Interaksi Objek

Pendahuluan

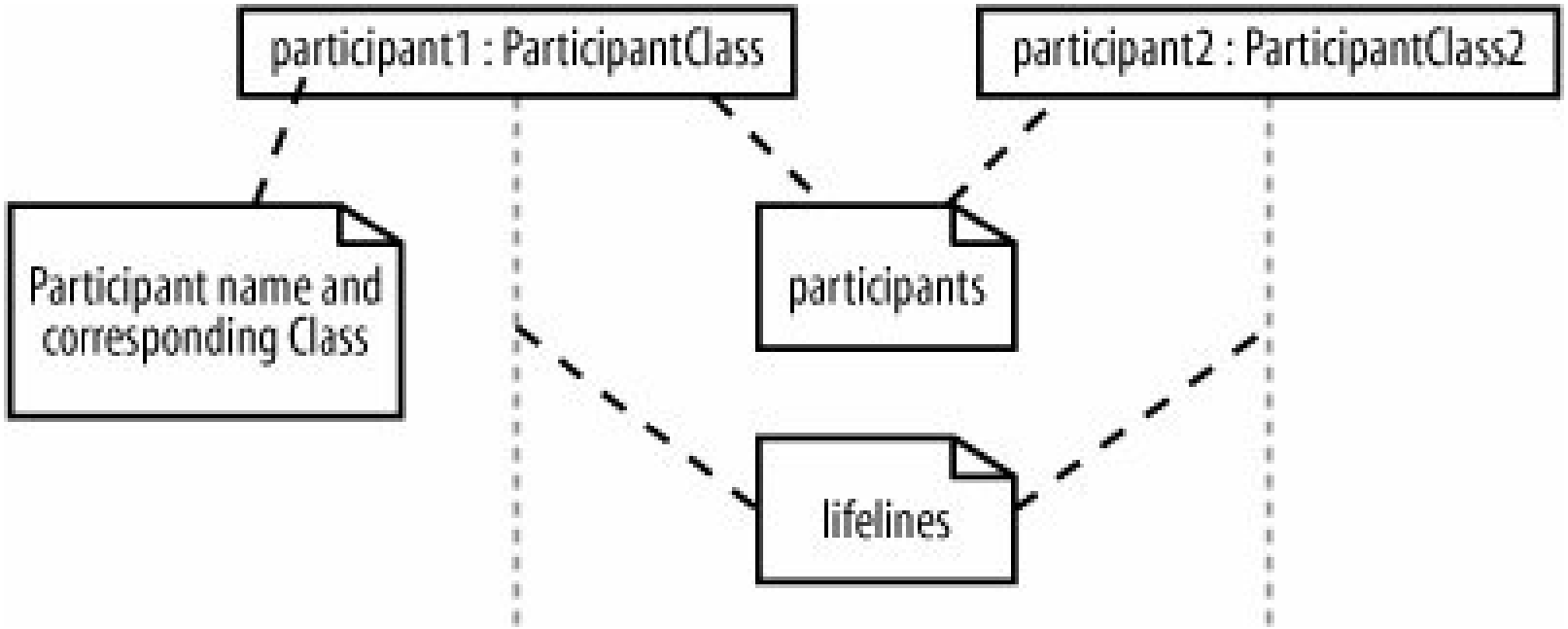
- Komunikasi dan kolaborasi antra objek merupakan sesuatu yang fundamental pada pendekatan berorientasi objek. Pada UML 2.0 menyediakan beberapa diagram untuk memodelkan interaksi antara objek diantaranya adalah :
- Sequence Diagram
- Communication Diagram
- Overview Diagram dan
- Timing Diagram.

Sequence Diagram

- Merupakan salah satu diagram interaksi yang terdapat pada UML.
- Secara sematik hampir mirip dengan communication diagram jika kasusnya sederhana.
- Sequence Diagram menggambarkan komunikasi antar objek sesuai dengan urutan waktu.
- Sequence diagram bisa digambarkan dengan level yang lebih rinci sesuai dengan tujuan yang akan dicapai.

