Coordination Languages

Chris Hankin
c.hankin@imperial.ac.uk

Department of Computing, Imperial College London

BCS Advanced Programming Group Lecture, December 2008

Outline

1 Introduction

2 Linda

3 JavaSpaces

4 KLAIM

5 AspectK

6 Example Programs

7 Conclusions

Outline

1. Introduction
2. Linda
3. JavaSpaces
4. KLAIM
5. AspectK
6. Example Programs
7. Conclusions

Outline

1 Introduction

2 Linda

3 JavaSpaces

4 KLAIM

5 AspectK

6 Example Programs

7 Conclusions

Outline

1. Introduction
2. Linda
3. JavaSpaces
4. KLAIM
5. AspectK
6. Example Programs
7. Conclusions

Outline

1. Introduction
2. Linda
3. JavaSpaces
4. KLAIM
5. AspectK
6. Example Programs
7. Conclusions

Outline

1 Introduction

2 Linda

3 JavaSpaces

4 KLAIM

5 AspectK

6 Example Programs

7 Conclusions

Outline

1. Introduction
2. Linda
3. JavaSpaces
4. KLAIM
5. AspectK
6. Example Programs
7. Conclusions

Background

The term Coordination Language was coined by Gelernter and Carriero in the context of Linda.

Concurrent programming = Computation + Coordination

Since their seminal work, a number of new languages have been proposed and described as coordination languages.

We focus on coordination models based on generative communication via a shared dataspace – theoretically this can be seen as a broadcast mode of communication

Applications: e-commerce, game playing, internet services, workflow management

Background

The term Coordination Language was coined by Gelernter and Carriero in the context of Linda.

Concurrent programming = Computation + Coordination

Since their seminal work, a number of new languages have been proposed and described as coordination languages.

We focus on coordination models based on generative communication via a shared dataspace – theoretically this can be seen as a broadcast mode of communication

Applications: e-commerce, game playing, internet services, workflow management

Background

The term Coordination Language was coined by Gelernter and Carriero in the context of Linda.

Concurrent programming = Computation + Coordination

Since their seminal work, a number of new languages have been proposed and described as coordination languages.

We focus on coordination models based on generative communication via a shared dataspace – theoretically this can be seen as a broadcast mode of communication

Applications: e-commerce, game playing, internet services, workflow management

Background

The term Coordination Language was coined by Gelernter and Carriero in the context of Linda.

Concurrent programming = Computation + Coordination

Since their seminal work, a number of new languages have been proposed and described as coordination languages.

We focus on coordination models based on generative communication via a shared dataspace – theoretically this can be seen as a broadcast mode of communication

Applications: e-commerce, game playing, internet services, workflow management

Background

The term Coordination Language was coined by Gelernter and Carriero in the context of Linda.

Concurrent programming = Computation + Coordination

Since their seminal work, a number of new languages have been proposed and described as coordination languages.

We focus on coordination models based on generative communication via a shared dataspace – theoretically this can be seen as a broadcast mode of communication

Applications: e-commerce, game playing, internet services, workflow management

Issues

In defining what a coordination language might be there are a number of issues which must be addressed:

1. what are the entities which are being coordinated?
2. what are the media for coordination?
3. what are the protocols and rules used for coordination?

Issues

In defining what a coordination language might be there are a number of issues which must be addressed:

1. what are the entities which are being coordinated?
2. what are the media for coordination?
3. what are the protocols and rules used for coordination?

Issues

In defining what a coordination language might be there are a number of issues which must be addressed:

1. what are the entities which are being coordinated?
2. what are the media for coordination?
3. what are the protocols and rules used for coordination?

General Observations

- Coordination languages are not general purpose programming languages, in particular they do not need to be Turing complete; rather, they are usually defined as language extensions or scripting languages.

- Coordination languages are most relevant in the context of open systems, where the coordinated entities are not fixed at the outset.

General Observations

- Coordination languages are not general purpose programming languages, in particular they do not need to be Turing complete; rather, they are usually defined as language extensions or scripting languages.
- Coordination languages are most relevant in the context of open systems, where the coordinated entities are not fixed at the outset.

