur peut définir les transformations des données du degré de complication qui implique normalement les modifications fait à main du texte source d'un programme. Tout ceci s'assure que la visualisation d'un niveau d'abstraction très élevé peut être obtenue avec les annotations minimales au texte source du programme, d'une maniere presque totalement **déclarative**. Quand le texte source est annoté, les arrangements de visualisation divers (y compris les visualisations de niveaux d'abstraction divers) peuvent être appliqués en utilisant un fichier manuscrit séparé pour chacun, sans le besoin de recompilation.

Nos contributions secondaires sont : une proposition d'attacher un système d'animation avec le programme étant visualisé, qui fait le système indépendant de langue de programmation, et l'architecture ouverte du système, qui, grâce à l'application de la technologie *ActiveX*, permet de la prolonger avec les objets définis par l'utilisateur.

ORGANISATION DE LA MEMOIRE

La redaction contient neuf chapitres. L'organisation de la partie suivante de ce sommaire reflète l'organisation de la redaction.

Les trois premiers chapitres sont une introduction large aux problèmes de la visualisation de logiciel et présentent l'etat actuel du problème. En chapitre 1 des définitions de base sont données. Le chapitre 1.2 contient les thèses principales (décrite dans la section précédente de ce sommaire). Le chapitre 2 contient une analyse plus profonde des problèmes les plus importants – de notre point de vue – de la visualisation de logiciel. Dans le chapitre 3 l'etat actuel du problème est présenté.

En chapitre 4 l'algorithme original pour l'animation des algorithmes basée sur la surveillance de flux des données est présenté. Trois principaux postulats pour obtenir une visualisation réussi et abstrait sont formulés, et on lui montre, comment ils peuvent être réalisés en utilisant notre méthode. On propose un modèle formel des flux des données utilisant le formalisme des réseaux de Pétri. Quelques versions des algorithmes sont décrites ; une d'entre elles, appelée tri-coloré de 1,5 pas, a été mise en application dans le système d'animation des algorithmes *Daphnis*.

Les quatre prochains chapitres ont pour sujet la description détaillée du système *Daphnis*. En chapitre 5 une description générale du système est donnée. Le chapitre 6 est consacré à l'environnement interactif de *Daphnis*. L'analyse de la conception d'interface d'utilisateur,

comme les applications potentielles et les besoins des utilisateurs du système sont présentés. L'architecture du système est le sujet du chapitre 7. Le chapitre 8 contient quelques exemples des animations.

Le chapitre final 9 est consacré aux remarques finales.

CHAPITRE 1 : INTRODUCTION

La compréhension de la voie, de laquelle les algorithmes fonctionnent, est une des demandes les plus significatives dirigées devant les créateurs de logiciel. C'est la partie essentielle d'éducation et de pratique d'ingénieurie en informatique.

La voie dans laquelle un algorithme est écrit a l'influence essentiel au degré comment peut-il être compris. La notation dans un langage de programmation formel est la plus précise, mais, en même temps, peu adapté au système humain de perception. Elle ne supporte pas le processus de compréhension d'un algorithme par un être humain. Une description écrit dans un langage naturel est beaucoup plus facile à comprendre (toutefois pas également précise), mais elle n'est toujours pas assez quand des algorithmes plus compliqués sont pris en considération. Ces deux méthodes de description sont textuelles : elles présentent un objet décrit analytiquement, pas à pas, mais ne supportent pas la vue synthétique, indispensable dans la compréhension du problème dans l'ensemble. Une description qui utilise une image est bien meilleure. Elle est beaucoup plus proche du modèle de perception humaine (plus vieux évolutionnaire que lingual), permet la vue synthétique du problème entier, sans exclusion de la possibilité de visualiser les détails. En observant, comment les enseignants, comme les ingénieurs d'informatique travaillent, nous notons, qu'ils emploient très souvent des schémas pour améliorer la compréhension des algorithmes. Les exemples des abstractions graphiques des algorithmes ce qu'ils utilisent sont les diagrammes blocs de flux de commande et images des structures des données. Ces schémas peuvent utiliser différents niveaux d'abstraction, aussi bien que les représentations graphiques spécifiques, liées aux réalités qui peuvent être observées, imaginations ou métaphores, facilitant la description du problème.

Dix ou plus d'années auparavant nous avions dans l'industrie informatique une période du développement sans précédent dans les solutions graphiques des interfaces homme-machine. Une manifestation à cette tendance dans l'enseignement d'informatique aussi bien que dans le développement de logiciel est la visualisation de logiciel. C'est un moyen d'explorer le comportement dynamique des algorithmes. La visualisation de logiciel donne aux utilisateurs un avis d'opération et de comportement des algorithmes, en utilisant la langue qui est la plus facile à comprendre par les humains – la langue de l'image. Elle présente des vues graphiques

multiples d'un algorithme dans l'action, exposant les propriétés que nous pourrions autrement trouver difficiles à comprendre ou même à remarquer.

Les quelques dernières années les problèmes de l'animation des algorithmes et des programmes ont attiré beaucoup d'attention. Beaucoup de systèmes logiciels fournissant l'équipement pour sa réalisation ont été développés. Le but de notre travail est de créer un système, qui permet de créer des animations de haute qualité en relativement peu de temps. Nous avons appelé ce système *Daphnis*.

CHAPITRE 2 : FONDAMENTS DE VISUALISATION DE LOGICIEL

Dans le chapitre 2 problèmes choisis de la visualisation de logiciel sont discutés.

Le contour général d'architecture, commun pour presque tous les systèmes de visualisation de logiciel implique trois étapes (fig. 2.1) :

- Rassemblement, collection et choix des données. Les données qui commandent la présentation entière doivent être obtenues à partir d'une partie de logiciel étant sujet de la visualisation.
- Traduction des données. Les données brutes de programme ne peuvent pas être employées pour l'affichage graphique sans une certaine transformation, au moins assurant la graduation nécessaire.
- Affichage des données. C'est ce qui fait de cette technologie une visualisation.

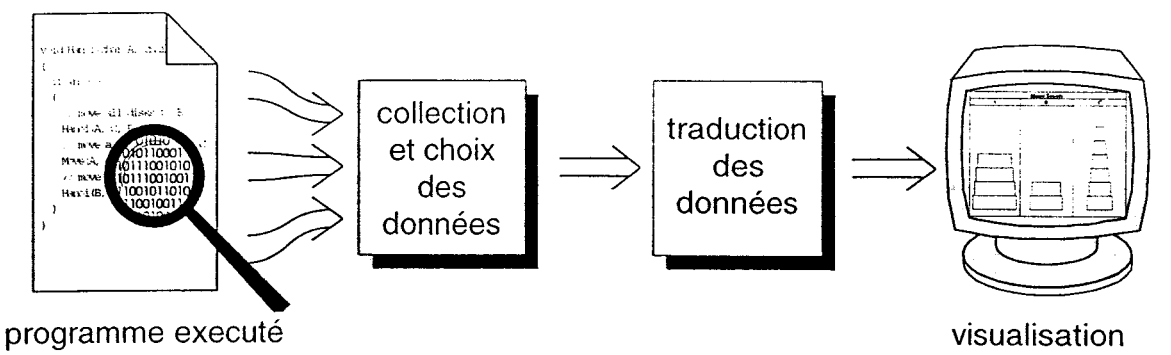


Fig. 2.1 Architecture simplifiée d'un système de visualisation de logiciel

Les moyens offerts par un système de visualisation de logiciel choisi assure qu'une visualisation de qualité normale peut être préparée d'une façon relativement facile. Pour obtenir une visualisation de qualité plus haute, l'utilisateur doit mettre plus de travail dans sa préparation.

La qualité (lisibilité, compréhensibilité, clarté) d'une visualisation compte beaucoup pour l'utilisateur qui doit observer ces projections. Elle est la plupart du temps influencée par deux facteurs. Le premier facteur est la possibilité de créer des projections à **niveau d'abstraction élevé**. Telles visualisations dépassent les applications isomorphes des structures des données ou code de programme a l'image graphique. Pour illustrer la sémantique d'un algorithme on doit montrer non seulement les éléments qui résultent de l'état dans lequel l'algorithme se trouve actuellement, mais aussi d'autres éléments, qui peut-être ne se produisent pas du tout explicitement de l'algorithme, et qui doivent être montrés. Le deuxième facteur est le **répertoire des représentations graphiques** offertes par le système. Il s'étend des techniques simples comme le texte accentuant jusqu'aux techniques multimédia avançés de graphique et d'animation 3D.

La facilité d'adapter des programmes pour projections sous un systéme est effectée par encore deux facteurs. Le premier d'entre eux est la **méthode de visualisation** appliquée dans le système. Cela implique le type de spécifications de visualisation aussi bien que des types divers de connexion systéme de visualisation – programme visualisé. Il inclut aussi les **annotations de text source** et autres éléments qui doivent être insérés au programme visualisé. Ils sont nécessaires pour informer le système sur la façon des changements de l'état du programme, et aussi comment ces changements doivent être illustrés. Le deuxième facteur est **l'aide** offerte par le système pour les utilisateurs qui prepare les visualisations. Ce facteur est souvent référé en littérature comme niveau d'automation, bien que le terme « l'aide » est meilleure, car quelques systèmes offrent des outils d'aide qui sont nullement automatiques.

Un autre facteur qui affecte fondamentalement tous les utilisateurs de la visualisation de logiciel est la **sphère des aspects du logiciel** étudiés par une visualisation individuelle. Enfin, **l'interface d'utilisateur** est un facteur, dont la qualité décide du sujet de confort de travail.

L'application de la visualisation de logiciel couvre principalement deux domaines :

1. Éducation informatique. C'est sans aucun doute le domaine d'application le plus réussi. Quelques études empiriques ont confirmé la valeur de la visualisation de logiciel comme aide éducative.
2. Création et developpement de logiciel. Dans ce secteur d'application la visualisation de logiciel est beaucoup moins réussie. Seulement un sous-ensemble limité d'équipements de visualisation a été appliqué dans des outils de développement employés couramment, et les pleines versions des systèmes de visualisation ne sont pas d'usage courant.

CHAPITRE 3 : ETAT ACTUEL DU PROBLEME

Chapitre 3 contient une revue des solutions existantes dans la visualisation de logiciel.

Bien que les anciens efforts dans le secteur qui aujourd'hui est appelé la visualisation de logiciel, remontent à plus de trente ans, le premier travail principal qui a fait l'inspiration pour tous les solutions postérieures, était un film *Sorting Out Sorting*. Historiquement parlant, les films algorithmiques comme ceux-ci sont les exemples les plus anciens des solutions de visualisation de logiciel.

Les dernières années les systèmes de visualisation des programmes et d'algorithmes constituent la direction principale du développement.

Les systèmes et les outils destinés à la visualisation de programmes ont un avantage distinct par rapport à la plupart des systèmes de visualisation d'algorithmes c'est-à-dire ils permetent facilement d'examiner le contenu des structures des données ou les autres aspects du logiciel visualisé sans effort, comme celuiqui est impliqué dans la construction d'une visualisation d'algorithme. Bien qu'ils sont relativement limités en comparaison des systèmes avançés d'animation d'algorithmes, ils offrent souvent les dispositifs entièrement automatiques. En fait ils créent le seul secteur de la visualisation de logiciel qui a des applications commerciales réussies, particulièrement dans l'élaboration des programmes et des environnements de correction. Suivants les systèmes et outils de la visualisation des programmes revues dans la redaction :

- SEE et autres outils de joli impression (ang. *pretty-printing*) pour la visualisation de code source.
- Incense – le premier système qui a automatiquement montré des vues des structures des données. Ses successeurs – Amethist et Pascal Genie – sont également décrits.
- Microsoft Visual C++ – un d'entre beaucoup d'outils de développement de logiciel disponibles dans le commerce qui emploient efficacement des technologies visuelles, y compris quelques formes de visualisation de logiciel.

La revue dans Chapitre 4 se concentre sur les systèmes de visualisation des algorithmes, principalement ceux-ci sont consacrés pour visualiser des programmes conventionnels, écrits en langues impératifs. Quelques systèmes d'intérêt spécial de notre point de vue sont discutés en détail. Ceux-ci sont :

- Le système BALSA était non seulement le premier système d'animation, mais également le meilleur système connu jamais fait. Dans beaucoup d'aspects BALSA et ses successeurs, BALSA II et Zeus, sont encore sans concurrence.