Participant pada Seq Diagram

- Sequence diagram terbentuk atas kumpulan participant sebagai bagian dari sistem yang saling berinteraksi antara satu dengan yang lain.
- Participant tidak boleh tumpang tindih antara satu dengan yang lain.


participant1 : ParticipantClass
participant2 : ParticipantClass2
Participant name and corresponding Class
participants
lifelines

Participant pada Seq Diagram

- Pada UML 1.x sebuah participant biasanya merupakan sebuah sistem. Namun pada perkembangan selanjutnya pada UML 2.x, participant adalah bagian yang terlibat dalam interaksi pada sequence diagram

Menamakan Participant

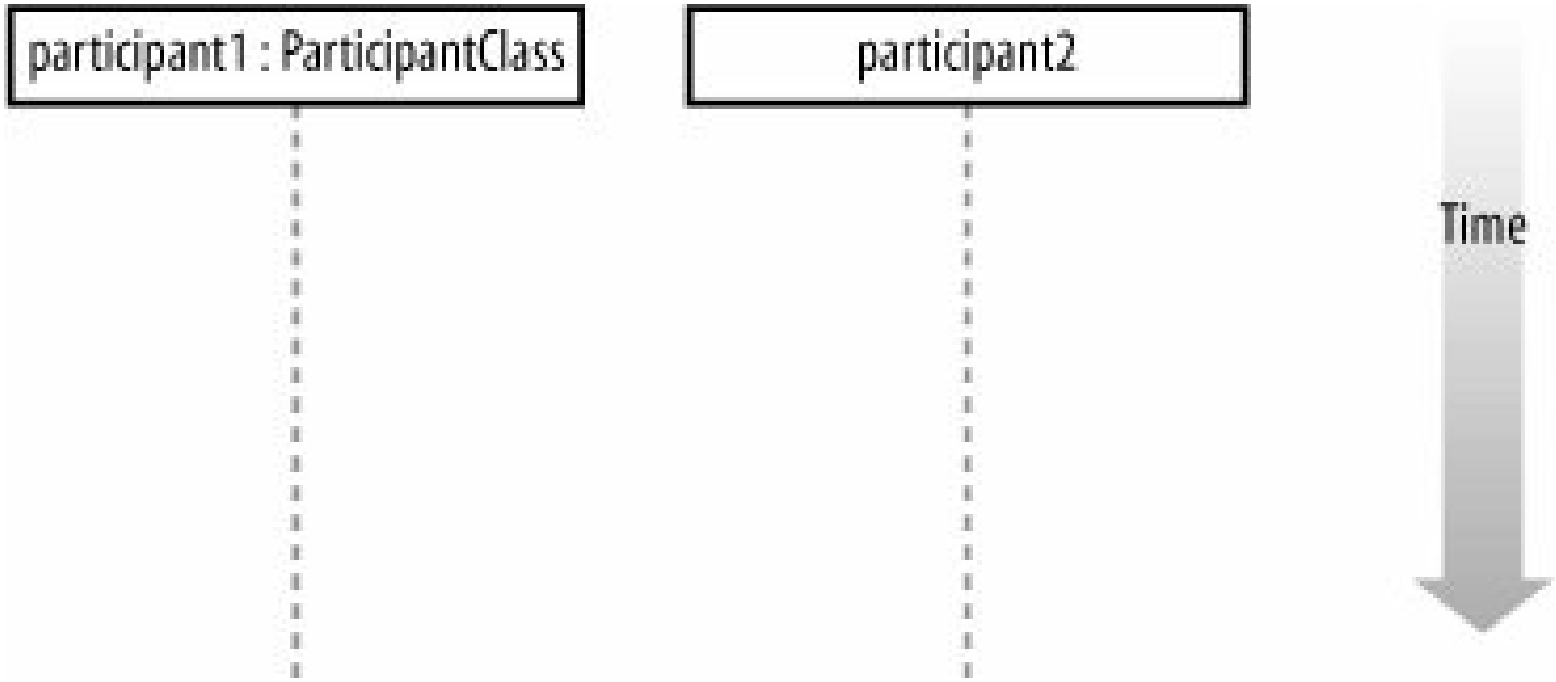
- Untuk memberi nama participant dilakukan dalam bentuk

name [selector] : class_name ref decomposition

Nama	Keterangan
admin	Bagian dari admin namun tidak ditetapkan sebagai suatu kelas
: ContentManagementSystem	Participant bentuk class yaitu class Content Management System
admin : Administrator	Merupakan Objek Admin dari Class Administrator
eventHandlers [2] : EventHandler	Akses Elemen kedua dari Class Event handler
: ContentManagementSystem ref cmsInteraction	Class Content Management System dimana participant bekerja secara internal dengan ClasscmsIteraction

