Les Composants Logiciels : Evolution technologique ou nouveau paradigme ?

Fr d ric Peschanski, Thomas Meurisse, Jean-Pierre Briot

Laboratoire d'Informatique de Paris 6
Th me OASIS

{Frederic.Peschanski, Thomas.Meurisse, Jean-Pierre.Briot}@lip6.fr

R sum :

Nous proposons dans cet article une analyse de la notion de composant logiciel, tr s en vogue l'heure actuelle. Deux d finitions pr liminaires sont pr sent es, l'une provenant du th me de recherche des architectures logicielles, l'autre plut t issue du monde industriel : les composants binaires. Nous d battons des sp cificit s des approches bas es sur les composants (par rapport aux approches objet en particulier), que ce soit dans les technologies industrielles ou dans nos propres travaux de recherche dans le domaine. Une proposition de notion identitaire de composant logiciel, associ e un cahier des charges minimal, est labor e. Nous expliquons pourquoi les approches industrielles ne satisfont pas compl tement ce cahier des charges. Nos architectures Comet et Maleva, ainsi que leurs instanciations, pr sent es ensuite, fournissent selon nous des l ments de r ponses.

Introduction

La notion de composant logiciel occupe aujourd'hui une partie importante du paysage informatique, notamment dans les domaines du g nie logiciel, des syst mes distribu s ou encore des r seaux. Comme tout concept mergent, l'ambigu t pr vaut lorsqu'il s'agit d'en donner une d finition claire. Nous pouvons cependant nous reposer sur une d finition relativement consensuelle :

Les composants logiciels sont des unit s de composition de logiciels

Cette proposition est bien s r peu satisfaisante mais elle est un point de convergence de nombreux travaux autour du concept de composant. D s les ann es 80, le terme de composant sera souvent employ en informatique (par exemple dans [1] et [2]). Nous pouvons cependant d gager deux approches fondatrices : les architectures logicielles et les composants binaires.

Issu de la recherche, le domaine des architectures logicielles propose de d crire la structure d'un logiciel comme un assemblage, ou plut t une interconnexion, de composants. Dans cette optique, le crit re de neutralit associ la notion de composant est tr s fort : il n'est fait aucune supposition sur le r le d'un composant. Dans [3], D. Garlan and M. Shaw proposent la d finition suivante :

Une architecture d'un syst me donn est une collection de composants calculables - ou plus simplement composants - associ e une description des interactions entre ces composants - les connecteurs.

Du point de vue industriel, les besoins en terme de composants logiciels s'expriment plut t par la n cessit de disposer de technologies permettant de composer des applications partir d' l ments logiciels r utilisables [4]. L'hypoth se forte qui pr vaut ici est la notion de diffusion sous forme binaire de ces l ments appel s composants. Nous pouvons extraire de ces concepts la d finition suivante :

Un composant est une entit logicielle (d'impl mentation), labor e et vendue (sous forme binaire) par une soci t et utilis e par un (ou des) client(s) pour composer un logiciel.

Les nombreuses solutions industrielles base de composants qui mergent aujourd hui (COM, JavaBeans, EJB, Corba Components) s inspirent principalement de ces deux courants. Nous commen ons en section 2 par d crire de fa on synth tique ces diff rentes approches. Apr s ce tour d horizon, nous pr sentons ce qui, pour nous, peut conduire une v ritable notion identitaire de composant logiciel (section 3). Ce point de vue semble cependant entrer en conflit avec certaines fondations sur lesquelles reposent les mod les industriels, ce qui a motiv l laboration de deux mod les de composant, diff rents et compl mentaires, au sein de notre quipe OASIS. Ces approches de recherche, Comet et Maleva, sont pr sent es en section 4. Nous expliquons galement dans cette section dans quelle mesure ces travaux satisfont la notion identitaire de composant logiciel propos e pr c demment. Finalement, la section 5 conclut cet article.

Propositions industrielles

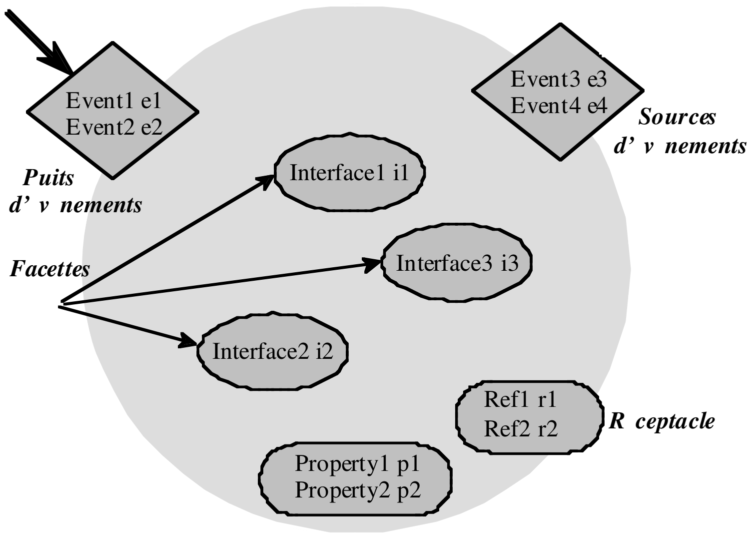
Depuis 1995, Microsoft d veloppe un ensemble de technologies permettant (d'apr s la documentation) la composition d'applications partir de briques logicielles d velopp es s par ment. Dans ces technologies, COM et d riv s [5], issues de OLE (*Object Linking & Embedding*), Microsoft privil gie l approche des composants binaires : l'utilisateur n'a aucun acc s au code des composants qu'il manipule. La vision de Microsoft est assez extr me puisque les informations disponibles sur les composants sont fortement r duites. L'abstraction principale concernant la composition est l'interface. Plus pr cis ment, le couplage entre composants est d fini en terme de satisfaction d'interface. Du point de vue technologique, COM s'appuie principalement sur un m canisme de d couverte d'interfaces l'ex cution. Ceci permet la composition dynamique des applications par des composants con us ind pendamment les uns des autres. En terme de r utilisation, les interfaces peuvent tre compos es par confinement ou agr gation. Selon Microsoft (et contrairement au m me concept dans le monde objet), un composant agr g est accessible directement depuis l ext rieur du composant agr gat[1]. Un composant confin , pour sa part, ne peut exporter d interface au niveau du composant qui le contient. Des probl mes li s ces outils de composition sont identifi s et pr cis ment analys s dans [18].