- Le système Tango était une contribution très importante dans le développement de la visualisation de logiciel. Grâce à l'application d'un nouveau paradigme de chemin-transition (ang. path-transition paradigm), pour la première fois les transitions parfaitement animées ont pu être incorporée à une animation d'algorithme. Tango permet de produire l'animation de qualité de dessins animés sans frais d'écriture de code spécifique pour chaque étape de transition. Les successeurs du système Tango s'appellent XTango et POLKA.
- Le système ANIM a été décrit dans la redaction pour son approche originale de créer des visualisations de logiciel. C'est un système *post mortem* dans lequel les animations sont fabriqué totalement à la main, en utilisant une interface généralisée simple, indépendant de langue et logiciel utilisées.
- Le système Animus, destiné pour la visualisation des programmes écrits en Smalltalk, a constitué le premièr étape sérieus vers la creation d'un système de visualisation dans lequel le modèle de spécification de projection est totalement déclaratif. Bien que les dispositifs de ce système aient été limités, il a apporté une contribution importante.
- Le système UWPI a un excellent mécanisme pour génération automatique des abstractions. Il fournit des visualisations pour les structures des données abstraites à niveau élevé, créés par le programmeur.
- Le système Pavane représente la situation actuelle dans la visualisation des algorithmes. Il est conçu pour fournir des visualisations tridimensionnelles déclaratives des programmes concourants. Il offre le répertoire des moyens graphiques très riche, et il est l'un de peu de systèmes dans lesquels le modèle de spécification de projection est totalement déclaratif.
- Le système Whizz est un outil pour établir les interfaces homme-machine animées. L'outil Witness constitue une des applications de Whizz, qui est un système d'animation des algorithmes basé sur une rêgle des objets communiquant avec des jets des données et d'événements.
- Le systèmes SANAL et WinSANAL, à la production desquels l'auteur de cette redaction a participé, sont les prédécesseurs directs du système *Daphnis*.
- Le système Siamoa, développé par nos collègues de l'Uuniversité de Lille 1, a constitué pour nous une inspiration et expérience importante.

En suite, quelques systèmes pour visualisation des aspects non impératifs des programmes sont presentés. Ceci inclut :

- Systèmes de visualisation pour langues fonctionnels,
- Systèmes de visualisation pour langues declaratifs,

- Systèmes de visualisation pour langues orienté objet,
- Systèmes de visualisation pour calcul parallèle.

Les conclusions de Chapitre 3 sont comme suit :

- La difficulté de préparation de visualisation dépend du niveau de l'abstraction désiré et de l'architecture du système appliquée. La préparation de visualisation est généralement plus facile dans les systèmes desquels le modèle de spécification est déclaratif.
- Ce qui manque aux systèmes de visualisation de logiciel est un dispositif appelé **l'intelligence**. C'est un service des génération automatique des abstractions. Un certain niveau d'intelligence peut être trouvé dans le système UWPI. Notre but est de créer une méthode d'animation des algorithmes qui augmente l'intelligence du système : il devrait faciliter la creation des vues abstraites des algorithmes.

CHAPITRE 4 : ANIMATION DES ALGORITHMES BASEE SUR LA SURVEILLANCE DU FLUX DES DONNEES

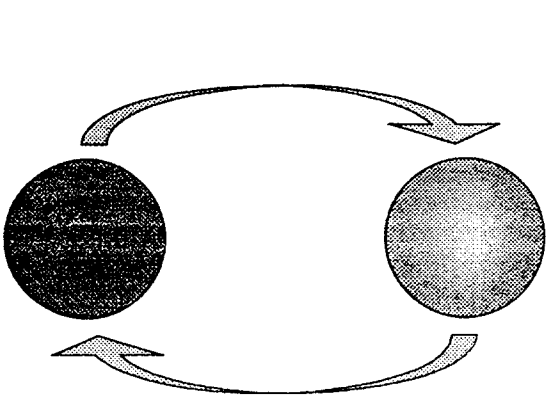
Le chapitre 4 est consacré à la méthode d'animation des algorithmes basée sur la surveillance de flux des données. Nous supposons que c'est la contribution la plus originale et la plus importante de notre travail.

NIVEAUX DE DESCRIPTION DES ALGORITHMES

Au début du chapitre, divers niveaux de description de processus d'exécution d'algorithme sont discutées. Ceux-ci sont :

- métaphore du monde réel (fig. 4.1a),
- description dans un langage formel (fig. 4.1b),
- séquence des valeurs consécutives de variables individuels (fig. 4.1c),
- transmissions observées au niveau des circuits électroniques (fig. 4.1d–f).

Sur divers niveaux d'analyse, divers solutions du même problème peuvent être realisés. La solution synchrone qui est évidente dans la métaphore du monde réel, n'est pas possible à réaliser dans un langage de programmation séquentiel/asynchrone. Une variable auxiliaire nécessaire dans le langage C est excessive dans une solution optimisée de langage assembleur. Le changement rapide des teneurs des registres qui sont imprévisibles au niveau de l'analyse de contenu des registres est une conséquence simple de transmission électrique de signal, quand l'analyse de niveau de circuits est concernée.

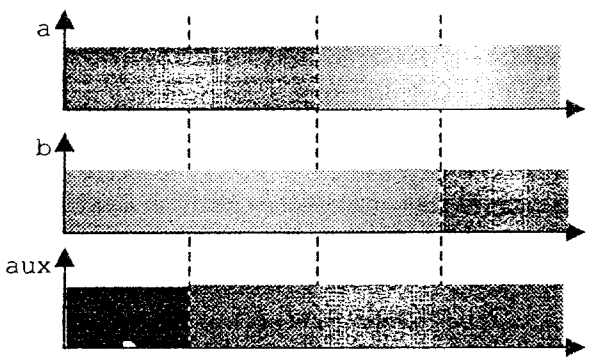


a

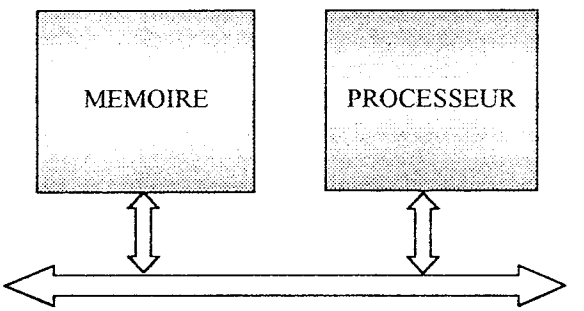
```
aux = a;
a = b;
b = aux;

mov  AX, a
mov  BX, b
mov  a, BX
mov  b, AX
```

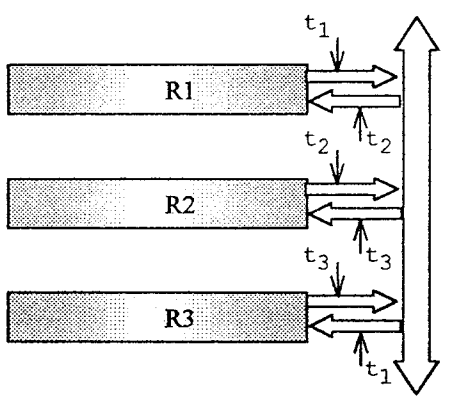
b



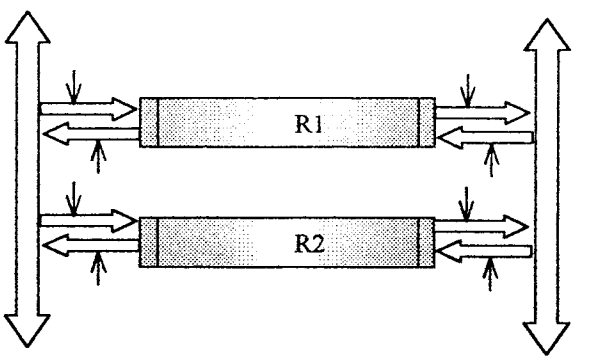
c



d



e



f

Fig. 4.1 Divers niveaux de description d'une opération d'échange de deux valeurs

Sans aucun doute, de tous les niveaux d'analyse présentés ci-dessus, le niveau des valeurs temporalles de variables/registres est celui de la moindre valeur cognitive. Cependant, du point de vue technique, il est le plus facile à obtenir ; c'est pourquoi la plupart des outils de correction traditionnels opèrent dans ce niveau. Ce n'est pas un bon point de sortie pour la

visualisation des algorithmes : il pourrait montrer à l'utilisateur ce qui passe, mais il ne pourrait pas expliquer pourquoi. Il n'y a aucune information sur la source et la destination des données, et aucune information sur ce qui est la cause du changement des données.

TROIS POSTULATS D'ANIMATION REUSSIE DES ALGORITHMES

Recherchant un meilleur modèle pour l'animation des algorithmes un postulat principal de la visualisation réussie a été formulé : le postulat d'illustration de flux des données. Au cours des recherche nous avons encore trouvé deux postulats, nécessaires pour une visualisation soutenant avec succès la compréhension d'un algorithme.

Au-dessous ces trois postulats, ou condistions d'animation des algorithmes réussie, sont discutés :

- **Postulat de l'illustration de flux des données** : la visualisation devrait appliquer l'animation coulante pour rendre les opérations de flux des données, qui se produisent pendant le processus d'exécution d'algorithme. Une bonne visualisation doit illustrer non seulement les états que le programme passe pendant l'exécution, mais également comment la transformation entre les états se produit. Une manière d'obtenir cela est une visualisation dans laquelle la représentation graphique des données est animée coulante de l'endroit représentant la source des données vers l'endroit représentant la destination. Elle traverse une série de positions intermédiaires qui n'ont aucune correspondance à l'état réel ni du programme ni de ses données, discernable à l'aide des outils de correction traditionnels. Les objets graphiques ne représentent pas tellement les variables, mais plutôt les données persistantes coulantes entre elles.
- **Postulat de la suppression spatiale d'information** : la visualisation devrait supprimer l'information au sujet des variables qui sont sans importance, gardant l'uniformité et la concordance de la vue globale de flux des données. Une visualisation qui se conforme à ce postulat a pour sujet une réalisation d'algorithme autre que le réel, dans lequel il y a moins des variables et des opérations de flux des données. L'objet de telle suppression sont les éléments, qui ne facilite pas la compréhension d'algorithme ; ce sont simplement les variables qui n'ont pas été indiquées par le visualisateur comme les variables importantes. Mais, l'ignoration des variables non importantes mènerait souvent à cesser la concordance de l'image globale de flux des données dans le programme. Souvent les variables qualifiés pour être sans importance font un étape intermédiaire de flux des données plus larges qui commencent et finissent à l'espace d'intérêt de l'utilisateur. Le postulat de la suppression spatiale implique cependant le besoin de la reconstruction des séries cassés de flux des

données. Les opérations élémentaires de flux, dont les variables des sources ou des destinations sont en dehors de l'espace d'intérêt de l'utilisateur, doivent être collées pour former, si possible, une opération nouvelle de longue distance, dans laquelle les variables de source et destination sont importantes du point de vue de l'utilisateur.

- **Postulat de la suppression temporelle d'information** : une partie de l'information concernant la commande en temps des opérations élémentaires doit être enlevée de la projection. En particulier, quelques opérations qui sont réellement exécutées séquentiellement peuvent être rendues comme si elles ont été exécutées en parallèle et synchroniquement. Ainsi, la réalisation rendue dans la visualisation diffère de celle réellement appliquée dans le logiciel visualisé. Les informations détaillées sur la commande de temps des opérations qui sont indépendantes et exécutés dans la période courte, ne facilitent pas la compréhension d'algorithme, mais – au contraire – ils attirent l'attention de l'observateur loin d'autres éléments, beaucoup plus importants.

Les visualisations qui satisfont tous les trois postulats caractérisent avec le niveau d'abstraction relativement élevé. La plupart des visualisations existantes vont dans cette direction ; des bons exemples peuvent être nombreux, comme celles crées littéralement pour tous les systèmes existant, visualisations des algorithmes de tri : les opérations élémentaires d'échange de valeur presque toujours sont illustrées comme processus symétrique et synchrone de permuter deux objets de représentation graphique, sans utilisation de variable auxiliaire. Dans les systèmes existants jusqu'ici, obtenant une visualisation qui satisfait les trois postulats a impliqué une conception manuelle. La méthode d'animation des algorithmes proposée, décrite ci-dessous, permet d'obtenir de telles visualisations automatiquement.