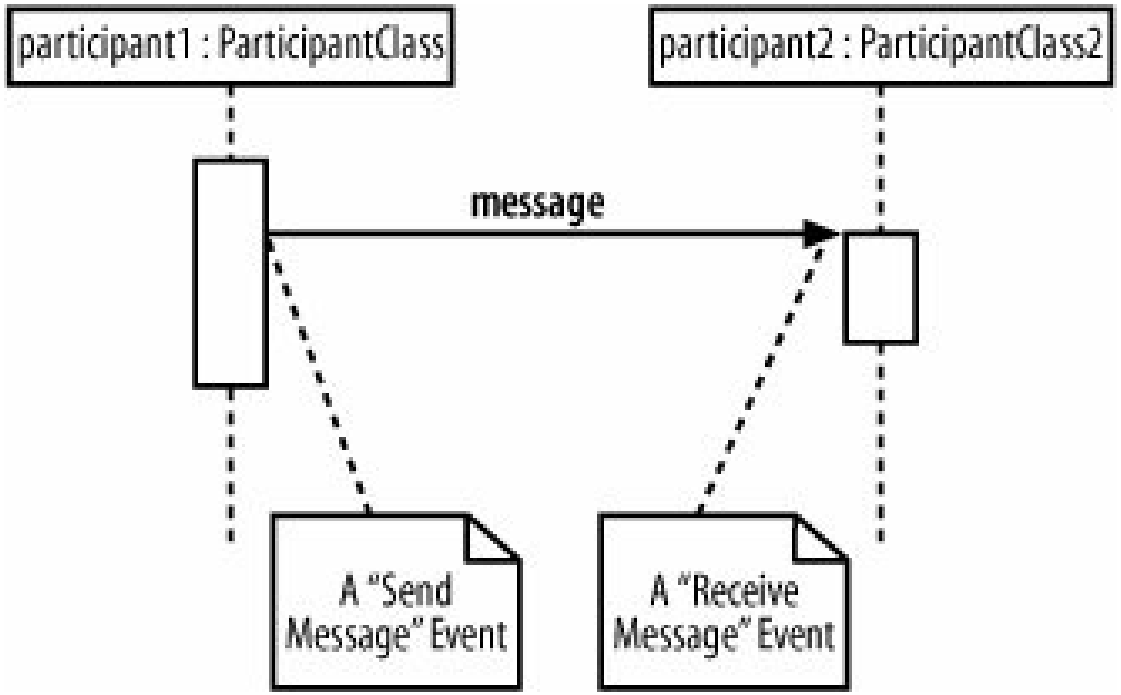
Time

- Sebuah Sequence Diagram menggambarkan urutan interaksi berlangsung, sehingga waktu adalah faktor penting.
- Waktu pada Sequence Diagram dimulai di bagian atas halaman, tepat di bawah Participant paling atas judul, dan kemudian berkembang ke bawah halaman. Urutan bahwa interaksi ditempatkan menunjukkan urutan di mana mereka interaksi.