Components

- Coordinated entities: There is general agreement that the coordinated entities should be active – agents or processes. Coordination of agents should not require reprogramming; the coordination mechanism is a wrapper around the existing, independent agents.
- Coordination media: Coordination is often accomplished via a shared data space, typically a tuple space, multiset or partitioned (multi-)set. In such models, communication is *generative*.
- Coordination rules In contrast to Linda, many of the recent proposals have been for rule-based languages; one consequence of this shift to a more declarative view of coordination is increased reasoning power.

Components

- Coordinated entities: There is general agreement that the coordinated entities should be active – agents or processes. Coordination of agents should not require reprogramming; the coordination mechanism is a wrapper around the existing, independent agents.
- Coordination media: Coordination is often accomplished via a shared data space, typically a tuple space, multiset or partitioned (multi-)set. In such models, communication is *generative*.
- Coordination rules In contrast to Linda, many of the recent proposals have been for rule-based languages; one consequence of this shift to a more declarative view of coordination is increased reasoning power.

Components

- Coordinated entities: There is general agreement that the coordinated entities should be active – agents or processes. Coordination of agents should not require reprogramming; the coordination mechanism is a wrapper around the existing, independent agents.
- Coordination media: Coordination is often accomplished via a shared data space, typically a tuple space, multiset or partitioned (multi-)set. In such models, communication is *generative*.
- Coordination rules In contrast to Linda, many of the recent proposals have been for rule-based languages; one consequence of this shift to a more declarative view of coordination is increased reasoning power.

Linda

The syntax of $\mathcal{L}(X)$ is formally defined by the following grammar:

$$
\begin{array}{lcl}
P & ::= & \textbf{stop} \mid C.P \mid P \mid P \mid P + P \\
C & ::= & \textbf{ask}(t) \mid \textbf{tell}(t) \mid \textbf{get}(t) \mid \textbf{eval}(P)
\end{array}
$$

t is a generic element called token in a denumerable set $\mathcal{D}$, P is a process and C a communication action (or prefix), the set of all processes is denoted by $\mathcal{P}$.

The parameter X defining a Linda-like language $\mathcal{L}(X)$ is a subset of the primitives defined by C.

Linda

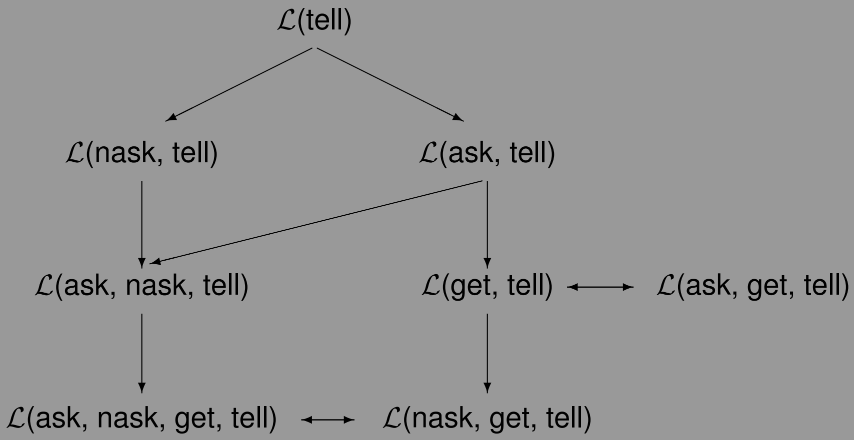
The syntax of $\mathcal{L}(X)$ is formally defined by the following grammar:

$$
\begin{array}{lcl}
P & ::= & \textbf{stop} \mid C.P \mid P \mid P \mid P + P \\
C & ::= & \textbf{ask}(t) \mid \textbf{tell}(t) \mid \textbf{get}(t) \mid \textbf{eval}(P)
\end{array}
$$

t is a generic element called token in a denumerable set $\mathcal{D}$, P is a process and C a communication action (or prefix), the set of all processes is denoted by $\mathcal{P}$.

The parameter X defining a Linda-like language $\mathcal{L}(X)$ is a subset of the primitives defined by C.