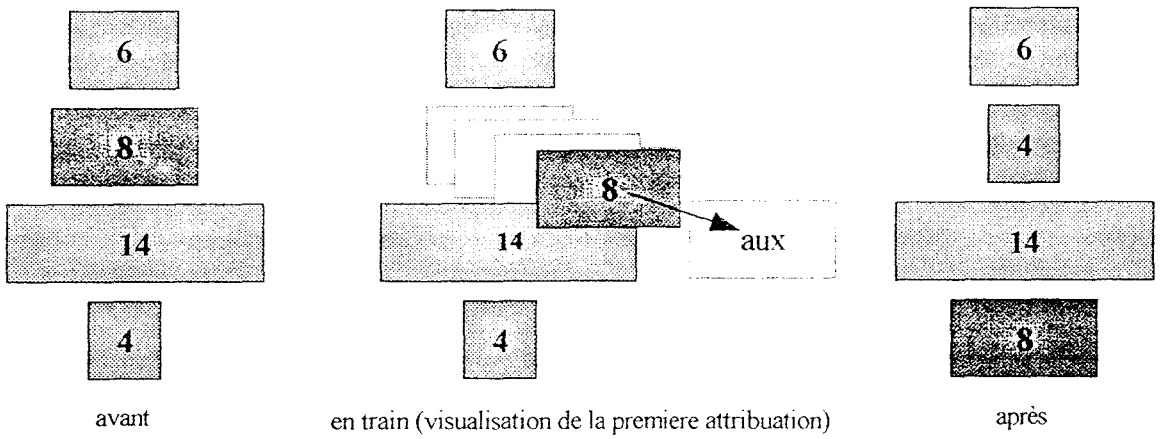
DESCRIPTION D'EXECUTION D'ALGORITHME EN TERMES DE FLUX DES DONNEES

Le processus de l'exécution d'algorithme, ou plus précisement, l'exécution d'un programme qui réalise un algorithme, peut être efficacement décrit comme ordre des operations des flux des données. Cela ne devrait pas être confondu avec une description d'un algorithme elle-même ; notre modèle indique seulement ce qui se passe pendant l'exécution du programme et ne peut pas être employé comme outil pour spécifications des algorithmes.

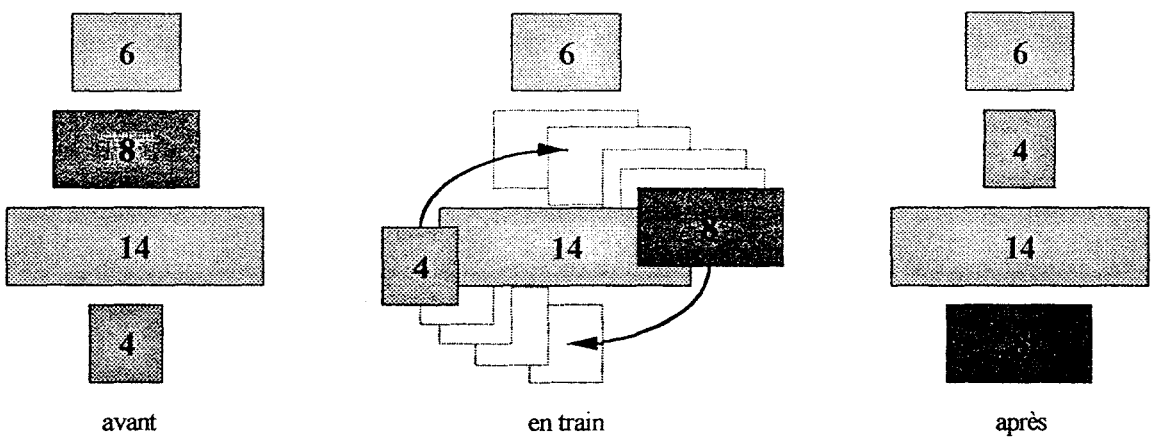
Les termes cruciales de ce modèle sont :

- **Variable** – un conglomérat du nom et du type attribué, et sur l'option d'une valeur. La variable qui a une valeur s'appelle **évalué**.

a) visualisation statique



b) animation coulante – l'illustration de flux des données



c) animation coulante sans la variable auxiliaire

Fig. 4.2 Visualisations exemplaires de l'opération de l'echange de deux valeurs

- Acte **d'évaluation** d'une variable – donner une (nouvelle) valeur à la variable. Normalement les variables n'ont aucune valeur avant qu'elles ne soient pas évalué pour la première fois (exception : variables gardant des paramètres d'entrée).
- **Flux des données** – un phénomène qui cause l'évaluation d'une variable de destination par une valeur défini par un ensemble de zéro ou plus variables de source. Pendant l'exécution d'un programme typique, l'ensemble des variables évaluées se prolonge pendant que les flux des données consécutifs se produisent.

UNE MACHINE POUR ANIMATION DES ALGORITHMES CONDUITE PAR FLUX DES DONNEES

Quand le processus d'exécution d'algorithme peut être décrit en termes de flux des données, nous pouvons créer une machine pour l'animation des algorithmes conduite par le flux des données. Cette machine satisfait le postulat d'illustration des flux des données.

Les outils traditionnels de correction et de visualisation obtiennent des données du programme observé dans la forme d'un séquence des valeurs temporaires des variables individuelles. Dans la machine proposée, c'est plutôt une série d'information sur les opérations élémentaires des flux des données qui forment l'entrée pour le système.

L'animation est maintenue dans le mouvement par une série d'appels consécutifs aux routines rapportant les opérations élémentaires des flux des données. Avant que la variable soit visualisée, elle doit être enregistrée au système, de sorte qu'elle puisse être identifiée et tracée ainsi. Après la visualisation les variables devraient être retirées du registre. Ceci doit se produire avant que les variables cessent d'exister. Ceci forme les trois opérations élémentaires pour notre machine d'animation :

- **Register** – commence la visualisation individuelle d'une variable.
- **Play** – informe le système que les opérations élémentaires des flux des données ont eu lieu.
- **UnRegister** – informe le système qu'une variable individuelle ne doit plus être visualisé.

Ces trois opérations sont les seules fonctions, avec lesquelles le texte source du programme étant visualisé doit être annoté. Il est remarquable que notre fonction *Play* informe le système au sujet des flux des données, pas de changements des données – au contraire d'autres outils de correction et de visualisation.

L'utilisateur du système (le visualisateur) fournit un fichier externe de manuscrit de configuration, qui indique la représentation graphique pour chaque variable visualisé, et aussi bien les règles qui definissent comment ses valeurs temporaires consécutives doivent être traduites pour commander différents attributs d'image finale. Ces attributs peuvent dépendre des

valeurs ou des adresses des variables. Pendant la visualisation d'une opération de flux des données, les éléments graphiques sont reliés plutôt aux valeurs stockées dans les variables, qu'aux variables. Non seulement la valeur, mais aussi son adresse peut changer – en d'autres termes, ainsi qu'avec les valeurs des données, aussi ses représentations graphiques coulent d'une variable à l'autre. Habituellement le changement de la valeur d'une variable est rendu comme déformation de sa représentation graphique, alors que le changement d'adresse d'une valeur (une opération de flux des données) est rendu comme un mouvement d'objet graphique de la position représentant l'adresse de source à la position représentant l'adresse de destination. Il est simplement réalisé en reliant l'adresse aux attributs définissant la position de l'objet (x et y).

APPLICATION DES RESAUX DE PETRI POUR LA DESCRIPTION DU COMPORTEMENT DES ALGORITHMES

Le moteur proposé dans la section précédente rencontre le premier des trois postulats de l'animation des algorithmes réussie : illustration des flux des données. Pour rencontrer les deux autres postulats, c.-à-d. suppression spatiale et temporelle d'information, nous avons dû employer le formalisme de réseau de Pétri, et puis décrire le processus de l'exécution d'algorithme utilisant ce formalisme. Ceci a laissé préparer la version finale de la méthode d'animation des algorithmes, qui satisfait tous les trois postulats.

Description de systèmes conduits par le flux des données

La description des algorithmes dans des ordinateurs conduits par le flux des données est relativement facile. Le modèle théorique d'un tel ordinateur présente une structure fortement parallèle, qui se compose des processeurs multiples (fig. 4.3 a). Chaque processeur possède un certain nombre d'entrées des données, représentant des arguments, et un certain nombre de sorties des données, représentant des résultats, et il est prévu pour calculer les résultats dans l'accord aux arguments. Un réseau de Pétri, dans lequel les endroits représentent les lignes des données et les transitions représente les processeurs peut représenter un tel système (fig. 4.3b). Les ensembles d'entrée des transitions reflètent les ensembles des données d'entrée du processeur. Les ensembles de sortie des transitions reflètent les ensembles des données de sortie du processeur. L'indication du réseau reflète l'évaluation des données représentées par les endroits. Tous les endroits qui représentent des données èvaluées sont marqués. Le tirage de transition reflèt dans ce modèle l'opération elementaire du flux des données.

Le modèle de réseau de Pétri reflète très bien deux caractéristiques fondamentales d'un système conduit par le flux des données : le parallélisme d'exécution et le non-déterminisme de l'ordre temporel des calculs.

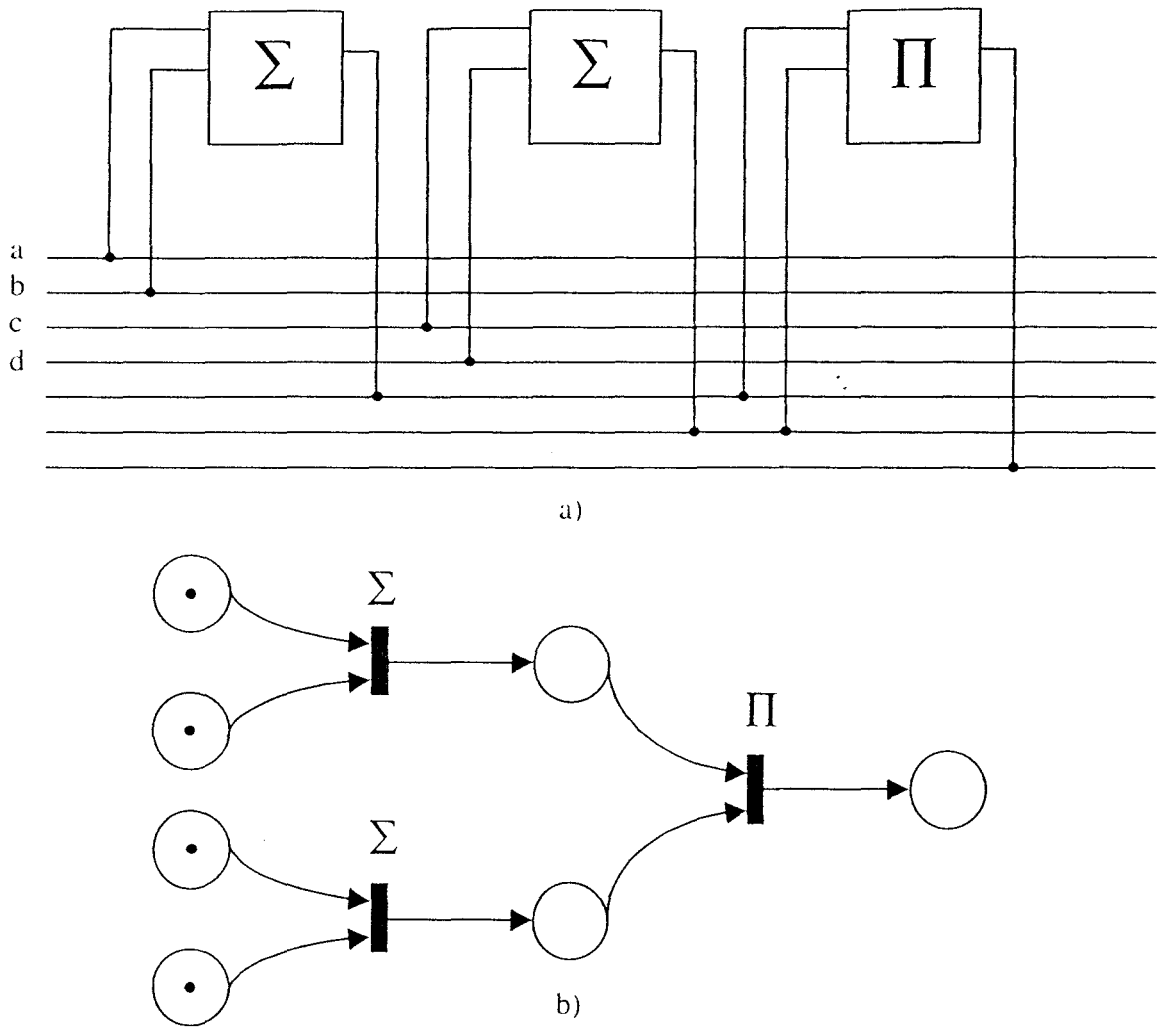


Fig. 4.3 Un système dirigé par flux des données qui calcule la valeur d'expression : *(a+b)*(c+d)* : a) schéma ; b) représentation sous la forme d'un réseau de Pétri

Description de systèmes conduits par le flux des opérations

Des systèmes informatiques traditionnels sont conduits par le flux d'opérations plutôt que des données. L'ordre des opérations est strictement déterminé, et le niveau du parallélisme est relativement bas : si nous considérons des ordres courts des opérations, dans la plupart des cas il n'y a aucun parallélisme.