Les JavaBeans [6], d velopp s par Sun Microsystems, repr sentent une sorte de contrepoint l'approche Microsoft. Du point de vue conceptuel, un JavaBeans est d'apr s les sp cifications *"un composant logiciel r utilisable qui peut tre manipul graphiquement dans un outil de conception"*. Inspir par les architectures logicielles, ce mod le de composant privil gie l'extraction de la relation de couplage via l'utilisation d'un mode de communication de type implicite, bas sur les v nements. Un Beans n'identifie pas directement le destinataire des v nements qu'il met, c'est le concepteur d'application qui explicite ces d pendances. Point important, le mod le de mise en oeuvre propos est clairement s par du mod le conceptuel. Au

[1] Dans la terminologie objet, un agr gat est un objet utilis en tant qu attribut d un autre objet et donc, par d finition, inaccessible depuis l ext rieur (sous peine de violation du principe d encapsulation).

niveau op rationnel, la r flexion [7] (introspection, intercession) joue un r le tr s important. Les Beans sont de plus empaquet s dans un format binaire (jar), ils ne sont donc pas incompatibles avec l'approche des composants binaires.

Les Enterprise JavaBeans [8] et les Corba Components [9] repr sentent les deux approches les plus r centes en terme de composants logiciels. Leur objectif principal, dans les deux cas, concerne l' laboration de la partie fonctionnelle des serveurs dans le cadre d'applications industrielles de type 3-tiers : client/serveur/troisi me tiers (essentiellement support de donn es). La logique m tier est donc con ue ind pendamment de la logique syst me qui est d l gu e au support d'ex cution. Pour d finir la logique m tier des serveurs, les EJB et les Corba Components proposent en premier lieu un mod le de composant. Les EJB peuvent tre compar s une extension des JavaBeans de ce point de vue. Les Corba Components proposent plut t une sorte de fusion entre les approches de Sun et de Microsoft. Deux outils de composition, pr sent s comme compl mentaires, sont fournis°: les interfaces et la communication implicite.



component exemple

```
component {
  attribute Property1 p1;
  attribute Property2 p2;

  provides Interface1 i1;
  provides Interface2 i2;
  provides Interface3 i3;

  consumes Event1 e1;
  consumes Event2 e2;

  uses Ref1 r1;
  uses multiple Ref2 r2;

  emits Event3 e3;
  publishes Event4 e4;
} exemple;
```

Figure 1. Mod le abstrait des composants Corba

Chaque composant Corba propose une ou plusieurs interfaces (les facettes) permettant d offrir des services dans une approche de type objets distribu s. Les composants peuvent galement s abonner et/ou diffuser des v nements, la mani re des JavaBeans. Sur la figure 1, le composant exemple propose trois facettes/interfaces°: i1, i2 et i3. Les puits d v nement e1 et e2 abonnent respectivement le composant aux v nements de type Event1 et Event2. La source d v nement e3 correspond une simple mission, e4 tant utilis e pour la diffusion. L interface IDL d crivant ce composant est galement pr sent e en figure 1. En plus de ces mod les abstraits, de nombreuses technologies utilitaires sont propos es°: gestion du cycle de vie, support des transactions, outils de d ploiement, etc. Le mod le des Corba Components peut tre per u, globalement, comme une sorte d'extension du mod le EJB, notamment en terme d'interop rabilit [2].

Vers une notion identitaire de composant logiciel

Les diff rentes approches industrielles, si elles ne sont pas forc ment en comp tition, proposent chacune leur propre vision de la notion de composant logiciel. Pour Microsoft, un composant est une bo te noire pr te tre instanci e dans un logiciel. Sun propose plut t, via les JavaBeans, un

[2] Les EJB sont restreints l'environnement Java.

mod le de programmation graphique. Les EJB et les Corba Components sont un peu part puisqu'ils se placent dans un cadre applicatif assez sp cifique. Mais l'axe de travail commun se situe presque exclusivement dans la sph re technologique. Il semble difficile d'en extraire une notion pr gnante et identitaire de composant logiciel.

Deux axes de recherche compl mentaires peuvent tre envisag s pour d gager une probl matique ad quate. Tout d'abord, il est possible de reconna tre, en guise de bilan de plus de vingt ann es de recherche dans le domaine, un certain nombre de limites l'approche objet, notamment dans le cadre des syst mes distribu s [20]. A l'autre bout du spectre, de nouveaux besoins, par exemple en terme de dynamicit , mergent aujourd'hui et n cessitent parfois d'importantes remises en question.

Nous pouvons distinguer deux familles de limites potentielles des mod les objets courants : les limites d'ordre structurel et les limites d'ordre comportemental, c'est- -dire relatives au contr le.

1.1. Aspect structurel

Le point de vue structurel identifie l'ensemble des concepts tangibles au niveau du mod le. Concernant les objets, nous avons extrait ce niveau trois types de probl mes : granularit , couplage et hi rarchisation.