Linda Hierarchy



Dining Philosophers

```
philosopher(int i)
{
  while(TRUE){
   think();
   in(”meal ticket”); in(”fork”,i);
   in(”fork”,(i+1)%5);
   eat()
   out(”fork”,i); out(”fork”,(i+1)%5);
   out(”meal ticket”);
  }
}
```

Dining Philosophers contd.

```
real_main()
{
  int i;
  for (i=0,i<5, i++){
   out(”fork”,i);
   eval(philosopher(i));
   if (i<4) out(”meal ticket”);
  }
}
```

Javaspaces philosophy

- Based on the concept of shared network-based persistent space that is used for both object storage and as an exchange space
- Simple API that is easy to learn but expressive for building sophisticated distributed applications
- Objects are passive.

Javaspaces philosophy

- Based on the concept of shared network-based persistent space that is used for both object storage and as an exchange space
- Simple API that is easy to learn but expressive for building sophisticated distributed applications
- Objects are passive.

Javaspaces philosophy

- Based on the concept of shared network-based persistent space that is used for both object storage and as an exchange space
- Simple API that is easy to learn but expressive for building sophisticated distributed applications
- Objects are passive.

JavaSpaces

Four primary operations on a Javaspaces service:

- `write()`: Writes new objects into a space
- `take()`: Retrieves objects from a space
- `read()`: Makes a copy of objects in a space
- `notify`: Notifies a specified object when entries that match the given template are written into a space

KLAIM philosophy

- The KLAIM language (Kernel Language for Agents Interaction and mobility) was introduced by De Nicola et al as a distributed mobile version of Linda
- It extends the Linda interaction model by replacing the single shared tuple space with multiple distributed tuples spaces
- It also allows explicit manipulation of localities and locality names

KLAIM philosophy

- The KLAIM language (Kernel Language for Agents Interaction and mobility) was introduced by De Nicola et al as a distributed mobile version of Linda
- It extends the Linda interaction model by replacing the single shared tuple space with multiple distributed tuples spaces
- It also allows explicit manipulation of localities and locality names

KLAIM philosophy

- The KLAIM language (Kernel Language for Agents Interaction and mobility) was introduced by De Nicola et al as a distributed mobile version of Linda
- It extends the Linda interaction model by replacing the single shared tuple space with multiple distributed tuples spaces
- It also allows explicit manipulation of localities and locality names

KLAIM Syntax

$N \in$ **Net**	N	$::=$	$N_1 \parallel N_2 \mid l :: P \mid l :: \langle \overrightarrow{l} \rangle$
$P \in$ **Proc**	P	$::=$	$P_1 \mid P_2 \mid \sum_i a_i.P_i \mid *P$
$a \in$ **Act**	a	$::=$	$\mathbf{out}(\overrightarrow{\ell})@\ell \mid \mathbf{in}(\overrightarrow{\ell^\lambda})@\ell \mid \mathbf{read}(\overrightarrow{\ell^\lambda})@\ell$
$\ell, \ell^\lambda \in$ **Loc**	ℓ	$::=$	$u \mid l \qquad \ell^\lambda ::= \ell \mid !u$

KLAIM Semantics

- Well-formedness conditions on use of variable names
- Structural Congruence
- Reaction Semantics
- Pattern Matching of Templates against Tuples

KLAIM Example

$$\begin{aligned}\text{User} :: \quad & \mathbf{read}(!\mathit{name}, !\mathit{telno})@\mathrm{YP}. \\ & \mathbf{read}(\mathit{telno}, !\mathit{val}_1, !\mathit{val}_2)@\mathrm{DB}. \\ & \mathbf{out}(\mathit{val}_1)@\mathit{name}\end{aligned}$$

Access Control

- Discretionary Access Control:
  1. Access Control Matrix DAC containing triples (s, o, a)
  2. Whenever User is performing a **read** action on a location l a reference monitor will check whether (User, l, **read**) $\in$ DAC
  3. Similarly for **out** actions
- Mandatory Access Control:
  1. We assign DB the level high and YP the level low
  2. A low user can only perform **read** actions on YP whereas **out** actions can be performed on any location
  3. A high user, on the other hand, will be able to perform **read** actions on both YP and DB. The out action can only be performed on high locations unless a notion of declassification is imposed that will lower the users' security level.