Nous pouvons représenter un système conduit par le flux des opérations avec un réseau de Pétri dans lequel les endroits représentent des variables, et les transitions représentent des opérations élémentaires des flux des données. Quelques exemples sont montrés sur fig. 4.4.

L'ordre des opérations des flux des données dans un programme traditionnel est strictement déterminé. Ainsi le tirage des transitions est également strictement déterminées à temps. Cependant, dans les réseaux de Pétri typiques le temps de tirage n'est pas

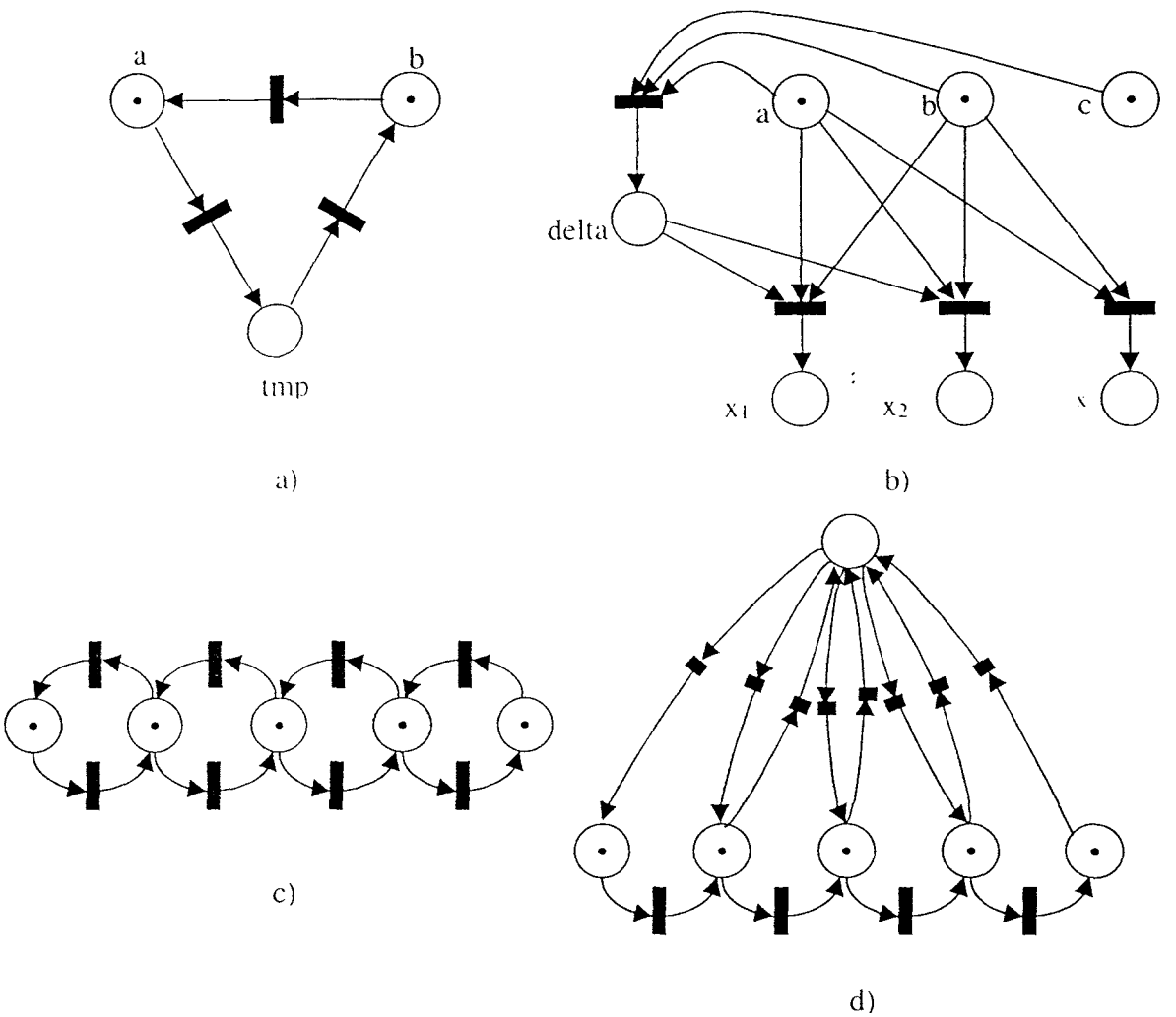


Fig. 4.4 : Flux des données dans quelques algorithmes exemplaires, représentés avec réseaux de Pétri : a) échange de deux valeurs ; b) carrée équation ; c) vue simplifiée de tri de bulle ; d) vue de tri de bulle avec une variable auxiliaire.

déterminé. Un modèle basé sur un tel réseau ne permet pas de définir l'ordre des opérations exécutées à temps. En plus, dans tel réseau chaque transition permise peut être mise au tirage. Au contraire, dans une algorithme, l'exécution d'une opération individuelle est déterminée dans les conditions additionnelles définies dans cet algorithme. Par exemple, dans un algorithme de tri, presque chaque flux entre deux éléments quelconques de la rangée est possible, et le modèle de réseau de Pétri les implique ainsi tous. Cependant, pour assortir avec succès une rangée, seulement un sous-ensemble de tous les ordres possibles des flux des données peut être appliqué, et dans un ordre déterminé. Pour faire ce modèle compatible avec le vrai processus, nous devons employer le réseau de Pétri **non autonome**, ou **synchronisé.** Tels réseaux décrivent les systèmes dont l'évolution est conditionnée avec des événements externes, se produisant à temps. Le tirage d'une transition est possible seulement si un

événement externe se produit. Dans notre modèle, un tel événement est l'exécution d'opération élémentaire de flux des données.

Au contraire des systèmes conduits par le flux des données, le modèle de réseau de Pétri ne reflète pas bien les systèmes conduits par le flux des opérations. Sans déterminer l'ordre temporel des tirages des transitions dans un resau non-autonome ce modèle n'est pas précis.

LE PAS D'ALGORITHME ET SON PROPRIETES

Dans cette section un modèle mathématique est proposé. Ce modèle est indirectement utilisé pour formuler la version finale de la méthode d'animation des algorithmes, qui illustrent non seulement des flux des données, mais satisfait également les postulats de la suppression spatiale et temporelle d'information. La description du comportement d'un système conduit par le flux des opérations, en utilisant un réseau de Pétri autonome est présentée. En raison des dépendances temporelles qui sont caractéristiques pour cette classe des systèmes, le processus entier de l'exécution d'algorithme ne peut pas être décrit, utilisant ce modèle. Ainsi, la description sera limitée à un segment d'algorithme, appelé un **pas d'algorithme**. Cette limite est fondamentale pour la suppression temporelle d'information. On proposera également une transformation réductrice des réseaux de Pétri, qui sert de base à la suppression spatiale d'information.

Supposons qu'un algorithme est mis en application dans un environnement multiprocesseur, dans lequel un nombre d'opérations consécutives peut être exécuté en parallèle. En raison de la sémantique d'algorithme, quelques opérations doivent être exécutées après que quelques autres opérations soient terminées. Ainsi, le processus entier de l'exécution d'algorithme peut être divisé en quelques étapes, pendant lesquelles des sous-ensembles consécutifs d'opérations sont (ou peuvent être) exécutés en parallèle. Chacun de tels sous-ensembles s'appelle un **pas d'algorithme**. Ce terme est également employé à l'exécution séquentielle des algorithmes : le fait d'appartenir plusieurs opérations au même étape indique la possibilité potentielle de les exécuter en mode parallèl.

Si nous supposons, que les opérations d'un pas d'algorithme sont exécutées non seulement en parallèle, mais également synchroniquement, nous pouvons augmenter ce pas à plus d'opérations. Cela ce fait parce que quelques opérations peuvent être exécutées synchroniquement – sans changement de la signification d'un algorithme – mais elles ne peuvent pas être exécutées asynchroniquement. Une étape d'algorithme augmentée de cette façon s'appelle **pas synchrone d'algorithme**.

Au-dessous quelques définitions formelles et lemmes importants sont présentés en bref :

- **Pas d'un algorithme** (définition 4.6) est une telle séquence des opérations d'algorithme exécutées en succession qu'un changement de la réalisation de cet algorithme est possible, qui consist à l'exécution des ces opération dans l'ordre non déterminé, ou en parallèle (sans le changement sémantique).
- **Pas synchrone d'un algorithme** (définition 4.7) est une telle séquence des opérations d'algorithme exécutées en succession qu'un changement de la réalisation de cet algorithme est possible, qui consist à l'exécution des ces opération en parallèle, a la manière synchrone (sans le changement sémantique).
- Que *PN (P, T, F)* soit un réseau de Pétri qui modèle une certaine séquence des opérations exécutées en succession par un système dirigé par flux des opérations. Qu'une fonction $\tau(t)$ indique le temps de tirage de la transition t. Que $^*t$ soit l'ensemble d'entrée d'une transition t, et t^* soit l'ensemble de rendement d'une transition t (définition 4.8) :
  1. Opérations représentées par les transitions $t_1, t_2 \in T$, où $\tau(t_1) < \tau(t_2)$, s'appellent **dépendant**, si $t_1^* \cap {}^*t_2 \neq \varnothing$. On dit qu'il y a une relation de dépendance entre eux (fig. 4.5a).
  2. Opérations représentées par les transitions $t_1, t_2 \in T$ s'appellent **séquentiel**, si $t_1^* \cap t_2^* \neq \varnothing$. On dit qu'il y a une relation de séquence entre eux (rys. 4.5b).

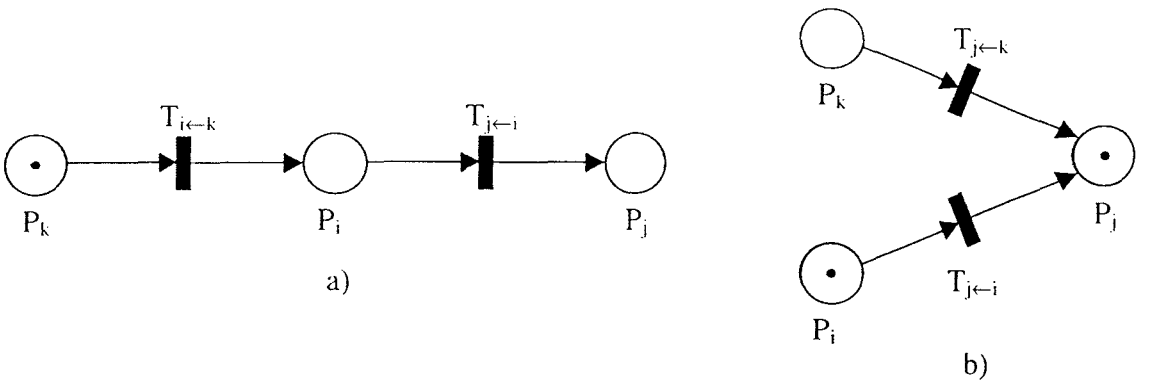


Fig. 4.5 Cas, dans lesquels une nouvelle transition $T_{j\leftarrow i}$ ne peut pas être mise au tirage en parallèle : a) dépendance ; b) séquence

- **Lemme de pas synchrone** (lemme 4.3) **:** Les opérations appartiennent à la même étape synchrone d'un algorithme quand, et seulement si, elles ne sont ni dépendantes ni séquentielles.
- Que *PN(P, N, F)* soit un réseau et que $\{p_i \rightarrow t_{x\leftarrow i} \rightarrow p_x\} \cup \{p_x \rightarrow t_{j\leftarrow x} \rightarrow p_j\} \in PN$ (fig. 4.6, à gauche). **Collage P_i – P_j autour de P_x** (définition 4.10) nous appelons une telle

transformation du réseau *PN*, dans le résultat duquel un réseau *PN'* est créé, dans lequel l'opération $\{p_i \rightarrow t_{j\leftarrow i} \rightarrow p_j\}$ remplacée par T $\{p_i \rightarrow t_{j\leftarrow i} \rightarrow p_j\}$ (fig. 4.6, à droite)* :

$$PN' = PN - \{p_i \rightarrow t_{j\leftarrow i} \rightarrow p_j\} \cup \{p_i \rightarrow t_{j\leftarrow i} \rightarrow p_j\}$$

- Que *PN(P, N, F)* soit un réseau et que $\{p \rightarrow t_{x\leftarrow i} \rightarrow p_x\} \cup \{p_k \rightarrow t_{x\leftarrow k} \rightarrow p_x\} \in PN$ (fig. 4.7, à gauche). **Interception $P_i - P_x$ par P_k** (définition 4.11) nous appelons une telle transformation du réseau *PN*, dans le résultat duquel un réseau *PN'* est créé, dans lequel l'opération $\{p_i \rightarrow t_{x\leftarrow i} \rightarrow p_x\}$ a été enlevé (fig. 4.7, à droite)* :

$$PN' = PN - \{p_i \rightarrow t_{x\leftarrow i} \rightarrow p_x\}$$

- **Variable sans importance** (définition 4.12) est une variable qui est hors de l'espace d'intérêt de l'utilisateur – l'observateur, et c'est pourquoi il manque la représentation graphique.
- **Lemmes des transformations spatiales** (lemmess 4.4 et 4.5). Ces deux lemmes indiquent que les transformations de collage $P_i - P_j$ autour P_x et d'interception $P_i - P_x$ par P_k ne change pas la sémantique de l'algorithme représenté par le réseau de Pétri transformé, dans la condition – pour les deux cas – que P_x représente une variable sans importance.