participant1 : ParticipantClass
participant2
Time

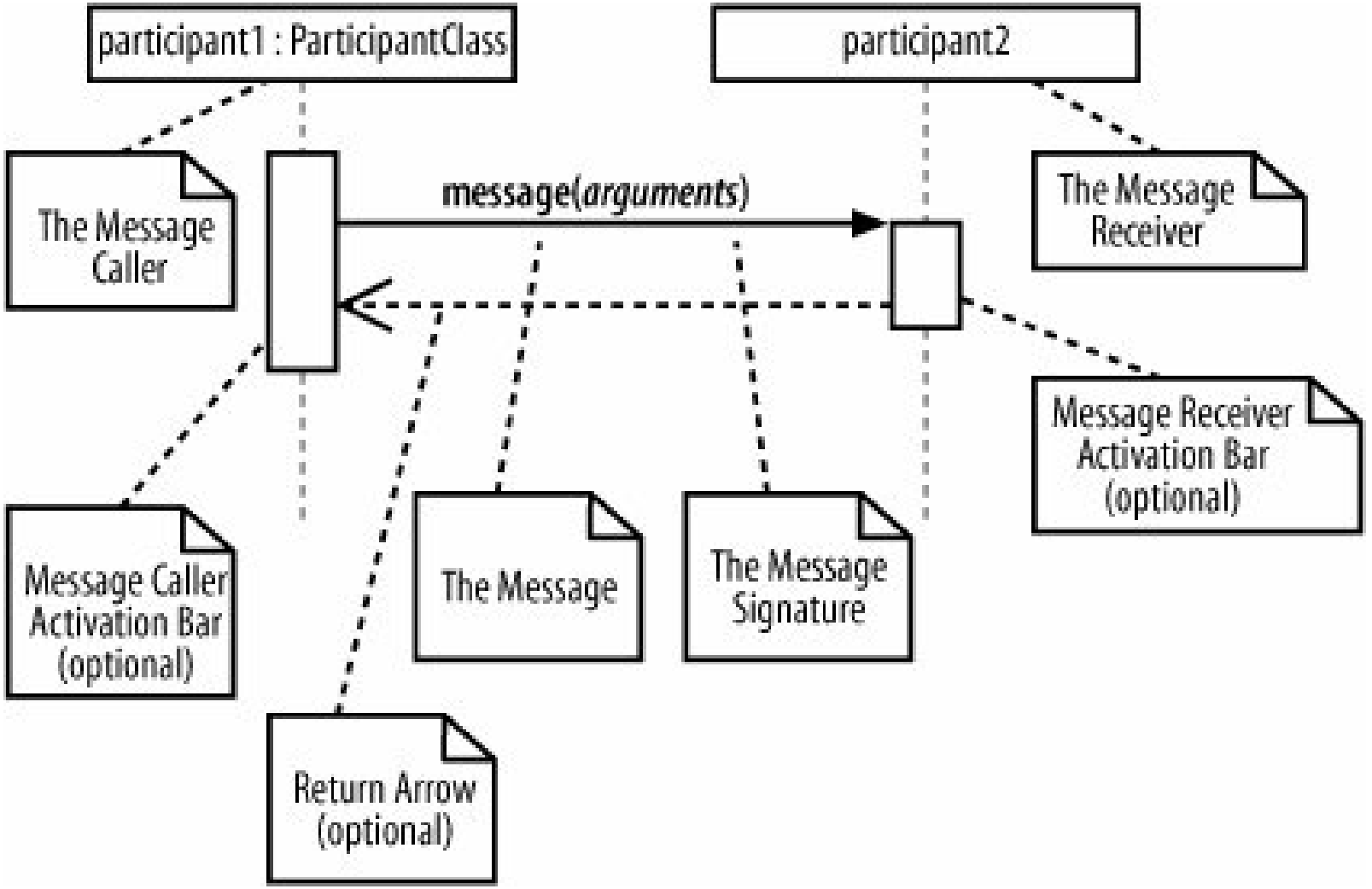
Events, Signals, dan Messages

- Bagian terkecil dari interaksi adalah sebuah ***event***. ***Event*** adalah setiap titik dalam interaksi di mana sesuatu terjadi



Events, Signals, dan Messages

- Event adalah blok bangunan untuk sinyal (Signal) dan pesan (*Message*). Sinyal dan pesan merupakan nama yang berbeda untuk konsep yang sama: sinyal adalah istilah yang sering digunakan oleh desainer sistem, sementara desainer perangkat lunak biasanya lebih memilih pesan. Kita akan gunakan istilah pesan (Message) untuk rancangan kita.
- Sebuah iteraksi pada sequence diagram terjadi jika sebuah participant memutuskan untuk mengirimkan pesan kepada participant lainnya.


participant1 : ParticipantClass
participant2
The Message Caller
message(arguments)
The Message Receiver
Message Receiver Activation Bar (optional)
Message Caller Activation Bar (optional)
The Message
The Message Signature
Return Arrow (optional)

Events, Signals, dan Messages

- Pesan pada sequence diagram digambarkan dengan menggunakan panah dari participant yang ingin meneruskan pesan, MessageCaller, ke participant yang menerima pesan, MessageReceiver . Pesan dapat mengalir kiri ke kanan, kanan ke kiri, atau bahkan kembali ke MessageCaller itu sendiri.

Message Signatures

- Sebuah panah pesan dilengkapi dengan deskripsi, atau **Signatures**. Format untuk **Signatures** pesan:

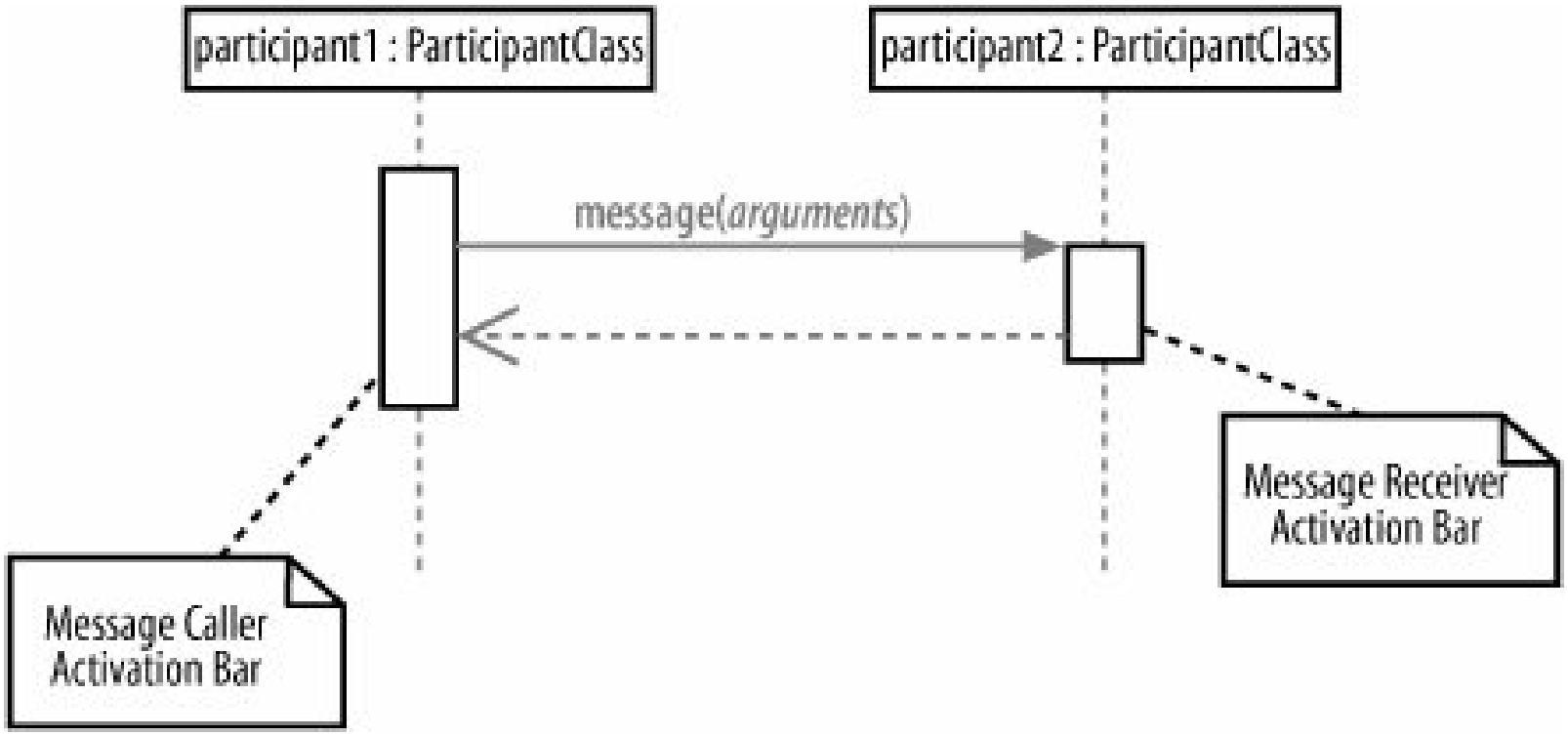
attribute = signal_or_message_name (arguments) : return_type

Message Signatures

Nama Message Signature	Keterangan
doSomething ()	Nama pesan yang doSomething, tetapi tidak ada informasi lebih lanjut yang diketahui tentang hal itu.
doSomething (number1: Nomor, number2: Nomor)	Nama pesan adalah doSomething, dan membutuhkan dua argumen, number1 dan number2, yang keduanya Nomor kelas
doSomething (): ReturnClass	Nama pesan adalah doSomething, dibutuhkan tidak ada argumen dan mengembalikan sebuah objek dari kelas ReturnClass.
myVar = doSomething (): ReturnClass	Nama pesan adalah doSomething, dibutuhkan tidak ada argumen, dan mengembalikan sebuah objek dari ReturnClass kelas yang ditugaskan untuk atribut myVar dari pemanggil pesan