AspectK Syntax

$S \in \textbf{System}$	S	$::=$	$\textbf{let } \overrightarrow{asp} \textbf{ in } N$
$asp \in \textbf{Asp}$	asp	$::=$	$A[cut] \triangleq body$
$body \in \textbf{Advice}$	$body$	$::=$	$\textbf{case}\ (cond)\ sbody\ ;\ body \mid sbody$
	$sbody$	$::=$	$as\ \textbf{break} \mid as\ \textbf{proceed}\ as$
$as \in \textbf{Act}^*$	as	$::=$	$a.as \mid \varepsilon$
$cond \in \textbf{BExp}$	$cond$	$::=$	$\textbf{test}(\overrightarrow{\ell^\lambda})@\ell \mid \ell_1 = \ell_2 \mid cond_1 \wedge cond_2 \mid \neg\ cond$
$cut \in \textbf{Cut}$	cut	$::=$	$\ell :: a$
$\ell^\lambda \in \textbf{Loc}$	ℓ^λ	$::=$	$\ell \mid !u \mid ?u$

AspectK Semantics

$$\frac{N \rightarrow N' \quad \text{(where globally } \Gamma_A = \overrightarrow{asp}\text{)}}{\textbf{let } \overrightarrow{asp} \textbf{ in } N \rightarrow \textbf{let } \overrightarrow{asp} \textbf{ in } N'}$$

$$l_s :: \underline{\textbf{stop}}.P + \cdots \rightarrow l_s :: 0$$

$$l_s :: \underline{\textbf{out}}(\overrightarrow{I})@l_0.P + \cdots \rightarrow l_s :: P \mid\mid l_0 :: \langle \overrightarrow{I} \rangle$$

$$l_s :: \underline{\textbf{in}}(\overrightarrow{\ell^\lambda})@l_0.P + \cdots \mid\mid l_0 :: \langle \overrightarrow{I} \rangle \rightarrow l_s :: P\theta \quad \text{if } match(\overrightarrow{\ell^\lambda}; \overrightarrow{I}) = \theta$$

$$l_s :: \underline{\textbf{read}}(\overrightarrow{\ell^\lambda})@l_0.P + \cdots \mid\mid l_0 :: \langle \overrightarrow{I} \rangle \rightarrow l_s :: P\theta \mid\mid l_0 :: \langle \overrightarrow{I} \rangle \quad \text{if } match(\overrightarrow{\ell^\lambda}; \overrightarrow{I}) = \theta$$

AspectK Semantics contd...

$$\frac{l_s :: \Phi_{\textbf{proceed}}(\Gamma_A; l_s :: \textbf{out}(\overrightarrow{l})@l_0).P \rightarrow N}{l_s :: \textbf{out}(\overrightarrow{l})@l_0.P + \cdots \rightarrow N}$$

$$\frac{l_s :: \Phi_{\textbf{proceed}}(\Gamma_A; l_s :: \textbf{in}(\overrightarrow{\ell^\lambda})@l_0).P \mid\mid N' \rightarrow N}{l_s :: \textbf{in}(\overrightarrow{\ell^\lambda})@l_0.P + \cdots \mid\mid N' \rightarrow N}$$

$$\frac{l_s :: \Phi_{\textbf{proceed}}(\Gamma_A; l_s :: \textbf{read}(\overrightarrow{\ell^\lambda})@l_0).P \mid\mid N' \rightarrow N}{l_s :: \textbf{read}(\overrightarrow{\ell^\lambda})@l_0.P + \cdots \mid\mid N' \rightarrow N}$$

The Φ function

- The result of $\Phi_f(\Gamma_A; \ell :: a)$ is a sequence of actions trapping $\ell :: a$; Γ_A is a global environment of aspects. The index f is either **proceed** or **break**
- In the case of **proceed** the action $\underline{a}$ is eventually emitted
- Otherwise the action is dispensed with and replaced by **stop**
- Advice is searched in declaration order and applies in a parenthesis-like fashion.

