STL: Un Modèle et Langage de Coordination pour les Systèmes Distribués*

M. Schumacher, O. Krone, F. Chantemargue, B. Hirsbrunner
Département d'Informatique, Groupe PAI, Université de Fribourg, CH-1700 Fribourg, Suisse
URL: http: //www-iiuf.unifr.ch/pai

Résumé

Cet article présente STL, un nouveau modèle de coordination et langage associé pour faciliter le développement d'applications distribuées. L'expressivité de STL est illustrée à travers un exemple classique.

1. Introduction

Les modèles de programmation parallèle existants sont difficilement utilisables pour un large spectre d'implantations distribuées, car ils ne proposent pas de séparation claire entre la partie calcul d'une application distribuée et la partie coordination des entités de calcul: généralement ces deux parties sont inextricables. Pour formaliser et mieux décrire ces inter-dépendances, il est nécessaire de séparer ces deux parties essentielles [3]. La théorie de la Coordination introduite par Malone [9], étudie tout ce qui traite de la gestion des dépendances entre diverses activités. Les principes développés dans cette théorie regroupent des aspects conceptuels et méthodologiques permettant de faciliter l'expression et l'implantation d'applications distribuées par une séparation claire entre la coordination et le calcul. La recherche dans ce domaine s'est concentrée en informatique sur la définition de plusieurs modèles de coordination et langages associés [10]. Un langage de coordination est l'incarnation linguistique d'un modèle de coordination [3] et doit être défini orthogonalement à un langage de calcul. Le plus représentatif de cette classe de langages est Linda [2] qui introduit la notion de communication générative dans un modèle de coordination basé sur une abstraction de type espace de tuples.

2. Le Modèle de Coordination STL

Le modèle de coordination STL, qui reprend quelques caractéristiques du modèle de coordination IWIM [1], comprend cinq éléments: (1) les *Blops*, une abstraction et un mécanisme de modularisation pour processus et ports; (2) les *Processus*, comme entités actives à coordonner; (3) les *Ports*, comme interfaces des processus/blops vers l'extérieur; (4) les *Evènements*, un mécanisme permettant de réagir dynamiquement à des changements d'état; (5) les *Connections*, comme représentation de ports connectés.

La Figure 1 donne une première idée de la métaphore de programmation sur laquelle STL est basée. Une application STL consiste en une hiérarchie de blops dans lesquels plusieurs processus s'exécutent. Les processus communiquent et se coordonnent entre eux via des évènements et des connections. Les ports constituent les points finaux de communication pour les connections qui résultent en des appariements de ports.

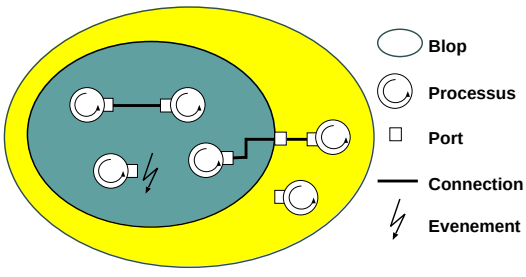

Figure 1: Le Modèle de Coordination de STL.

2.1. Blop

Un blop est une abstraction pour une agglomération d'objets à coordonner. Il sert d'espace privé de noms pour les ports, les processus et les blops subordonnés, ainsi que de mécanisme d'encapsulation pour les évènements. Les blops ont la même interface que les processus, c'est-à-dire un nom et un ensemble de ports. Ils peuvent être hiérarchiquement structurés. On distingue la déclaration d'un blop de son

*Ce travail est partiellement financé par le Fonds National Suisse de la Recherche Scientifique (projet 20-05026.97)

instanciation, exception faite du méta blop par défaut appelé *world*, dans lequel toute activité est confinée. La création d'un blop est gérée de la même façon que la création des processus (voir 2.2): elle inclut l'initialisation de tous les processus et ports statiques définis pour ce blop et les blops subordonnés.

2.2. Processus

Un processus en STL est un objet typé; il possède un nom et dispose d'un ensemble de ports statiques. Comme pour les blops, la gestion des processus en STL est réalisée en deux étapes: déclaration du type de processus, puis son instanciation et invocation. Les processus peuvent également générer des ports dynamiques au cours de leur existence.

Les processus en STL ne disposent d'aucun moyen d'identification; un modèle de type boîte noire ("black box" en anglais) est utilisé. Un processus n'a pas à se soucier du processus vers lequel l'information sera transmise ou du processus duquel l'information a été transmise, puisqu'il communique à travers des ports (voir 2.3). Les processus sont activés soit depuis le langage de coordination statiquement ou dynamiquement par évènement (voir 2.5), soit depuis le langage de calcul dynamiquement. La terminaison d'un processus est implicite.

2.3. Ports

Les ports sont les interfaces des processus et des blops pour établir des connections avec d'autres processus/blops, c'est-à-dire la communication en STL est gérée via une connection et par conséquent par des ports. La communication par port permet de mettre en oeuvre de la communication anonyme pour des entités boîtes noires, ce qui renforce la réutilisabilité de celles-ci.