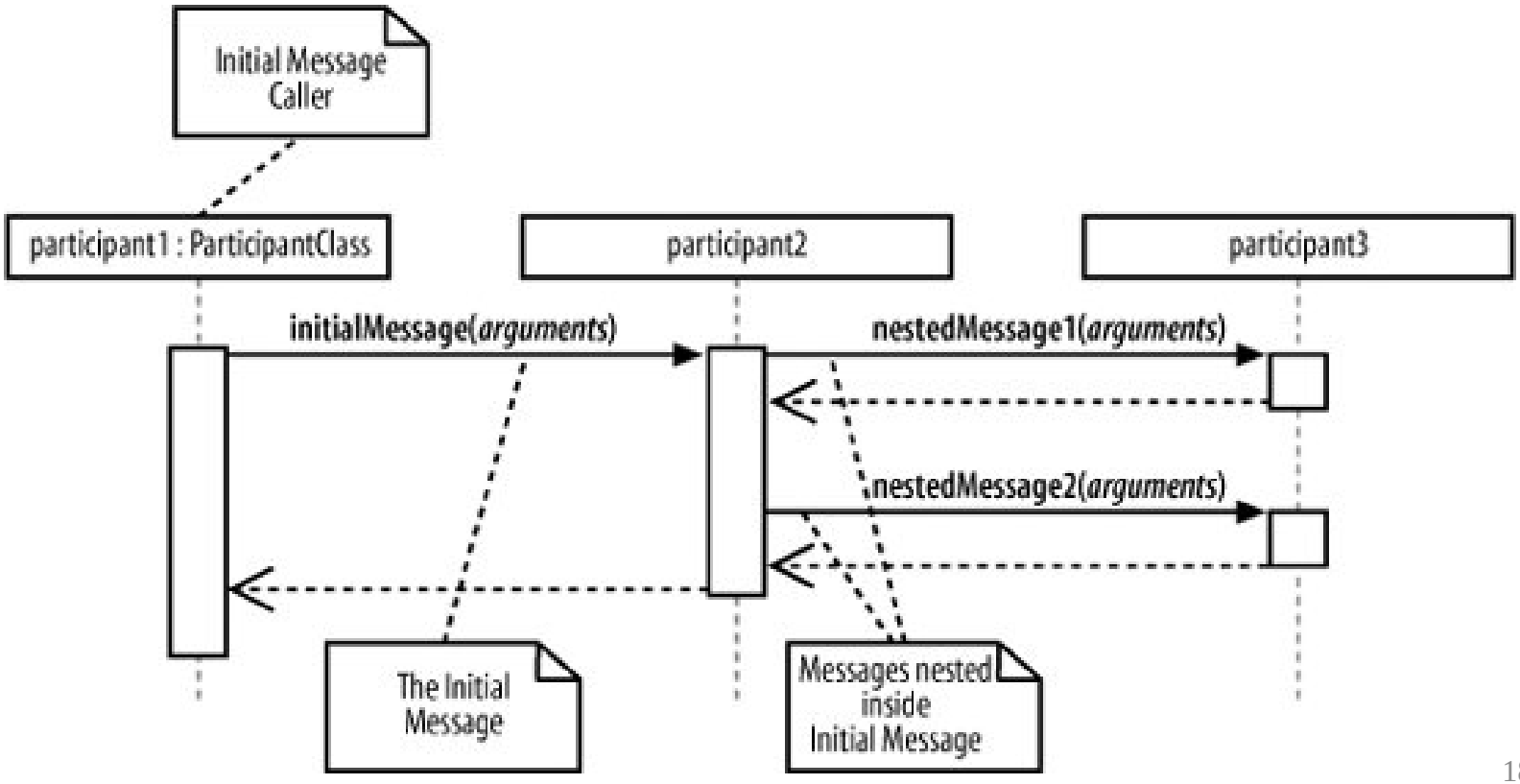
Activation Bars

- Ketika pesan akan diteruskan ke Participan hal ini memicu, atau memanggil, paticipant penerima untuk melakukan sesuatu, pada titik ini, Perticipant penerima dikatakan aktif. Untuk menunjukkan bahwa paticipant penerima, yaitu, melakukan sesuatu, dapat menggunakan bar aktivasi


participant1 : ParticipantClass
participant2 : ParticipantClass
message(arguments)
Message Caller
Activation Bar
Message Receiver
Activation Bar