DAC

Discretionary access control can be imposed by introducing a location DAC containing two kinds of triples

- $\langle \mathit{user}, \mathrm{DB}, \mathbf{read} \rangle$ for selected users, and
- $\langle \mathit{user}, \mathit{name}, \mathbf{out} \rangle$ for the same selected users and all names.

The following aspect declarations will then impose the desired requirements:

$$A^{read}_{\mathrm{DAC}}[u :: \mathbf{read}(?x, ?y, ?z)@\mathrm{DB}] \triangleq \mathbf{case}(\mathbf{test}(u, \mathrm{DB}, \mathbf{read})@\mathrm{DAC})\ \mathbf{proceed};\ \mathbf{break}$$

$$A^{out}_{\mathrm{DAC}}[u :: \mathbf{out}(z)@l] \triangleq \mathbf{case}(\mathbf{test}(u, l, \mathbf{out})@\mathrm{DAC})\ \mathbf{proceed};\ \mathbf{break}$$

DAC contd...

Using aspects it is easy to modify the access control policy so as to allow a user to access his own entries in DB even though he does not have access to the complete database. We simply modify the aspect A^{read}_{DAC} to become

$$A^{read}_{DAC-1}[u :: \textbf{read}(!x, ?y, ?z)@\text{DB}] \triangleq \textbf{break}$$

$$A^{read}_{DAC-2}[u :: \textbf{read}(x, ?y, ?z)@\text{DB}] \triangleq \textbf{case}(\textbf{test}(u, \text{DB}, \textbf{read})@\text{DAC} \vee \textbf{test}(u, x)@\text{YP}) \; \textbf{proceed}; \; \textbf{break}$$

MAC

For the mandatory access control policy we introduce a location MAC with the following pairs:

- $\langle \mathrm{YP}, \mathrm{low} \rangle$ reflecting that the phonebook has low security level,
- $\langle \mathrm{DB}, \mathrm{high} \rangle$ reflecting that the customer database has high security level,
- $\langle s, \mathrm{low} \rangle$ for all users and names s with low security level, and
- $\langle s, \mathrm{high} \rangle$ for all users and names s with high security level.

We now consider the Bell-LaPadula security policy in a setting where both subjects and objects have fixed security levels.

MAC contd ...

The first part of the policy states that a subject is allowed to read or input data from any object provided that the object's security level dominates that of the object; this is captured by the following aspects (which enforce *no read-up*):

$$A^{read_2}_{\mathrm{MAC}}[u :: \mathbf{read}(?x, ?y)@l] \triangleq \mathbf{case}(\neg(\mathbf{test}(u, \mathrm{low})@\mathrm{MAC} \wedge \mathbf{test}(l, \mathrm{high})@\mathrm{MAC})) \quad \mathbf{proceed}; \quad \mathbf{break}$$

$$A^{read_3}_{\mathrm{MAC}}[u :: \mathbf{read}(?x, ?y, ?z)@l] \triangleq \mathbf{case}(\neg(\mathbf{test}(u, \mathrm{low})@\mathrm{MAC} \wedge \mathbf{test}(l, \mathrm{high})@\mathrm{MAC})) \quad \mathbf{proceed}; \quad \mathbf{break}$$

MAC contd ...

The second part of the policy, the star property, allows a subject to write to any object provided that the security level of the object dominates that of the subject. This is captured by the following aspect (enforcing *no write-down*):

$$\begin{aligned} \mathrm{A}^{out}_{\mathrm{MAC}}[u :: \mathbf{out}(z)@l] \quad \triangleq \quad & \mathbf{case}(\neg(\mathbf{test}(u, \mathrm{high})@\mathrm{MAC} \wedge \\ & \mathbf{test}(l, \mathrm{low})@\mathrm{MAC})) \\ & \quad \mathbf{proceed}; \\ & \quad \mathbf{break} \end{aligned}$$

Declassification

In order to allow a high user to write to a low name we may introduce *declassification* of security levels. To keep things simple we may do so by introducing a billing location that does not need to adhere to the security policy and replace the process by:

$$\begin{array}{lll} & \text{User} :: & \mathbf{read}(!name, !key)@\text{YP}. \\ & & \mathbf{read}(key, !val_1, !val_2)@\text{DB}. \\ & & \mathbf{out}(name, val_1, val_2)@\text{Billing} \\ \| & \text{Billing} :: & \mathbf{in}(!n, !v_1, !v_2)@\text{Billing}.\ \mathbf{out}(v_1)@n \end{array}$$

We add the pair $\langle \text{Billing}, \text{high} \rangle$ to the MAC location thereby allowing all high users to output to Billing.