Un port possède un nom et un jeu d'attributs (voir tableau 1 pour un aperçu). Le nom et les attributs d'un port représentent sa *signature* dans un blop. Un port est *statique* s'il est créé dans le langage de coordination, tandis qu'il est *dynamique* s'il est créé lors de l'exécution dans le langage de calcul. Cependant, le type d'un port dynamique doit être déterminé dans le langage de coordination.

Attributs	**Exemple**	**Explication**
Communication	`blackboard`, `stream`, `group`	Structure de communication
Saturation	`saturation = 7`	7 autres ports peuvent se lier; défaut: 1
Capacity	`capacity = 5`	Capacité d'un port: 5 éléments; défaut: ∞
Synchronization	`synchron`, `asynchron`	Sémantique du modèle passage de messages
Orientation	`in, out, inout`	Direction du flot de données

Tableau 1: Attributs d'un port.

La combinaison des différents attributs d'un port donne le type d'un port. On distingue les principaux types: port de sortie point-à-point (`P2P_o`), port d'entrée point-à-point (`P2P_i`), port bi-directionnel point-à-point (`P2P_io`), groupes (`Group`) et tableaux noirs (`BB`). Des variantes sont possibles et peuvent être définies par l'utilisateur par modification des valeurs d'attributs, comme par exemple pour définir un style de communication de type point-à-point (1: n) en augmentant à `n` l'attribut `saturation` d'un port de type `P2P_o`.

L'appariement ("matching" en anglais) de ports est défini par une relation entre signatures de ports. Cette relation n'est pas statique; elle ne peut pas être déterminée lors de la compilation, car elle dépend de l'état courant des attributs du port. Cinq conditions doivent être remplies pour l'appariement de deux ports: (1) avoir la même valeur d'attribut de communication, (2) avoir le même nom, (3) ne pas être saturés, (4) appartenir au même niveau d'abstraction, c'est-à-dire, être visibles dans la même hiérarchie de blops, et (5) appartenir à des objets différents. De plus amples détails peuvent être trouvés dans [6].

2.4. Connections

Les connections entre les processus peuvent avoir les sémantiques suivantes: (1) Point-à-point: 1: 1, 1: n, n: 1 et n: m types de communication sont possibles; (2) Groupe clos ("closed groups" en anglais): les messages sont diffusés à tous les membres du groupe; (3) Tableau noir: les messages sont placés sur un tableau noir utilisé par plusieurs processus.

2.5. Evènements

Un gestionnaire d'évènement peut être attaché à un port. Une condition détermine quand l'évènement est exécuté dans le blop. Le Tableau 2 présente les différentes conditions considérées. Un évènement est déclenché par le système si et seulement si des données passent sur le port ou un processus accède aux données du port. Une fois qu'un évènement a été déclenché, il faut réinstaller la routine de gestionnaire d'évènement pour pouvoir le déclencher à nouveau. Ceci est généralement fait dans le traitement de l'évènement en cours.

Conditions sur les ports	Explication
`accessed(p)`	Le port a été accédé
`unbound(p)`	Pas de partenaire de communication
`isempty(p)`	Ne contient pas de données
`isfull(p)`	Le port est plein
`msg_handled(p, int n)`	n messages gérés
`less_msg_handled(p, int n)`	$\leq$ n messages gérés

Tableau 2: Conditions sur les ports, `p` symbolise un port.

2.6. Primitives

STL est aussi un langage à part entière qui doit être utilisé en complément à un langage de calcul. Cependant, certains mécanismes de coordination doivent aussi être accédés depuis le langage de calcul pour permettre la gestion des ports et des messages. Ceci est fait en fournissant dans une librairie un jeu de primitives qui permet l'interaction entre le calcul et la coordination d'une application distribuée. Ces primitives sont décrites dans [6].

2.7. Exemple d'application de STL

Pour illustrer STL, nous avons choisi le *crible d'Eratosthène*, dont le but est la détermination des nombres premiers. La figure 3 présente le code pour la partie coordination et la partie calcul et la figure 2 permet de visualiser graphiquement le fonctionnement de l'implantation dans un environnement mono-blop. Notons qu'une version multi-blop, où chaque blop contient un segment du pipeline est très facilement réalisable en STL. Un processus `source` génère une série de nombres et les écrit sur son port `right`. A l'aide de l'évènement `newSieveEvt` lié avec la condition `unbound` aux ports `right` des processus `source` et `sieve`, le pipeline grandit au fur et à mesure que des nombres premiers sont trouvés. Chaque processus `sieve` dans le pipe-line est responsable de filtrer les multiples du nombre premier dont il est responsable (la première valeur qu'il a reçue). Dès qu'il ne s'agit pas d'un multiple, il écrit la valeur sur son port `right`; comme ce dernier n'est pas lié (`unbound`), `newSieveEvt` est déclenché pour créer un nouveau processus `sieve`.

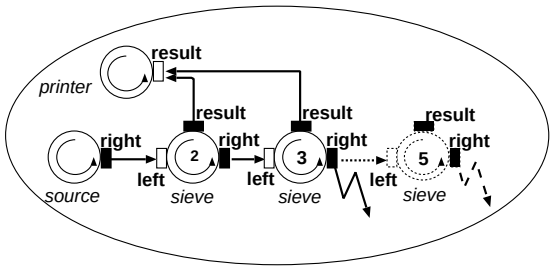

Figure 2: Crible d'Eratosthène en STL.