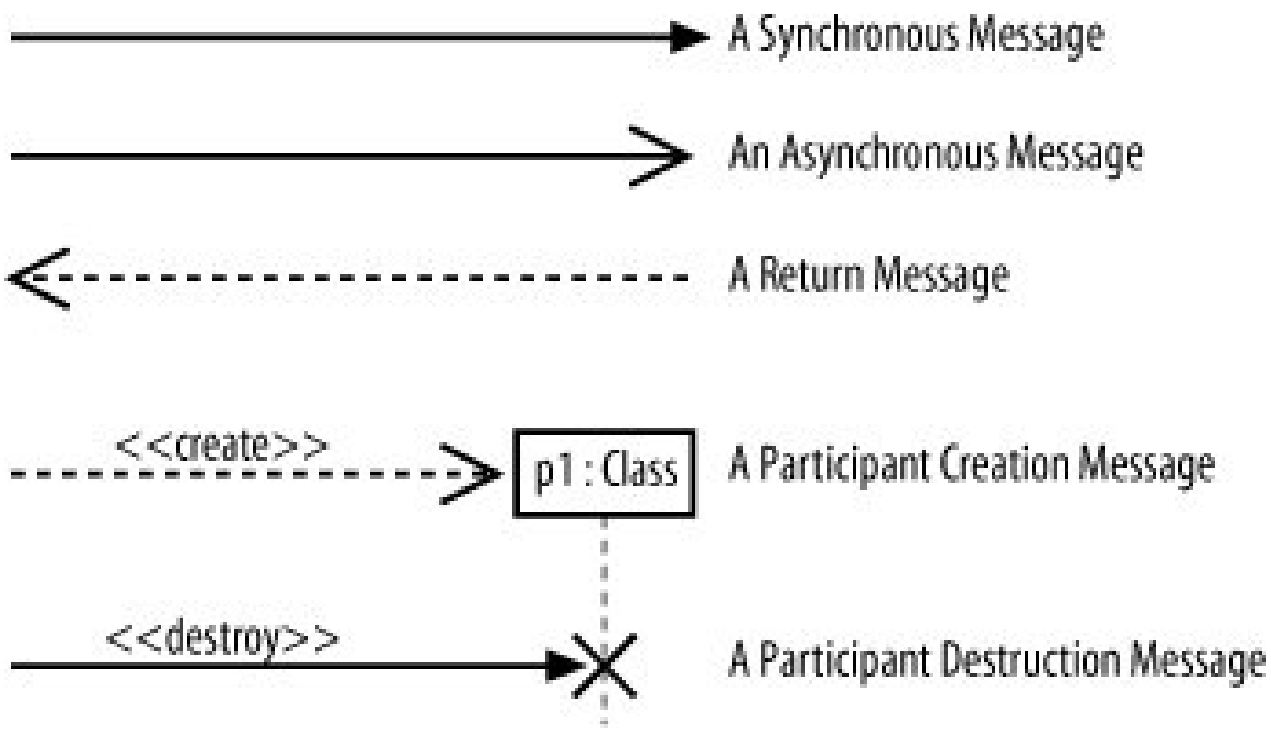
Nested Messages

- Ketika sebuah participant menghasilkan lebih dari satu Message maka ini disebut Nested Message



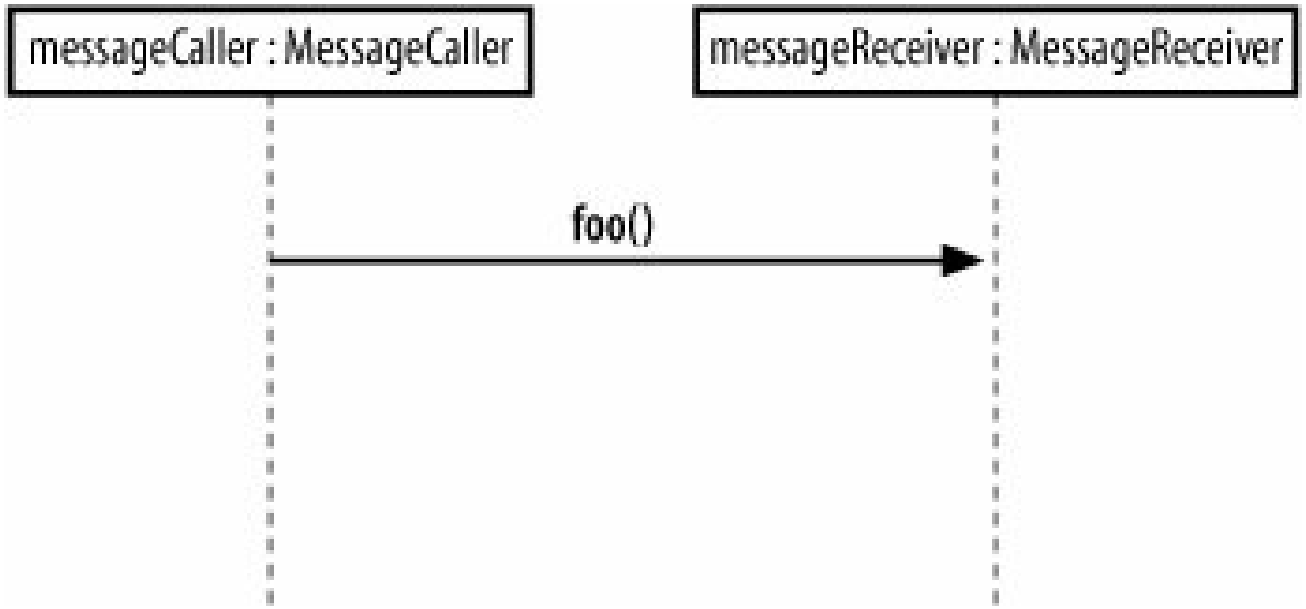
Message Arrows

- Jenis panah yang ada di pesan juga penting untuk memahami apa jenis pesan yang dikirimkan. Sebagai contoh, Message Caller mungkin ingin menunggu pesan returni sebelum melanjutkan dengan pesan Sinkronisasi. Atau mungkin ingin hanya mengirim pesan ke MessageReceiver tanpa menunggu setiap pesan return asynchronous.