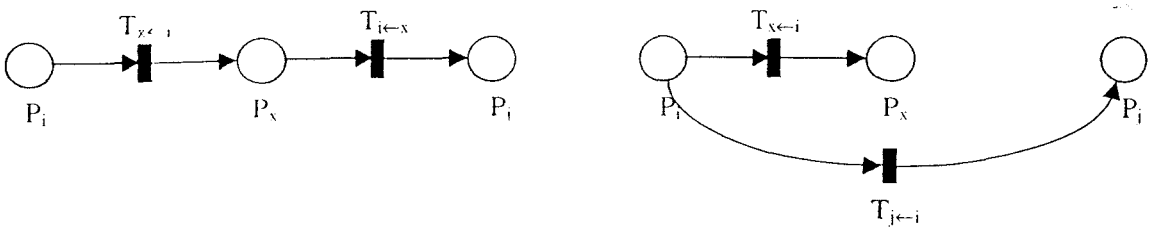


Fig. 4.6 Transformation de collage $P_i - P_j$ autour de P_x

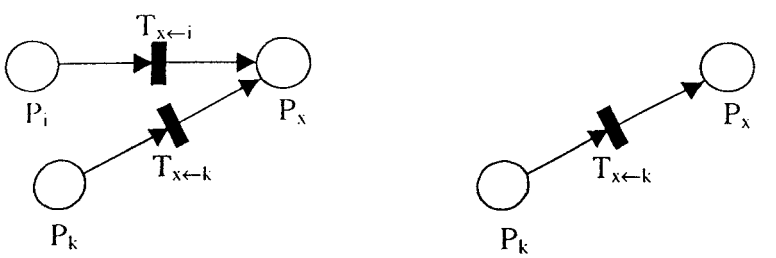


Fig. 4.7 Transformation d'interception $P_i - P_x$ par P_k

---

* Nous employons une notation dans laquelle $\{p_i \rightarrow t_{j\leftarrow i} \rightarrow p_j\}$ indique un sous-ensemble de réseau de Pétri *PN(P, T, F)* qui se compose de deux endroits p_i et p_j, et une transition $t_{j\leftarrow i}$, si $F = \{(p_i, t_{j\leftarrow i}), (t_{j\leftarrow i}, p_j)\}$, où *F* est relation d'écoulement de *PN*. Ce sous-ensemble représente une opération de flux des données avec la source représentée par p_i et les destinations représentés par p_j.

ALGORITHMES POUR VISUALISATION DES ALGORITHMES BASEES SUR LE FORMALISME DE RESEAUX DE PETRI

Quand le modèle mathématique a été créé, la préparation des versions finales de l'algorithme de l'animation des algorithmes est relativement facile. Trois versions de cet algorithme ont été définies :

- **Algorithme d'une couleur et d'un pas**. L'idée principale est rassembler – pendant l'exécution de l'algorithme – les opérations consécutives qui appartiennent au même pas synchrone de l'algorithme, et de les visualiser toutes après terminaison de ce pas, de la façon synchrone. Pour détecter si une nouvelle opération appartient au pas courant, le lemme de pas synchrone a été appliqué. Cet algorithme satisfait le postulat de la suppression temporelle d'information, mais ne satisfait pas le postulat de la suppression spatiale d'information.
- **Algorithme tri-coloré d'un pas**. Cet algorithme est basé sur le précédent. D'abord, tous les endroits représentant les variables importantes sont peints noirs, les endroits représentant les variables sans importance qui sont destinations flux des données venants de variables importantes ("noir") sont peint gris (fig. 4.8), et tout le reste de variables sans importance sont peints blanc. L'idée principale est d'appliquer la transformation de collage de réseau de Pétri pour coller quelques opérations des flux à une opération nouvelle longue, qui vont d'un endroit noir, venu à un endroit noir, mais a tous les endroits intermédiaires gris – bien que les variables représentées par elles-mêmes sont sans importance (fig. 4.9a). Une autre idée est de supprimer quelques opérations des flux qui ont une variable sans importance comme sa destination, en utilisant la transformation de l'interception (fig. 4.9b). Cet algorithme satisfait les postulats de la suppression temporelle et spatiale d'information.
- **Algorithme tri-coloré de 1,5 pas**. C'est l'amélioration de l'algorithme précédent, dans lequelle quelques chaînes d'opération peuvent être collées à travers les limites des pas synchrones consécutifs d'algorithme. Cette version de l'algorithme a été mise en application dans le système d'animation des algorithmes *Daphnis*.

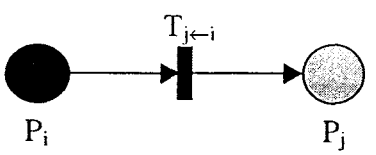


Fig. 4.8 La règle de la peinture place le gris dans l'algorithme tri-coloré

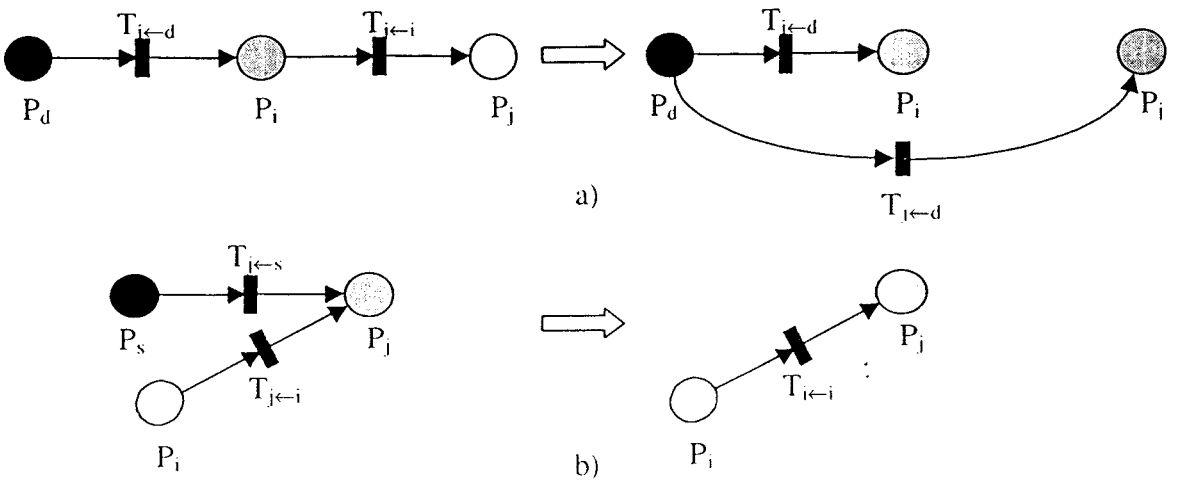


Fig. 4.9 Cas spéciaux dans l'algorithme tri-coloré et la manière de leur réduction

CONCLUSION DU CHAPITRE 4

Les postulats d'illustration des flux des données, comme de la suppression temporelle et spatiale d'information décident de la fonctionnalité de la visualisation : ils sont essentiels à la question de faciliter la compréhension des algorithmes. Le fait, dont la majorité d'animations des algorithmes préparée avec les systèmes de visualisation existants jusqu'ici, plus ou moins satisfont ces postulats, parle des volumes pour leur importance. Jusqu'ici, le développement d'une telle visualisation a impliqué la conception main-ouvrée. L'algorithme tri-coloré de 1,5 pas, mis en application dans le système *Daphnis*, permet d'obtenir sans effort une visualisation préliminaire qui est brute, mais satisfait tous les trois postulats. Nos expériences avec le système *Daphnis* ont prouvé que d'une façon rapide et semi-automatique nous pouvons obtenir les visualisations préliminaires qui reflètent très exactement la sémantique d'algorithme.

CHAPITRE 5 : DESCRIPTION GENERALE DU SYSTEME *DAPHNIS*

Le nom du système est une abréviation librement traitée de *DAta-Flow-driven aNImation System*. Travaux de développement sur le système *Daphnis* commencé en 1997. C'est un successeur des systèmes précédents, *SANAL* et *WinSANAL*. Une inspiration importante était également un système de prototype *Siamoa*, créé à l'Université de Lille 1. *Daphnis* est une solution complètement nouvelle, bien que quelques de ses éléments soient venus de *WinSANAL*, et même une partie du texte source a été réutilisée. *Daphnis* se distingue de son prédécesseur avec son architecture interne reconstruite. L'innovation la plus importante est l'application de la méthode d'animation des algorithmes basée sur la surveillance de flux des données.

Daphnis est un système d'animation des algorithmes d'application général. Cela veut dire, qu'il est capable de visualiser l'action de n'importe quelle classe d'algorithmes. Le système

fonctionne d'aprés les système *Windows 9x/NT 4.0/2000* ou plus nouveaux, sur des ordinateurs avec des processeurs d'Intel. Les besoins en matériel minimaux sont comme pour le système d'exploitation proportionné, mais *Pentium* ou meilleur processeur est recommandé.

Le système *Daphnis* est généralement indépendant de la langue, dans laquelle l'algorithme visualisé est formulé. Le programme étant visualisé doit être un module de plateforme des *Windows 32*, capable pour accéder aux bibliothèques de lien dynamiques (*dll*). L'interface de programmeur (*API*) a été créée pour faire la machine projective du système accessible pour des programmes écrits dans C et C++, ainsi la préparation des programmes pour la projection est particulièrement facile si elles sont écrites dans un de ces langues. Employer d'autres langues exige l'application des techniques de déclaration d'interface appropriées à ces langues. Jusqu'ici, nous avons créé quelques projections basées sur des programmes écrits en *Visual Basic*.

Daphnis est un système conduit par les données, avec méthode déclarative de spécifications de visualisation. Quelques annotations du texte source des programmes sont encore nécessaires, ainsi le modèle des spécifications est invahissant. Pour contrôler une projection, deux articles doivent être fournis. Ceux-ci sont :

- Une version de programme visualisé appropriement modifiée. Elle est obtenue en compilant et en liant le texte source annoté.
- Fichier de manuscrit de configuration, qui contient des spécifications déclaratives de la visualisation, qui détermine en détail sa portée et forme.

Quand un programme est modifié et compilé, il peut être exécuté en utilisant manuscrits de configuration divers, sans besoin de recompilation. Des projections diverses du même algorithme peuvent être obtenues de cette façon. Elles peuvent différer avec leur portée, forme et niveau d'abstraction. Elle limite le caractère invahissant des spécification – au moins du point de vue de l'utilisateur.

Daphnis se compose de plusieurs pièces et modules. Les pièces suivantes sont essentielles :

- Unité de machine projective, mise en application sous la forme de deux bibliothèques de lien dynamiques (ang. *dynamic-link libraries, dll*). On les fournit pour les liér aux programmes étant visualisés.
- Composants *ActiveX* (*COM*) – bibliothèques qui exportent les interfaces uniformes des objets participant à la projection. Des nouveaux composants peuvent être facilement ajoutés par l'utilisateur – et ceci fait l'architecture de *Daphnis* ouvert.

Les pièces et unités additionnelles suivantes ne sont pas nécessaire pour faire une projection (mais ils facilitent la tâche de l'utilisateur) :

- Un ensemble des animations exemplaires. Actuellement c'est le point de départ de créer une grande bibliothèque des projections qui peuvent être utilisables pour l'enseignement d'informatique.
- Un programme appelé *DAPHNIS.EXE*, qui facilite l'activation des projections choisies.
- Outils qui facilitent le travail de la visualisateur : préprocesseur de texte source et magiciens (ang. *wizards*) interactifs pour créer les manuscrits de configuration. Actuellement, ces outils sont en construction.