3. Conclusion

Dans cet article, nous avons présenté STL, un nouveau modèle de coordination et langage associé. Bien que STL ait des similarités avec MANIFOLD [1], ConCoord [5], Darwin [8] ou Linda [2], il diffère cependant en plusieurs points. Tout d'abord, STL permet à l'utilisateur de définir plusieurs types de ports, offrant différentes métaphores de communication comme la communication générative. Deuxièmement, en utilisant un langage de description imbriqué pour spécifier les hiérarchies des espaces de coordination, le modèle STL semble plus explicite que celui utilisé dans MANIFOLD . Les blops ne sont pas seulement une sorte de processus contrôlant la coordination, mais ils sont également un espace de noms pour des objets de type ports et ils offrent aussi un mécanisme de modularisation pour la gestion des évènements.

```
blop world() {
  process printer(PORTS result) {
    P2P1N_i result("PRINTER");
  }
  process source(VALUES n PORTS right){
    P2P_o right("NEW-SIEVE");
  }
  process sieve(PORTS left right result){
    P2P_i left("NEW-SIEVE");
    P2P_o right("NEW-SIEVE");
    P2P_o result("PRINTER");
  }
  event newSieveEvt {
    create process sieve s;
    when unbound(s.right) then newSieveEvt;
  }
  create process printer p;
  create process source so;
  when unbound(so.right) then newSieveEvt;
}
```

```
void source(int n, P2P_o right) {
  int counter = 2;
  while (counter < n) {
    Msg valueMsg(counter);
    right.put(0, valueMsg);
    counter++;
  }}

void sieve(P2P_i left, P2P_o right,
                       P2P_o result) {
  IntTempl val;
  Msg newVal(val);
  left.get(newVal);
  int myPrime = val;
  result.put(newVal);
  while(TRUE) {
    left.get(newVal);
    if (value % myPrime){
      right.put(newVal);
    }}}
```

Figure 3: Extraits de code du Crible d'Eratosthène: Partie coordination en STL (à gauche), Partie calcul (à droite). Le processus Printer est volontairement omis car il n'a pour but que d'afficher les nombres premiers.

Nous avons réalisé une plateforme de coordination basée sur le modèle de coordination STL. Ce premier prototype STL a été développé au-dessus de la plateforme déjà existante Pt-PVM [7]. Cette dernière, implantée sur PVM, fournit des facilités de passage de messages et de gestion de processus, au niveau de processus légers et lourds pour un réseau de stations de travail UNIX. STL est aussi utilisé dans le domaine de l'intelligence artificielle distribuée pour implanter des systèmes multi-agents sur des architectures distribuées (par exemple pour simuler une application de robotique collective) [4] [11].

Bibliographie

1. F. Arbab. The IWIM Model for Coordination of Concurrent Activities. In Paolo Ciancarini and Chris Hankin, editors, *Proceedings of the First International Conference on Coordination Models, Languages and Applications*, number 1061 in LNCS. Springer Verlag, April 1996.
2. N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4): 444–458, 1989.
3. N. Carriero and D. Gelernter. Coordination Languages and Their Significance. *Communications of the ACM*, 35(2): 97–107, February 1992.
4. F. Chantemargue, O. Krone, M. Schumacher, T. Dagaeff, and B. Hirsbrunner. Autonomous Agents: from Concepts to Implementation. In *Proceedings of the Fourteenth European Meeting on Cybernetics and Systems Research (EMCSR'98)*, Vienna, Austria, April 14-17 1998.
5. A.A. Holzbacher. A Software Environment for Concurrent Coordinated Programming. In Paolo Ciancarini and Chris Hankin, editors, *Proceedings of the First International Conference on Coordination Models, Languages and Applications*, number 1061 in LNCS. Springer Verlag, April 1996.
6. O. Krone. *STL and Pt-PVM: Concepts and Tools for Coordination of Multi-threaded Applications.* PhD thesis, University of Fribourg, 1997.
7. O. Krone, B. Hirsbrunner, and V.S. Sunderam. PT-PVM+: A Portable Platform for Multithreaded Coordination Languages. *Calculateurs Parallèles*, 8(2): 167–182, 1996.
8. J. Magee, N.Dulay, and J. Kramer. Structuring Parallel and Distributed Programs. *Software Engineering Journal*, pages 73–82, March 1993.
9. T.W. Malone and K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1): 87–119, March 1994.
10. G. A. Papadopoulos and F. Arbab. Coordination Models and Languages. In M. Zelkowitz, editor, *Advances in Computers, The Engineering of Large Systems*, volume 46. Academic Press, August 1998.
11. M. Schumacher, F. Chantemargue, T. Dagaeff, O. Krone, and B. Hirsbrunner. STL++: A Coordination Language for Autonomy-based Multi-Agent Systems. Technical report, Computer Science Department, University of Fribourg, Fribourg, Switzerland, March 1998.