1.1.1. Granularit

Dans les applications, les objets issus de la conception - g n ralement de grain important - et les objets utilitaires - de grain plus fin[3] - se c toient. Ce d coupage quelque peu abrupt doit bien s r tre raffin . En effet, les fonctionnalit s communes aux objets, comme l'instanciation, l'utilisation (invocation des m thodes) ou la destruction ne suffisent plus aujourd'hui. Certains objets - et seulement certains - sont sujets la migration, la persistance, le versioning, etc. La possibilit de distinguer sur le plan structurel diff rentes cat gories d entit s semble aujourd'hui n cessaire. Ces besoins, sans tre tiquet s sous l'appellation "probl me de granularit ", sont en partie identifi s dans les JavaBeans, EJB et Corba Components. Les composants y sont par exemple consid r s comme unit s d'empaquetage (JavaBeans) ou de persistance (EJB/Corba Components).

1.1.2. Couplage

Evoqu beaucoup plus souvent, le probl me du couplage implicite doit galement tre pris en compte [19]. Dans l'expression des fonctionnalit s des objets (code des m thodes), il est possible d'instancier et d'utiliser d'autres objets. Il est galement possible de r cup rer (en argument) des r f rences externes. Ces liens d'utilisation n'apparaissent pas au niveau structurel mais r sultent plut t de l'ex cution proprement dite des programmes. Prenons l exemple de deux classes A et B d finies (en Java) de la fa on suivante°:

```
class A {
  void ma() {
    B b = new B()°;
    b.mb()°;
  }
}

class B {
  void mb() {
```

[3] Un exemple frappant est celui des entiers en Smalltalk.

```
    // code de mb
  }
}
```

Ces deux classes sont clairement coupl es puisque la m thode ma() de A ˙°a besoin°¨ de B et de sa m thode `mb()` pour fonctionner. En raffinant, on peut tablir que les m thodes `A.ma()` et `B.mb()` sont fortement coupl es[4]. Si l on ne dispose pas du code de ces classes, il est impossible d inf rer ce couplage pourtant tr s fort (statique). En terme de r utilisation, en particulier au niveau binaire, cette forme de couplage repr sente un v ritable obstacle. Dans notre exemple, la classe A est difficilement r utilisable si l on ignore son lien tacite avec la classe B. Les mod les de communication implicite par diffusion/abonnement (*publish/subscribe*) apportent ici une bauche de solution. D. Garlan et M. Shaw pr sentent dans [3] les avantages et inconvenients de ce style architectural (*Event-based Implicit Invocation*). L id e principale est qu au lieu d invoquer directement la proc dure d une entit , un composant (publisher) propose sp cifiquement un ou plusieurs v nements en mission. D autres composants (subscribers) peuvent s enregistrer aupr s du publisher afin de recevoir ces v nements. Cette id e est reprise dans les EJB et les Corba Components.
Plus simplement, l extraction du couplage statique entre entit s d un syst me complexe est possible en interdisant l explicitation des destinataires des messages ou v nements dans le code m me de ces entit s. Le couplage est alors d crit explicitement par l interm diaire de connexions. La partie haute de la figure 2 pr sente une interaction entre deux objets *un* et *autre*. Le couplage entre ces deux objets est la fois de nature forte et implicite. La partie basse de la figure transcrit cet exemple dans le monde de la programmation orient e connexion [4].

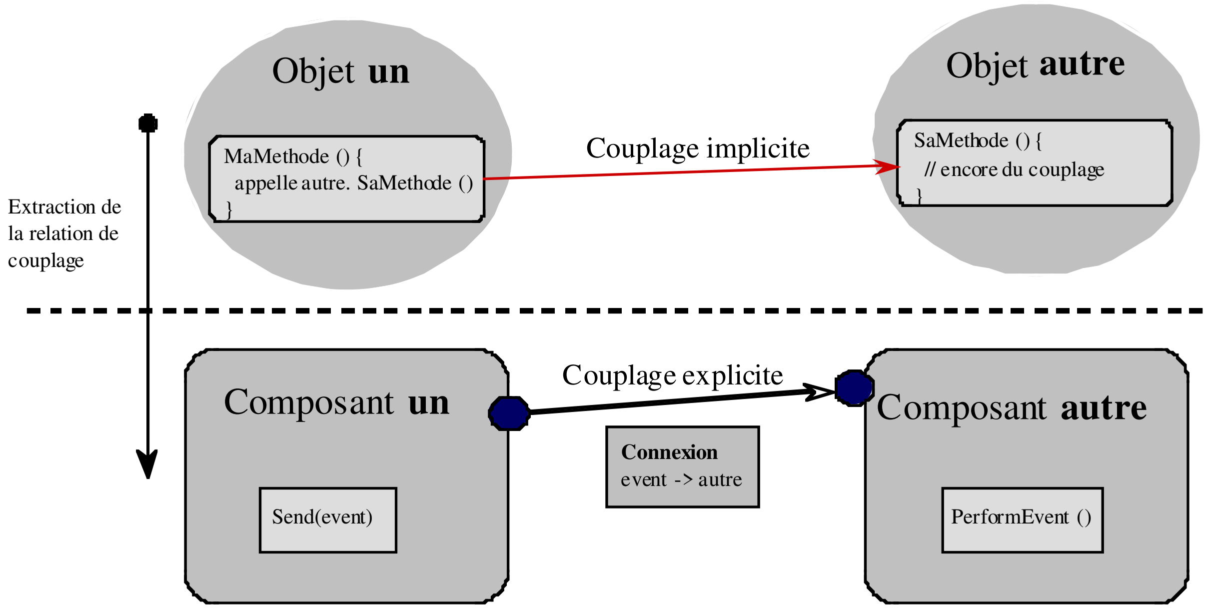


Figure 2.˚Des objets aux composants via l extraction de couplage

1.1.3. Hi rarchisation

La hi rarchisation structurelle repr sente la possibilit d'exprimer, au niveau des mod les (ou des structures, des architectures), le fait qu'un groupe d'entit s forme une autre entit de niveau sup rieur. Le probl me de couplage implicite emp che de "contenir" (notion de confinement) la structure d'un objet, emp chant par la m me occasion la hi rarchisation structurelle. Pourtant, cette "voie hi rarchique" semble proposer une solution tr s int ressante au probl me de granularit . En effet, la hi rarchisation offre une grande flexibilit pour le choix du niveau de granularit . La section 4.1. illustre l importance de ce crit re par un exemple de composition

[4] Toute invocation de `A.ma()` entra ne in vitablement une invocation de `B.mb()`.

hi rarchique dans la plateforme Maleva. Aucune approche industrielle ne semble se diriger vers une solution de ce type.