CHAPITRE 6 : ENVIRONNEMENT INTERACTIF DU SYSTEME *DAPHNIS*

L'environnement interactif du système *Daphnis* a été conçu de sorte que pour répondre à des exigences de divers groupes d'utilisateurs. Il ne pourrait pas être fait, à moins qu'une analyse des groupes d'utilisateurs potentiels soit faite.

En général, les utilisateurs du système peuvent être divisés en quatre catégories :

- **Étudiants – observateurs**. Cette catégorie contient les utilisateurs qui utilisent le système comme aide éducative, et qui observent principalement des projections au préalable préparées par quelqu'un d'autre. L'interface simple et convivial pour l'observation des projections et le répertoire riche du moyen graphique sont cruciaux pour cette catégorie d'utilisateurs.
- **Étudiants – explorateurs**. Ce groupe contient les étudiants qui, après observation de la projection prête à fonctionner, essai pour explorer le système. Leurs actions typiques sont : l'interaction avec l'image observée (examination ou modification des structures des données), les modifications des projections, et, en fin, la création des projections de noveau. Nous pouvons compter que les utilisateurs connaissent les dispositifs elementaires du système, ou qu'ils les apprendront. En général, les conditions des étudiants – explorateurs se trouvent entre les conditions des étudiants – observateurs et des enseignants.
- **Enseignants** sont des utilisateurs, qui emploient le système pour préparer de nouvelles projections qui seront employées comme aide éducative pour les étudiants. L'interface simple et convivial n'est peut-être pas aussi critique pour cette catégorie d'utilisateurs comme elle était dans le cas des étudiants, mais le système devrait encore fournir les outils les plus faciles que possible.
- **Concepteurs.** Cette catégorie groupe les utilisateurs qui appliquent le système comme outil de correction. Ils doivent créer des projections vraiment rapidement et facilement. La projection devrait leur permettre d'explorer le logiciel visualisé, mais les conditions au sujet du moyen et du niveau graphiques de l'abstraction sont limitées par rapport au groupe d'étudiants.

L'état critique est la facilité de préparation de projection ; les solutions entièrement automatiques, même elles seraient preferées si elles n'etaient pas trés sophistiqueées.

L'interface simple et facile à utiliser est un des conditions les plus critiques. Les étudiants – observateurs doivent avoir une interface facile à utiliser pour courir des projections. Les conditions critiques sont : manière facile d'activer une nouvelle projection et outils faciles à utiliser qui permettent de commander la course de la projection : la commencer, faire une pause, l'arrêter, commander son pas, mesurer l'image, examiner les valeurs des données montrées.

Les concepteurs ont besoin de l'environnement extrêmement facile à utiliser pour préparer les nouvelles projections qui leur permettraient de créer les projections simples sans beaucoup d'effort. Des solutions automatiques ou semi-automatiques seraient préférées. Plus la projection est compliquée, plus sa préparation peut être difficile : les visualisations créées par les enseignants pour buts éducatifs peuvent être relativement difficiles, et peuvent exiger beaucoup d'effort de l'utilisateur et un certain niveau de la connaissance du système. D'autre part, la réaction de l'utilisateur ne devrait pas être terreur, irritation ni sondage. Les conditions critiques sont : dégager, langue bien conçue des spécifications de la visualisation, outils qui facilitent la tâche de visualisateur (des préprocesseurs de textes et des outils interactifs), et, enfin, intégration des outils et interface utilisateur confortable et conviviale.

Quant au répertoire graphique du système, il doit répondre aux exigences didactiques, car ils sont les plus grands. D'autres utilisateurs se satisferont avec un sous-ensemble de ce répertoire. Les états critiques de la projection éducative réussie sont : compréhensibilité, clarté, contenu informationnel, niveau d'abstraction et forme attrayante. Le répertoire des moyens graphiques est essentiel.

Les parties suivantes de l'interface utilisateur du système *Daphnis* sont discutées :

- **Interface d'observateur**. Un programme spécial (fig. 6.1, voir également les planches 6.1 – 6.4 dans la redaction) facilite d'activer des projections. Les utilisateurs peuvent choisir des prélever projections, ou trouver un fichier exécutable par eux-mêmes. En général, il suffit faire marcher un programme préalablement préparé pour la projection, ce qui cause l'ouverture de la fenêtre de projection, qui montre l'image et donne les outils nécessaires pour commander la projection (fig. 6.2, voir également la planche 6.5 dans la redaction). Ces outils contiennent des commandes pour la gestion de la projection (le début, font une pause, reprennent et arrêt), accordant le pas, la graduation d'image (fig. 6.3, tableau 1). Dans la conception des toolbars une métaphore du monde vraie d'équipement audio a été appliquée. Les outils pour interaction avec le programme visualisé sont également disponibles. Une fonction de manipulation directe des éléments graphiques est actuellement en construction (elle était disponible dans le système *WinSANAL*).

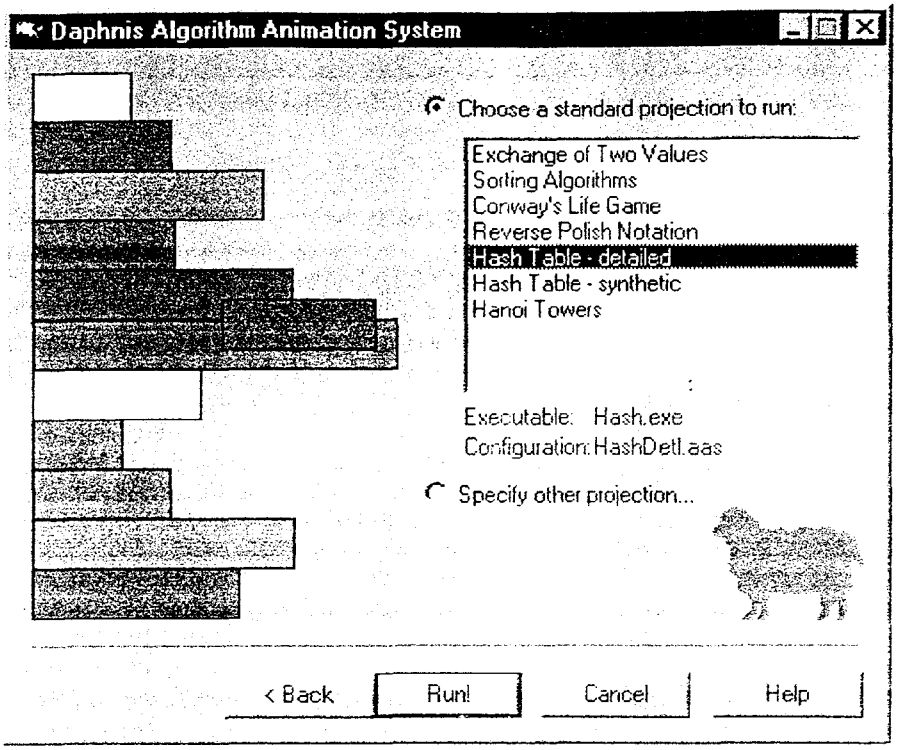


Fig. 6.1 *DAPHNIS.EXE*, programme qui facilite l'activation des projections

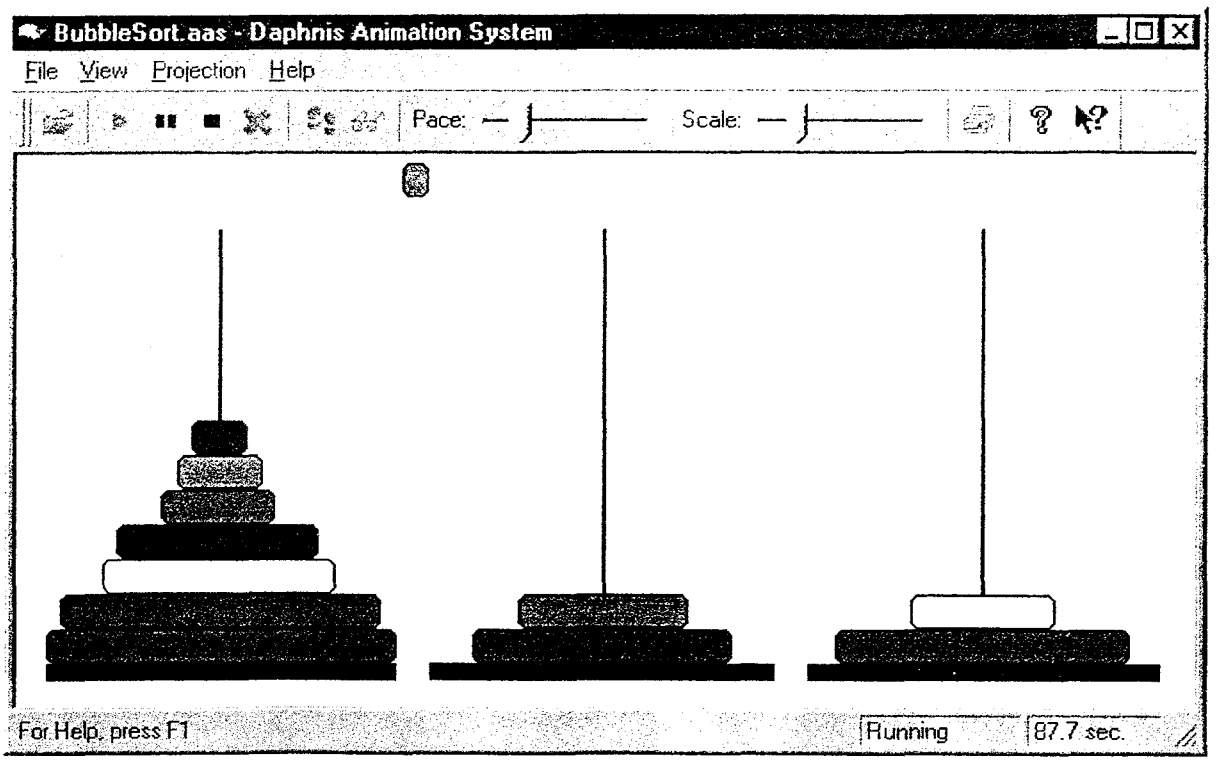


Fig. 6.2 Fenêtre de projection du système *Daphnis*

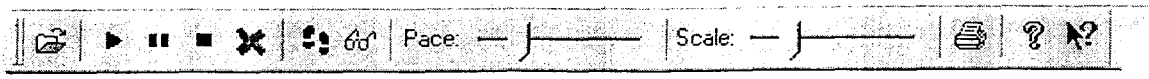


Fig. 6.3 Toolbar du système *Daphnis*

Menu	Option	Description	Touche	Bouton
File	Open script...	Permet de choisir le manuscrit de configuration	Ctrl+O	
	Edit script...	Édite le manuscrit de configuration	Ctrl+E	
	Print...	Imprime le cadre d'animation courante	Ctrl+P	
	Print preview	Montre prévision des pages imprimées		
	Print setup	Permet d'installer l'impression		
	Exit	Termine le programme	Alt + F4	
View	Toolbar	Commute la visibilité de toolbar		
	Status bar	Commute la visibilité de la barre de statut		
Projection	Start	Commence la projection	F5	
	Pause/Resume	Fait une pause pendant la projection	F6	
	Stop	Termine la projection	F7	
	Single step	Anime les opérations en mode d'étape	F2	
	Inspect	Montre la fenêtre d'inspection des données	F4	
	Clear	Enlève tous les graphiques articles		
Help	Help	Montre la fenêtre d'aide	F1	
	Index	Montre la fenêtre de contenu d'aide		
	About Daphnis	Montre les informations sur le programme		
		Détermine le pas de projection	Ctrl ←→	
		Mesure l'image	Ctrl ↑↓	

Tableau 6.1 Sommaire des commandes disponibles pour l'utilisateur dans la fenêtre de projection

- **Le répertoire graphique** répond aux les exigences didactiques. Les conditions critiques sont :
  - **Compréhensibilité et clarté**. Les graphique du système peuvent être brièvement décrits comme « l'animation coulante conduite par le flux des données ». Ceci fait des projections faciles à comprendre. Autre contraint à concevoir pendant la création du répertoire graphique étaient : variété d'objets graphiques, flexibilité et architecture ouverte.