Declassification contd...

We also modify the aspect for **out** actions to ensure that they are always allowed to **proceed** at the Billing location:

$$\mathrm{A}^{out}_{\mathrm{MAC}}[u :: \mathbf{out}(z)@l] \triangleq \mathbf{case}(\neg(\mathbf{test}(u, \mathrm{high})@\mathrm{MAC} \wedge \mathbf{test}(l, \mathrm{low})@\mathrm{MAC}) \vee (u = \mathrm{Billing}))\ \mathbf{proceed};\ \mathbf{break}$$

Logging

As a final example, which illustrates the need for actions both before and after **proceed** we define an aspect which maintains a log of **read** action on DB:

$$A_{\mathrm{LOG}}[u :: \mathbf{read}(?x, ?y, ?z)@\mathrm{DB}] \triangleq \begin{array}{l} \mathbf{in}(sem)@\mathrm{semaphore} \\ \mathbf{proceed} \\ \mathbf{out}(u, x, y, z)@\mathrm{logfile}. \\ \mathbf{out}(sem)@\mathrm{semaphore} \end{array}$$

Conclusions

Features studied:

- Shared space(s) and generative communication
- Located spaces and processes
- Mobility of processes and data
- Aspect-oriented programming
  - Distributed policies
  - Composition of policies
  - Verification

Conclusions

Features studied:

- Shared space(s) and generative communication
- Located spaces and processes
- Mobility of processes and data
- Aspect-oriented programming
  - Distributed policies
  - Composition of policies
  - Verification

Conclusions

Features studied:

- Shared space(s) and generative communication
- Located spaces and processes
- Mobility of processes and data
- Aspect-oriented programming
  - Distributed policies
  - Composition of policies
  - Verification

Conclusions

Features studied:

- Shared space(s) and generative communication
- Located spaces and processes
- Mobility of processes and data
- Aspect-oriented programming
  - Distributed policies
  - Composition of policies
  - Verification

Major Contributors

- N. Carriero and D. Gelernter
- J.-P. Banâtre and D. Le Métayer
- R. De Nicola
- P. Ciancarini
- F. Arbab

Major Contributors

- N. Carriero and D. Gelernter
- J.-P. Banâtre and D. Le Métayer
- R. De Nicola
- P. Ciancarini
- F. Arbab

Major Contributors

- N. Carriero and D. Gelernter
- J.-P. Banâtre and D. Le Métayer
- R. De Nicola
- P. Ciancarini
- F. Arbab

Major Contributors

- N. Carriero and D. Gelernter
- J.-P. Banâtre and D. Le Métayer
- R. De Nicola
- P. Ciancarini
- F. Arbab

Major Contributors

- N. Carriero and D. Gelernter
- J.-P. Banâtre and D. Le Métayer
- R. De Nicola
- P. Ciancarini
- F. Arbab

End Notes

- Short Bibliography
- Collaborators:
  - D. Le Métayer
  - D. Sands
  - H. Wiklicky and A. Di Pierro
  - F. Nielson and H. R. Nielson
- Systems:
  - Sun Javaspaces
  - IBM TSpaces
  - KLAIM: Klava

End Notes

- Short Bibliography
- Collaborators:
  - D. Le Métayer
  - D. Sands
  - H. Wiklicky and A. Di Pierro
  - F. Nielson and H. R. Nielson
- Systems:
  - Sun Javaspaces
  - IBM TSpaces
  - KLAIM: Klava

End Notes

- Short Bibliography
- Collaborators:
  - D. Le Métayer
  - D. Sands
  - H. Wiklicky and A. Di Pierro
  - F. Nielson and H. R. Nielson
- Systems:
  - Sun Javaspaces
  - IBM TSpaces
  - KLAIM: Klava