1.2. Aspect comportemental

L'aspect comportemental est galement extr mement sensible. Sous l'appellation comportement, nous r unissons tout ce qui touche de pr s ou de loin au contr le sur les entit s logicielles mises en jeux. Ceci concerne notamment l'expression proprement dite du contr le, mais galement tous les aspects potentiellement intrusifs ce niveau[5].

1.2.1. Contr le implicite

Dans les environnements objet courants, une grande partie du contr le sur les objets est int gralement d l gu e au support d'ex cution. Pour de nombreux syst mes informatiques r cents, ce contr le devient complexe : concurrence, r partition, s curit , persistance et autres caract ristiques comportementales doivent tre explicit es. Le cas le plus vident de probl me li au comportement concerne la gestion du flot de contr le dans les objets. En g n ral, ces derniers communiquent par invocation synchrone de m thode : l'objet appelant transmet le flot de contr le lorsqu'il invoque la m thode d'un autre objet. Supposons qu un objet a de classe A dispose d une m thode `ma()` effectuant un appel une m thode `mb()` d un objet b de classe B. Sur la figure 3, nous pouvons constater qu apr s invocation de `a.ma()`, l ex cution de la m thode se poursuit jusqu l invocation de `b.mb()`. Le flot de contr le passe alors sous la responsabilit de l objet b. Une fois l'ex cution de la m thode `b.mb()` termin e, l'objet cible b redonne le contr le l'objet source a qui peut continuer l ex cution de sa m thode `ma()`.

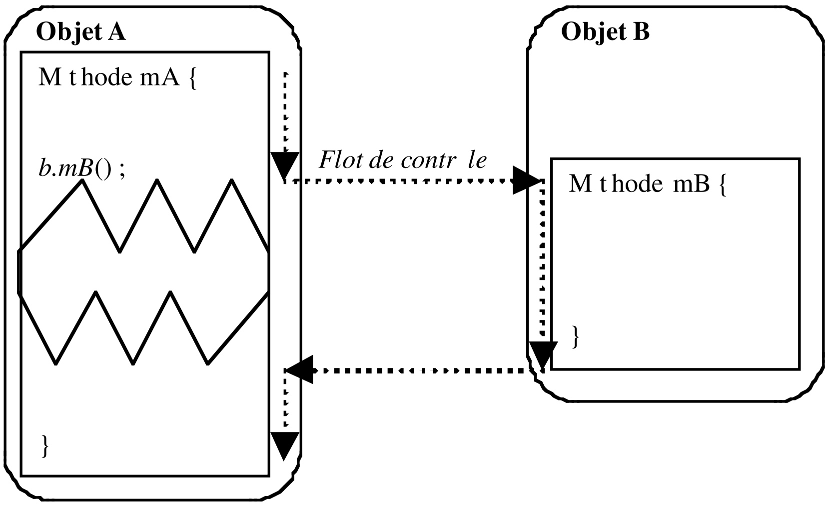


Figure 3. Gestion implicite du flot de contr le dans les mod les objets

Dans les applications parall les, ce mode de fonctionnement n'est pas suffisant puisqu'il est alors n cessaire de disposer d'une ma trise - au moins partielle - sur le flot de contr le. Nous pouvons par exemple imaginer une situation dans laquelle les m thodes `ma()` et `mb()` peuvent s ex cuter en parall le (cf. Figure 4).

[5] Il est, par exemple, difficilement imaginable de d coupler totalement contr le et fonctionnalit s.

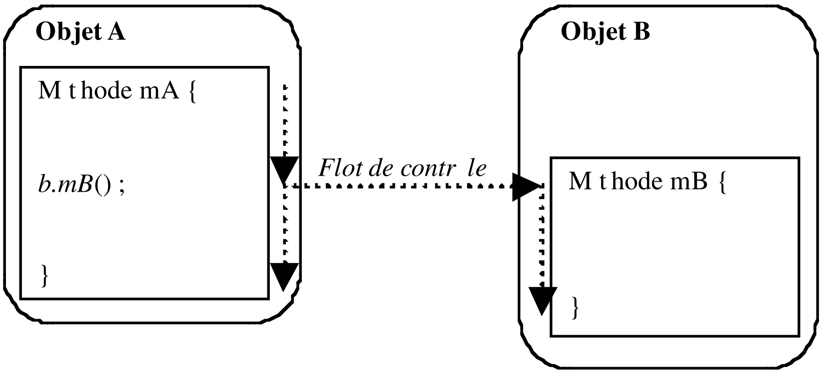


Figure 4. D doublement du flot de contr le lors de l invocation de m thodes.°

Pour expliciter les fonctionnalit s li es au contr le des objets (principalement sur les aspects de concurrence et de r partition), trois familles de solutions sont propos es dans [10] : approche applicative (construction de biblioth ques), approche int gr e (construction linguistique), et approche r flexive (r ification du contr le). Nous verrons que, dans le cadre des composants, une possibilit originale peut tre propos e : explicitation structurelle du contr le (cf. pr sentation de Maleva). Il est important de noter que ces diff rentes solutions ne sont pas forc ment contradictoires.