- **Contenu informationnel et niveau d'abstraction**. Notre méthode originale d'animation des algorithmes est cruciale dans ce secteur. Le système fournit également les moyens graphiques qui peuvent augmenter le niveau de l'abstraction, mais ils impliquent la conception main-ouvrée.
- **Forme attrayante**. Elle augmente non seulement l'enclenchement d'utilisateur, mais aussi facilite l'observation des détails de l'image. Les effets qui augmentent l'attraction des projections sont : variété de formes graphiques, l'animation coulante, utilisation étendue de couleur et de bruit, et qualité esthétique de l'image.

- **Interface de visualisateur**. Création d'une projection dans *Daphnis* implique trois étapes :
  - **Annotation du texte source de programme**. Il fonctionne selon des règles fixes, et pourrait être facilement automatisé (les travaux sur un préprocesseur automatique de texte source sont en marche). Les fonctions qui sont insérées dans le texte du programme visualisé peuvent être divisées en deux groupes. La premiere d'entre elles est un groupe de fonctions de conduite, que maintiennent la projection dans le mouvement. Ils sont essentiels pour la méthode d'animation appliquée, et ont été discutés en chapitre 4. L'autre groupe est le groupe de fonctions techniques (auxiliaires). Certains d'entre elles reproduissent la fonctionnalité de l'interface utilisateur.
  - **Compilation et liaison**. Les fonctions utilisées à annoter des programmes source sont disponibles sous la forme d'une bibliothèque de lien dynamique. Compilation et liaison des programmes dans C/C++ est facile, et la préparation des programmes dans d'autres langues est seulement un peu plus compliqué. Dans les deux cas le résultat est un fichier exécutable modifié et prêt à fonctionner. Il peut être executé avec manuscrits divers de configuration, et le manuscrit changeant de configuration n'exige pas de recompilation.
  - **Fournition du manuscrit de configuration**. C'est l'étape la plus difficile dans la préparation de projection. Pour créer un manuscrit, les utilisateurs non seulement doivent connaître la langue des manuscrits, mais connaître également les règles générales de la structure de la machine projective. En général, le manuscrit de configuration contient une série de définitions des objets, appelées les acteurs. Chaque acteur est relié à une variable individuelle soumise à la visualisation et avec l'élément graphique (appelé le ***grel***), cela représente la variable. Il contient également un ensemble de règles de traduction, qui déterminent la manière, comment les valeurs ou les adresses des variables doivent être traduites aux valeurs commandant de divers attributs de l'image finale. La description de la syntaxe et de

la sémantique des manuscrits de configuration est illustrée dans la redaction par plusieurs exemples – simple et avançé.

En conclusion du chapitre 6, la rentabilité du système est discutée. Cette discussion est plutôt théorique, car aucune recherche empirique n'a été entreprise. La rentabilité est analysée dans quatre catégories :

- **L'utilité**. Le système visualise divers algorithmes avec succès, toutefois l'utilité de *Daphnis* peut être ameliorée en augmentant son répertoire graphique et en accomplissant les outils facilitant les efforts de visualisateur.
- **L'efficacité**. Elle était une des plus importante contraint pris en considération pendant la conception d'interface utilisateur. Le déclenchement d'une projection et le contrôle de celle-ci est facile et efficace. La possibilité de faire le processus de la préparation de projection plus efficace est limitée, mais les projections obtenues avec l'effort relativement bas sont bien meilleur en comparaison avec la majorité d'autres systèmes (avec le niveau semblable de l'effort de l'utilisateur).
- **L'assimilation**. L'interface de l'utilisateur qui observe une projection est simple, conviviale et intuitive. Normalement elle exige peu ou pas de formation précédente. L'interface de visualisateur est un peu plus compliqué. Le facteur le plus important est une syntaxe de manuscrit de configuration, que – comme nous considerons – est claire.
- **L'impression**. Les solutions graphiques qui sont appliquées dans *Daphnis*, comme l'animation coulante, fait habituellement une impression positif aux utilisateurs. Les utilisateurs, qui ont jusqu'ici observés et travaillés avec *Daphnis*, ont été satisfaits et impressionnés.

CHAPITRE 7 : CONSTITUTION DU SYSTEME *DAPHNIS*

Le chapitre 7 est consacré à l'architecture interne du système *Daphnis*. La partie la plus importante de cette architecture est la machine projective. En bref elle peut être caractérisée appliquant le modèle *POST*. La machine projective de *Daphnis* constitue la réalisation alternative du modèle de *POST* (par rapport à l'implementation originale dans Smalltalk).

Plusieurs unités peuvent être distinguées dans la machine projective. Ceux-ci sont :

- La machine projective vraie – la pièce de grain. Elle est responsable du processus entier de la marche de la projection et de l'affichage d'image. Elle acquiert l'information du programme visualisé, la traduit et la montre. Toutes autres unités sont les interfaces à la machine projective vraie.

- L'interface du programmeur (*API*) est une bibliothèque des fonctions disponibles aux programmes soumis à la visualisation, aussi bien qu'à l'unité de l'interface d'utilisateur.
- Le traducteur de manuscrit est une interface entre le manuscrit de configuration et la machine projective vraie. Il lit le manuscrit, fait son analyse lexicologique, syntactique et sémantique, et contrôle la création des objets de prototype à l'intérieur de la machine projective vraie. Ces objets sont utilisés pendant une projection comme des modèles pour créer de nouveaux acteurs – objets qui participent à la projection.
- L'interface utilisateur est une unité responsable pour l'affichage de la fenêtre de projection et pour l'interaction avec l'utilisateur. Cette unité est également responsable de la graduation et du défilement d'image. Cependant, la tâche de dessiner l'image est hors de la responsabilité de cette unité. Tout ce qu'elle fait c'est qu'elle passe le contexte de dispositif de la fenêtre de projection vers la machine projective vraie, qui fait toute la tâche.
- Les composants *ActiveX* sont une pièce intégrale de la machine projective, mais leur localisation – dans des fichier séparés *dll* – et le mode de liaison avec le reste du système par l'interface *COM/ActiveX*, qui permet d'ajouter facilement de nouveaux composants à *Daphnis*. Cela fait l'architecture du système ouverte. Ajouter de nouveaux composants n'exige pas la recompilation de *Daphnis*.

La section finale du chapitre 7 est consacrée aux problèmes liés à l'architecture ouverte du système *Daphnis*. Des interfaces *COM/ActiveX* et le mode de l'enregistrement du système sont définis.

CHAPITRE 8 : ANIMATIONS EXEMPLAIRES

Les suivants exemples d'animations sont présentées dans le chapitre 8 :

- Opération de l'échange de deux valeurs.
- Algorithmes de tri.
- Simulation d'un automate cellulaire.
- Traduction des expressions à la notation polonaise inversée.
- Accès de rangée avec l'utilisation de la fonction d'informations parasites (ang. *hash function*).
- Tours de Hanoi.

La galerie des instantanés des visualisations est montrée dans fig. 8.1 (voir également les planches 8,1 – 8,6 dans la redaction). En-dessous une d'entre animations exemplaires est récapitulée.

ALGORITHMES DE TRI

Un manuscrit de configuration court est facile de créer, et il est assez pour faire une visualisation simple de n'importe quel algorithme de manipulation de rangée, y compris toute les algorithmes de tri. Une projection créé avec un tel manuscrit, et le texte annoté et compilé

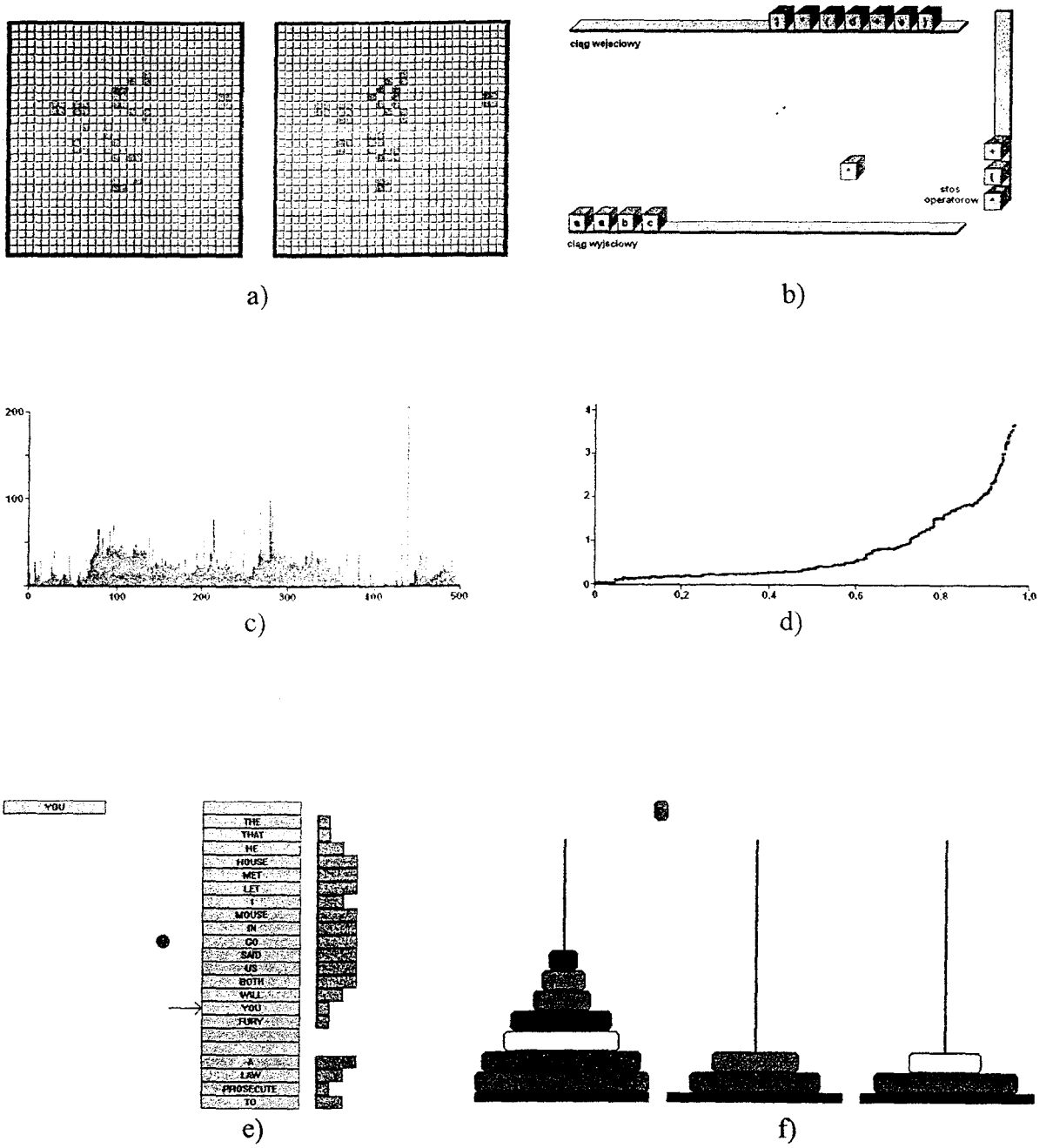


Fig. 8.1 La galerie des animations des algorithmes preparées en utilisant le systéme *Daphnis* : a) Simulation d'un automate cellulaire. b) Traduction des expressions à la notation polonaise inversée. c), d), e) Trois vues d'accès de rangée avec l'utilisation de la fonction d'informations parasites. f) Tours de Hanoi.