1.2.2. Coh rence comportementale de la composition

Discuter pr cis ment sur la coh rence s mantique des compositions logicielles semble utopique si l'on exclut cette dimension comportementale. La plupart des approches industrielles table sur une validit purement structurelle (li e aux interfaces), position hautement contestable. Consid rons par exemple la gestion probl matique des activit s multiples dans les composants COM ou JavaBeans. Si l'on ne se repose que sur les m canismes propos s, la cr ation d'une nouvelle activit peut, dans certain cas, totalement d stabiliser une application.

1.3. Crit re de neutralit

Un autre besoin extr mement important concerne le crit re de neutralit associ la notion de composant que nous voulons tablir. Nous devons ici effectuer la jonction avec le th me des architectures logicielles. Pour d crire la structure d'un logiciel quel qu'il soit, il ne doit tre fait aucune conjecture sur le r le d'un composant ou le cadre applicatif des architectures. Les EJB et les Corba Components posent ici probl me puisque la cat gorie de logiciel consid r e est restreinte (quoique importante).

1.4. Proposition

Nous proposons donc d'approcher les composants logiciels par le biais de besoins d'ordres structurel - granularit , couplage, hi rarchisation - et comportemental - expression du contr le et int gration de l'aspect comportemental dans la probl matique de composition - tout en respectant le crit re de neutralit , n cessaire pour satisfaire l'approche des architectures logicielles. De fa on compl mentaire, les composants binaires d coulent, nous semble-t-il, naturellement de cette notion.

Deux approches de recherche

Deux mod les de composants logiciels ont t d velopp s au sein du th me OASIS pour donner corps la notion abstraite d crite pr c demment. Nous ne ferons qu introduire bri vement leurs principes par rapport aux probl matiques mentionn es ci-dessus. Le lecteur peut se r f rer respectivement [11] et [13] pour plus de d tails sur les mod les, architectures et prototypes voqu es.

1.5. Maleva

Maleva est une plate-forme de conception et de simulation de syst mes multi-agents [11]. Le but de cette approche est de fournir un mod le et des outils permettant la mod lisation de chaque agent sous forme d'un r seau interconnect de composants. Les composants mis en ˇuvre dans Maleva sont proches conceptuellement des JavaBeans. La principale originalit de ce mod le concerne la diff renciation flot de contr le / flot de donn es. Ceci est clairement explicit au niveau du graphe d'interconnexion via l'introduction de liens de contr le s'ajoutant aux liens classiques de donn es (cf. figure 5). L'id e principale de Maleva est de permettre la r ification au niveau structurel de certaines propri t s comportementales.

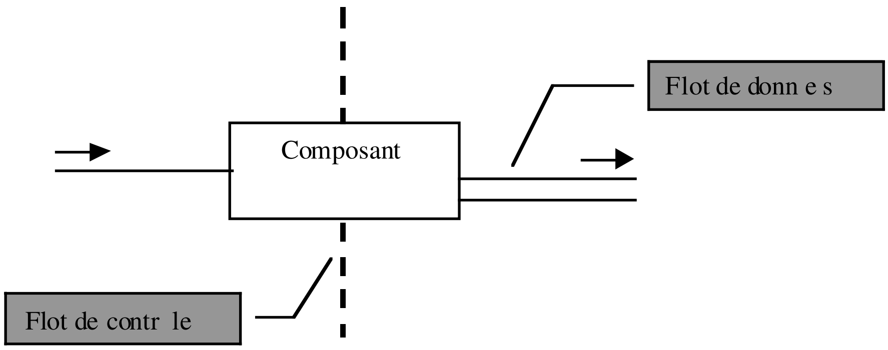


Figure 5. Mod le de composant Maleva

Ce mod le de conception propose galement une approche hi rarchique. En effet, il est possible d'instancier des composants de type composite encapsulant un ensemble de composants de grain inf rieur. Les composants encapsul s sont alors confin s ; ils ne sont donc plus visibles autrement que par l'interm diaire de leur sup rieur (au sens hi rarchique). La figure 6 pr sente un exemple de d composition hi rarchique de l architecture d un agent simulant le comportement une fourmi ouvri re voluant dans un cosyst me. Le niveau sup rieur de la hi rarchie fournit les abstractions les plus lev es°: la capacit pour l agent vieillir (composant *Maturing*) et son comportement global (composant *Behavior*). Le composant *Behavior* est d compos structurellement afin de fournir deux comportements sp cialis s qui sont un comportement de mouvement al atoire (*Random Move*) et un comportement d pendant de la position sp cifique de l agent (*Local Behavior*). Enfin un troisi me niveau de d composition est propos permettant la r action de l agent fourmi dans le cas o il capterait une ph romone mise par une autre fourmi. Cet exemple montre que l approche hi rarchique fournit un cadre int ressant pour la conception descendante. Contrairement aux m thodes de d composition fonctionnelles de type SADT, il est important de garder l esprit que les composants existent galement au niveau de l impl mentation. Bien s r, les approches ascendantes et mixtes sont galement envisageables.

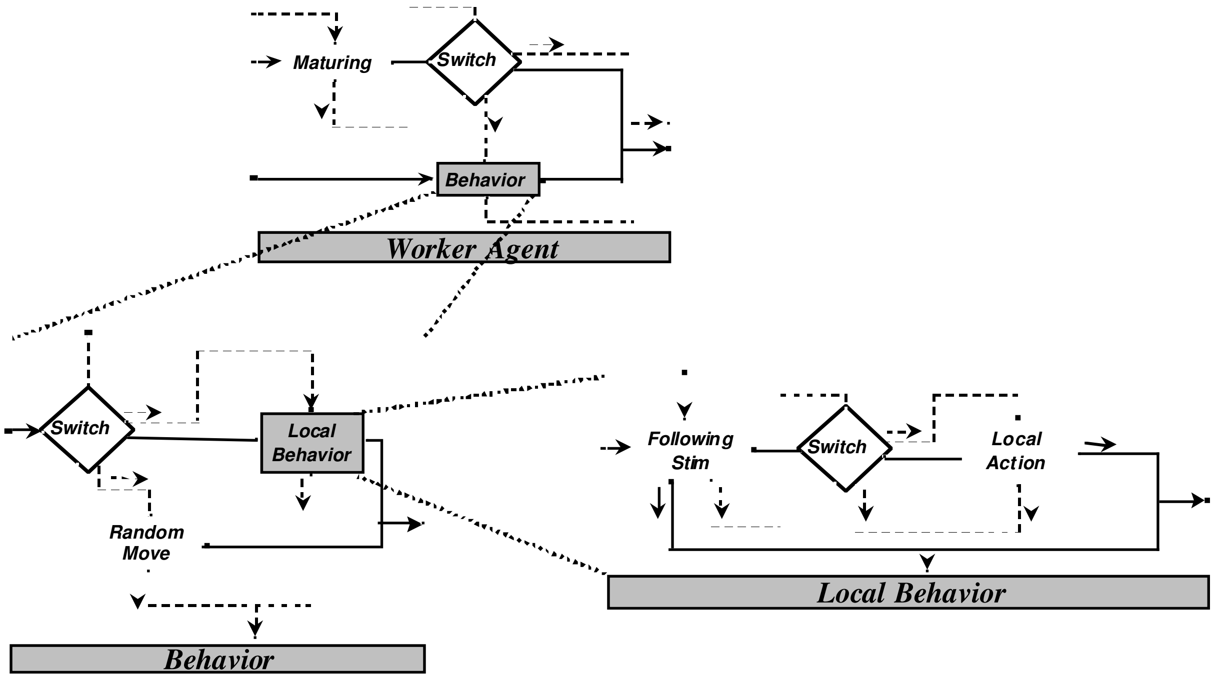


Figure 6. La hi rarchisation dans Maleva°: d composition d un agent ouvri re

Le mod le conceptuel propos par Maleva est compl t par un mod le d'implantation bien distinct. Dans les JavaBeans cette distinction est pr sente, toutefois la coh rence entre les deux mod les est contestable [6]. Dans Maleva, nous avons plut t essay d'assurer continuit et coh rence lors de la concr tisation des concepts propos s. Cette contrainte complexifie le mod le op rationnel dans une large mesure. De nombreux m canismes doivent en effet tre mis en oeuvre : communications asynchrones via l'utilisation de bo tes aux lettres, gestion des activit s multiples, typage dynamique, etc. Cependant, il est difficile l heure actuelle de valider cette coh rence de continuit mais il est important de mettre en vidence le fait que les concepts tangibles au niveau du mod le Maleva se retrouvent au niveau impl mentation.

Deux instanciations de l'architecture ont t d velopp es : Maleva/D con ue en Delphi, et Maleva/J en environnement Java (b ti partir du mod le de JavaBeans). Parmi les exemples impl ment s et test s l'aide de ces deux plates-formes, nous pouvons citer la mod lisation d'un cosyst me (proies/pr dateurs) ainsi qu'une r ing nierie du projet MANTA [12] (simulation d'une colonie de fourmis).

1.6. Comet

L'objectif principal du projet Comet [13] est de fournir une aide pour la conception, l'impl mentation et l'ex cution de syst mes distribu s. Les syst mes que nous visons plus pr cis ment sont particuli rement dynamiques (il se modifie en cours d'ex cution) et h t rog nes en terme de strat gies de contr le.

La proposition que nous faisons est double : mod le conceptuel pour la probl matique de conception et plate-forme(s) op rationnelle(s) pour l'impl mentation et l'ex cution. Le mod le de composant utilis est inspir du framework Rapide [14]. Dans ce mod le (cf. figure 7), les composants communiquent par v nements asynchrones. Le type des composants indique quels v nements ils sont susceptibles de recevoir ou d' mettre.

[6] Par exemple, les missions d' v nements au niveau conceptuel correspondent des invocations synchrones de m thodes au niveau op rationnel. Ceci peut poser probl me si l'on souhaite modifier la s mantique d' mission des v nements (envois asynchrones).