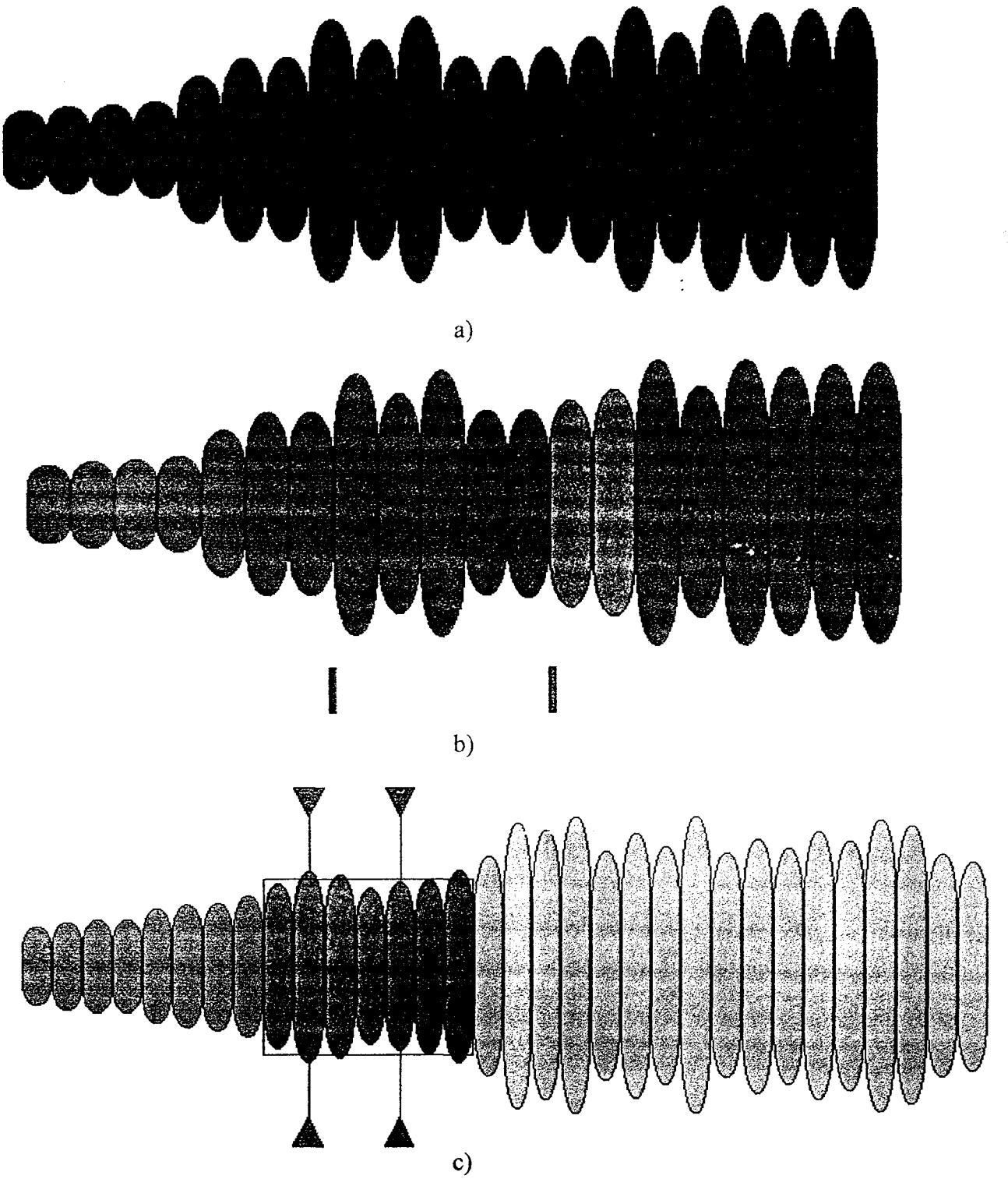


Fig. 8.2 Animations des algorithmes de tri : l'algorithme de tri par échange de paire de clés : a) l'animation simple, b) l'animation avançée, c) l'animation d'algorithme de tri rapide.

de l'algorithme de tri par échange de paire de clés (ang. *bubble-sort*) est présentée (fig. 8.2a). Cette visualisation est générique, et ne contient pas d'information particulière au sujet de ce type d'algorithme. Une version plus avançée de manuscrit a été appliquée pour créer une vue

plus abstraite (fig. 8.2b). Elle contient la représentation graphique des variables de commande de boucle et elle emploie de diverses couleurs pour distinguer les articles triés et non triés de la rangée, et aussi pour accentuer les articles étant comparés. Il est remarquable que le texte source a été réutilisé sans aucun changement, et également sans besoin de recompilation.

Un autre exemple des algorithmes de tri est également présentée : une algorithme de tri rapide (ang. *quicksort*, fig. 8.2c). Elle fournit la vue abstraite, dans laquelle le codage de couleurs a ètè utilise pour marquer les sous-ensembles des elements étant actuellement triés. La représentation graphique des variables de commande de boucle et de la valeur de la médiane sont aussi montré.

CHAPITRE 9 : REMARQUES FINALES

En chapitre 9 nous concluons que la thèse principale a été prouvée. Une méthode originale d'animation des algorithmes qui utilise surveillance de flux des données a été développée. Elle introduit automatiquement des éléments à la projection qui augmentent de manière significative le niveau de l'abstraction de la visualisation. La méthode proposée satisfait trois postulats d'animation réussie des algorithmes :

- L'illustration des flux des données,
- Suppression spatiale d'information,
- Suppression temporale d'information.

La méthode a été pratiquement vérifiée dans le système d'animation des algorithmes *Daphnis*. Les expériences avec le système ont confirmé la rentabilité de la méthode proposée. La création des projections dans *Daphnis* est facile et automatiquement satisfait les postulats donnés ci-dessus. Les projections facilement créées dans *Daphnis* impliqueraient beaucoup d'additionnel conception manuel dans les systèmes de visualisation de logiciel existant jusqu'ici, au moins par ceux connus de nous.

Les directions principales des travaux futurs sont :

- Développement ultérieur du système *Daphnis*, particulièrement la prolongation des moyens graphiques pour visualisation des structures dynamiques, l'implementation de vue du texte source et amélioration des techniques de manipulation directe.
- Développement de la visualisation de trois-dimension et de l'auralisation (visualisation en utilisant le bruit comme médias de communication).
- Exécution des projections distribuées dans l'environnement du réseau.
- Développement des outils d'aide du visualisateur.

- Expansion de l'ensemble d'animations exemplaires.

Parmi les travaux futurs qui ont besoin encore d'autres divagations théoriques sont :

- Application d'analyse statique du texte source pour enrichir l'information dynamiquement rassemblée sur des données flux.
- Trouvant un modèle réussi de la visualisation pour les aspects non impératifs de logiciel, particulièrement pour les langues déclaratives et orientés objet.
- Visualisation de grands systèmes.

BIBLIOGRAPHIE

La redaction contient une bibliographie étendue, divisée en catégories thématiques. Au-dessous quelques positions les plus utiles concernant la visualisation de logiciel sont presentées.

1. M. H. BROWN. *Exploring Algorithms Using Balsa II.* IEEE Computer, Vol. 21, N°5, 1988, str. 14-36.
2. M.H. BROWN. *Algorithm Animation.* The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1988. ISBN 0-262-02278-8.
3. J. FRANCIK, P. SZMAL. *Wizualizacja synchroniczna algorytmów – realizacja w systemie SANAL.* Zeszyty Naukowe Politechniki Śląskiej, seria Informatyka 27, 1994, str. 37-54.
4. A. HYRSKYKARI. *Development of program visualization systems.* Technical Report A-1995-3, University of Tampere, Department of Computer Science, P.O. Box 607, FIN-33101 Tampere, Finland, 1995.
5. B. A. MYERS. *Taxonomies of Visual Programming and Program Visualisation.* Journal of Visual Languages and Computing, Vol. 1, 1990, str. 97-123.
6. M. PETRE, B.A. PRICE. *Why Computer Interfaces Are Not Like Paintings : the User as a Deliberate Reader.* Proc. of East-West International Conference on Human-Computer Interaction, Moskwa, 1992.
7. A. PRICE, R. M. BAECKER, I. S. SMALL. *A Principled Taxonomy of software Visualisation.* Journal of Visual Languages and Computing, Vol. 4, 1993, str. 211-266.
8. S. P. REISS. *Program Visualization : Where We Go From Here.* IFIP Transactions A Vol : A-12, 1992, str. 218-27.
9. G. C. ROMAN, K. C. COX. *A Taxonomy of Program Visualization Systems.* IEEE Computer, no 12, 1993, str. 11-24.
10. G. C. ROMAN, K. C. COX. *Program visualization : the art of mapping programs to pictures.* International Conference on Software Engineering, New York, 1992, str. 412-420.
11. G. C. ROMAN, K. C. COX. *Abstraction in algorithm animation.* Proceedings of 1992 IEEE Workshop on Visual Languages, Los Alamitos, CA, USA, 1992, str. 18-24.
12. J.T. STASKO, J.B. DOMINGUE, M.H. BROWN, B.A. PRICE, editors. *Software Visualisation. Programming as a Multimedia Experience.* Foreword by Jim Foley. The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1998. ISBN 0-262-19395-7.

13. J. T. STASKO, C. PATTERSON. *Understanding and characterising software visualization systems.* Proceedings of IEEE Workshop on Visual Languages. Seattle, WA, USA, 1992, str. 3-10.
14. P. SZMAL, J. FRANCIK. *Algorithm animation and debugging with the WinSanal system.* Proc. of IASTED Conference Applied Informatics, Innsbruck, Austria, 1997.
15. F. VAN DE VEIRE. *La Simulation et l'Animation Modulaire d'Algorithmes en Langage Objet.* Rozprawa doktorska, Université des Sciences et Technologies, Lille, Francja, 1997.
16. *Arne's Bibliography on Software Visualization.* http://i44s11.info.uni-karlsruhe.de/~frick/SoftVis/bibliography.html

REMERCIEMENTS
(TRADUCTION LITTERALE)

Je m'étais préparé au devoir plaisant d'exprimer ma grâce à tout ceux, dont les efforts ont contribué le fait, que mes cinq années de travail à l'Ecole Polytechnique Silesienne et à l'Université de Lille peuvent être aujourd'hui couronné avec cette redaction.

D'abord je voudrais remercier les directeurs de ce travail, M. le Professeur Stefan Węgrzyn et M. le Professeur Jean-Marc Toulotte – non seulement pour leur aide substantielle, mais aussi pour la main-forte donnée dans les affaires d'organisation. Je remercie également les directeurs des établissements, dans lesquels j'ai travaillé : Mes. le Professeur Andrzej Grzywak et le Professeur Christian Vasseur, et également M. le Professeur Stanisław Kozielski. Sans leur enclenchement dans les affaires d'organisation je n'aurais pas pu réaliser mes plans.

Le mot commun "merci" n'est absolument pas suffisant par rapport à Monsieur le Docteur Przemysław Szmal, mon chef, collègue et ami, qui a indéfiniment lu toutes les versions prématurées (et non seulement) du texte et a contribué dans beaucoup d'ampleur à son amélioration. Tout le temps il était également indefatigable présent pour m'encourager à réussir ce travaille, auquel – en tant que mon chef – il me créé de parfaites conditions à sa réalisation. Le texte du manuel a également été lu beaucoup de fois par M. le Docteur Rachid Ikni de l'Université Littoral de Calais. Ses remarques essentielles, aussi bien que techniques et même éditorials, ont causées que le texte est aujourd'hui bien meilleur (et il ressemble mieux à une dissertation scientifique).

Je remerci également l'équipe de mes collaborateurs plus proches, telles que : Mes. Krzyrztof Dobosz, Mateusz Nowak, Paweł Gonera, Artur Migas, Krzyrztof Wójcik, Mme Anna Tomaszewska et les auteurs des versions premieres de *SANAL*, et également M. Frédéric Van de Veire. Ils ont tous contribué à ce travail. Je remercie également mes collègues de l'Institut d'Informatique de l'Ecole Polytechnique Silesienne et du Laboratoire

d'Automatique I^{3}D de Lille, particulièrement Mes. Olivier Losson, Jean-Marc VanNobel et Eryk Czesnalowicz pour leur aide et amitié, qu'ils m'ont offert pendant mon séjour en France, et M. le Professeur Tadeusz Czachórski, qui a non seulement traduit pour moi des textes en français, mais également a échangé avec moi sa connaissance au sujet de la France et des expériences de la vie dans ce beau pays. Je ne puis pas omettre Mme K. Wrześniowska, qui m'a présenté dans le monde de la langue française, ni ma belle-mère. Je la remercie car elle m'a beaucoup renseignée sur le pays de son enfance.

Il me serait impossible d'énumérer tout le monde, a qui je dois des remerciements pour les discussions créatrices, qui ont contribué à mon travail. Je mentionnerai quand même quelques personnes : Mes. le Professeur Adam Mrózek, le Docteur Marek Konopka, le Docteur Lech Znamirowski, Wojciech Mikanik et Michał Kolano. Je remercie également tout ceux, dont leur travail était pour ma persone essentiel et qui ont pris soin du bon état de l'équipement technique et ont arrangé des millions d'affaires de bureau. Mes remerciements exceptionnels à Krzyrztof Podstawa, qui était non seulement surveillant de valeur inestimable de réseau, mais qui est également mon très bon ami.

Mes remerciements finals, mais pour moi très importants, je voudrais les passer vers les membres de ma famille, qui ont dû supporter mes modes. Sont les soucis en tant de mes parents (phase premiere), en tant à ma femme (phase postérieure). Ma femme, Katherine, a indirectement contribué à mon travail : elle a créé quelques illustrations, dont son contours ont été modifiés tant de fois...

Merci une fois de plus à ma belle-mère pour son aide dans la traduction de ce resumé.