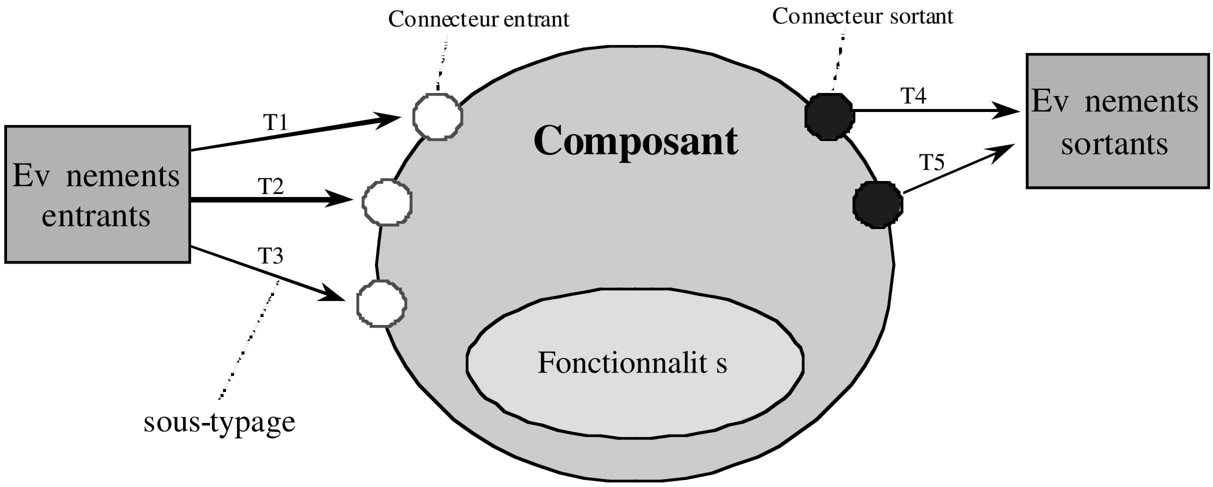


Figure 7. Comet°: mod le de composant, point de vue structurel

Au niveau comportemental, la r ification du contr le se fait dans le cadre d'une m ta-architecture de grain fin, inspir e de CodA [15]. L'id e de base est la r ification des fonctionnalit s de contr les - sous forme de m ta-entit s comparables aux objets - li es au cheminement des v nements au sein d'un composant (cf. figure 8). Cette approche tr s flexible permet la construction de nombreux mod les comportementaux interop rables.

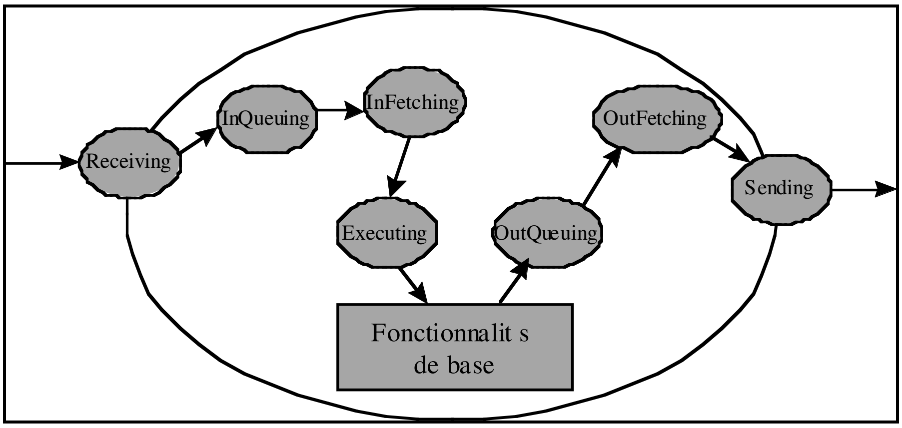


Figure 8. R ification fonctionnelle du contr le dans Comet

L ad quation entre le mod le structurel, le mod le comportemental et - dans un avenir plus lointain - les mod les d impl mentation de Comet est en cours d tude via l utilisation de m thodes formelles.

Au niveau de la granularit , Comet tablit clairement une distinction entre objet (de grain minimal fin) et composant (de grain minimal plus important). Par exemple, les composants Comet sont consid r s comme unit s de r partition/d ploiement. Ceci signifie que la localisation d un composant donn n est pas significative au niveau de l application°: le d ploiement est r alis de fa on compl tement transparente [7].

L'architecture Comet s'int resse galement de fa on approfondie la probl matique de l' volution logicielle [16]. L'id e est d'autoriser les applications tre modifi es (ou s'auto-modifier) dynamiquement. Les possibilit s sont nombreuses ce niveau : modification des architectures (topologie dynamique) ou des composants (adaptabilit dynamique). Les r les et protocoles Comet couvrent ces aspects dynamiques mais leur description d passe le cadre de cet article.

[7] Il est par exemple possible de d ployer de fa on transparente tous les composants d une application sur une seule machine ou sur un vaste r seau de stations de travail.

Ces diff rents concepts ont t impl ment s dans le framework Comet/J bas sur l'environnement Java. Pour valider ce framework, nous avons con u et impl ment quelques syst mes distribu s simples : r plication adaptative, synchronisation d'horloges, etc. Des applications d'orientation industrielle sont aujourd'hui l' tude. D'autres instanciations de l'architecture sont galement pr vues.

Contrairement Maleva, Comet ne propose pas de cadre hi rarchique mais cette extension du mod le est en cours d' tude et semble incontournable.

1.7. Synth se des mod les

Les mod les propos s par Maleva et Comet sont diff rents et con us relativement ind pendamment l'un de l'autre (Maleva est ant rieur Comet) mais tous deux satisfont la notion identitaire de composant que nous proposons. Ils r pondent effectivement aux besoins mis en relief par cette notion°:

- **Besoins d'ordre structurel :**
    - **granularit** : nos approches distinguent sur ce plan composants et objets mais Maleva va encore plus loin en proposant un grain de d part minimal (le grain r el tant fonction du niveau hi rarchique).
    - **couplage** : l'extraction de la relation de couplage structurel est proche dans les deux cas. Comet propose de plus une approche formelle du probl me.
    - **hi rarchisation** : en cours d' laboration dans Comet (approche op rationnelle) et caract ristique centrale de Maleva.
- **Besoins d'ordre comportemental :**
    - **expression du contr le** : Maleva r ifie structurellement le contr le, ce qui le rend manipulable. Comet propose plut t la r ification du contr le dans une approche r flexive.
    - **composition comportementale** : le d couplage entre contr le et fonctionnalit s permet dans les deux cas une discussion sur la s mantique des compositions.
- **Crit re de neutralit :** nos approches, parce qu'elles prennent leur source dans le domaine des architectures logicielles, n'imposent aucune restriction sur la signification ou le r le des composants.

Deux autres caract ristiques communes peuvent tre signal es. Premi rement, Maleva et Comet proposent toutes deux une s paration claire entre le mod le conceptuel et la probl matique de mise en oeuvre. Nous pensons que ceci repr sente une tape indispensable, souvent absente des propositions industrielles. Deuxi mement, les composants Comet/J et Maleva/J sont repr sent s sous forme binaire. La probl matique de composition binaire n'est donc pas lud e, nous la consid rons comme un effet de bord certain, et n cessaire.

Conclusion

Dans cet article, en nous reposant sur de nombreux travaux de recherche, nous tablissons une proposition de notion identitaire de composant logiciel. Des besoins d'ordre structurel et comportemental doivent tre satisfaits pour donner corps cette notion. Les architecture Comet et Maleva, quoique dissemblables, se rejoignent sur ce point. Les approches industrielles, pour leur part, ne satisfont pas l'int gralit des besoins qui conduisent, selon nous, cette notion identitaire de composant logiciel. Par exemple, les JavaBeans proposent un cadre conceptuel tr s int ressant pour ce qui concerne l'assurance de coh sion structurelle des interconnexions de composants.

Cependant, l'aspect comportemental est quasiment lud . Le mod le COM pose galement probl me par l'absence relativement marquante d'abstractions. Le cadre conceptuel propos est peu d velopp , au profit de l'axe technologique. Les mod les EJB et Corba Components semblent pour leur part trop sp cialis s (architectures 3-tiers) et la d l gation totale de la logique syst me peut tre vue comme particuli rement restrictive dans le cadre d'applications o la distinction avec la logique m tier est floue (ex. : gestion fonctionnelle de la s curit ou de tout autre aspect intrusif au niveau contr le).

Pourtant, m me pourvu d'une notion pr gnante de composant, de nombreuses questions restent en suspens. La coh rence s mantique des compositions semble le point le plus sensible ici[8]. Nos approches abordent le probl me mais il n'existe pas - notre connaissance - de solution compl te et d finitive. Dans le cadre d'Unicon [17], ce probl me est trait par la d finition d'une s mantique claire au niveau des connecteurs. Selon nous, il semble cependant difficile de repr senter, au niveau des composants, les contraintes li es aux architectures (nous sugg rons plut t une vision protocolaire pour Comet et graphique - c'est dire bas e sur les graphes d'interconnexion - pour Maleva).

La probl matique g n rale de composition logicielle reste donc encore largement ouverte...

Bibliographie

[1] Cox B., *Object-oriented Programming : An Evolutionary Approach*, Addison-Wesley, 1986.

[2] Booch G., *Software Engineering with ADA*, The Benjamin/Cummings Publishing Compagny Inc., 1983.

[3] Garlan D., Shaw M., *An Introduction to Software Architecture*, AMBRIOLA V., TORTORA G., Eds, Advances in Software Enginnering and Knowledge Engineering, vol. 1, World Scientific Publishing Company, 1993.

[4] Szyperski, C., *Component Software : Beyond Object Oriented Programming*, Eds Addison-Wesley & ACM Press, 1998.

[5] Microsoft, *Microsoft COM Technologies*, http://www.microsoft.com/com, 1999.

[6] Sun Microsystems Inc. *Javabeans Specifications 1.01.* http://java.sun.com/beans/docs/spec.html, 1998.

[7] Sun Microsystems Inc. *JAVA Core Reflection API and Specification.* http://java.sun.com/products/jdk/1.1/docs/guide/reflection/spec/java-reflectionTOC.doc.html, 1998.

[8] Sun Microsystems Inc. *Entreprise Javabeans Specifications 1.1.* http://java.sun.com/products/ejb/docs.html, 2000.

[9] Object Management Group, *Corba Component Model*, http://www.omg.org, 1999

[10] Briot J-P., Guerraoui R., L hr K-P., *Concurrency and Distribution in Object-Oriented Programming*, ACM Computing Surveys, vol. 30, n¡3, p. 291-329, 1998.

[11] Guillemet A., Ha k G., Meurisse T., Briot J-P., and Lhuillier M,. *Une approche base de composants pour la conception d'agents.* Journ es Francophones pour l'Intelligence Artificielle

[8] Soulignons l'intervention de Linda Northrop lors d'un panel Ecoop'99 : *"Components that plug but just don't play"*.

Distribu es et les Syst mes Multi-Agents (JFIADSMA'99), Ile de la R union, Nov. 1999, France. Edition Herm s.

[12] Drogoul A., *De la Simulation Multi-Agents la R solution Collective de Probl mes.* Th se de Doctorat, Universit Paris 6, 1993.

[13] Peschanski F., *Comet : Architecture R flexive Base de Composants pour la Construction d'Applications Concurrentes et R parties.* LMO'2K, 26-28 Janv. 2000, Mont St-Hilaire, Canada. Edition Herm s.

[14] Luckham D. C., *Rapide : A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events*, DIMACS Partial Order Methods Workshop, vol. IV, 1996.

[15] MC Affer J., *Metalevel Programming with Coda*, Proceedings of ECOOP'95, LNCS 952, Springer Verlag, ao t 1995, p. 190-214.

[16] Evans H., DickMan P., *Zones, Contracts and Absorbing Change : An Approach to Software Evolution*, Proceedings of OOPSLA'99, ACM Press, Oct. 1999.

[17] Shaw M., Deline R., Klein D. V., Ross T. L., Toung D. M., Zelesnik G., *Abstractions for Software Architecture and Tools to Support Them*, IEEE Trans. On Software Engineering, vol. SE-21(N.4) avril 1995, p. 314-335.

[18] Sullivan K. J., Marchukov M., and Socha J., *Analysis of a Conflict Between Aggregation and Interface Negotiation in Microsoft s Component Object Model*, IEEE Trans. On Software Engineering, vol. SE-25(N.4) July/August 1999, p. 584-599.

[19] Briand L. C., Daly J. W., and W st J. K., *A Unified Framewoek for Coupling Measurement in Object-Oriented Systems*, IEEE Trans. On Software Engineering, vol. SE-25(N.1) January/February 1999, p. 91-121.

[20] Guerraoui R., *What Object-Oriented Distributed Programming may be — and What itdoes not have to Be.* White Paper. HP Labs. 1999