A Synchronous Message
An Asynchronous Message
A Return Message
<<create>>
p1 : Class
A Participant Creation Message
<<destroy>>
A Participant Destruction Message

Synchronous Messages

- pesan sinkron dipanggil ketika MessageCaller menunggu untuk Pesan kembali dari MessageReceiver



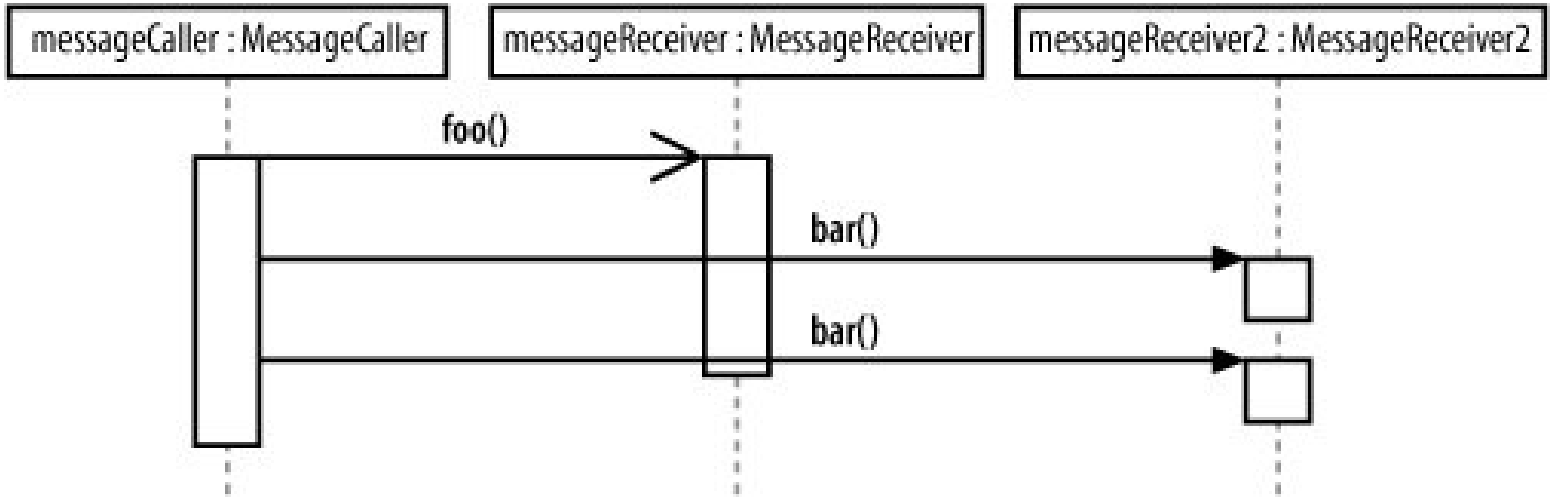
```
public class MessageReceiver
{
   public void foo(  )
   {
      // Do something inside foo.
   }
}

public class MessageCaller
{
   private MessageReceiver messageReceiver;

   // Other Methods and Attributes of the class are declared here

   // The messageRecevier attribute is initialized elsewhere in
   // the class.

   public doSomething(String[] args)
   {
      // The MessageCaller invokes the foo(  ) method

      this.messageReceiver.foo(  ); // then waits for the method to return

      // before carrying on here with the rest of its work
   }
}
```

Asynchronous Messages

- Jika semua interaksi di sistem terjadi satu demi satu dalam urutan sederhana yang bagus. Setiap participant akan menyampaikan pesan ke participant lain dan kemudian menunggu pesan kembali sebelum melanjutkan. Namun kebanyakan sistem tidak bekerja bekerja demikian. Interaksi dapat terjadi pada titik dengan waktu bersamaan, atau kadang-kadang kita akan ingin memulai mengumpulkan semua interaksi pada waktu yang sama dan tidak menunggu pesan kembali.

Asynchronous Messages

- Sebagai contoh, katakanlah kita sedang merancang sebuah software dengan user interface yang mendukung pengeditan dan pencetakan satu set dokumen. Aplikasi menawarkan tombol untuk pengguna mencetak dokumen. Percetakan bisa memakan waktu, sehingga kita ingin menunjukkan bahwa setelah tombol cetak ditekan dan dokumen sedang mencetak, pengguna dapat bekerja dengan hal-hal lain dalam aplikasi. Panah pesan reguler sinkron tidak cukup untuk menunjukkan jenis interaksi. Kita membutuhkan jenis baru panah : panah pesan asynchronous.

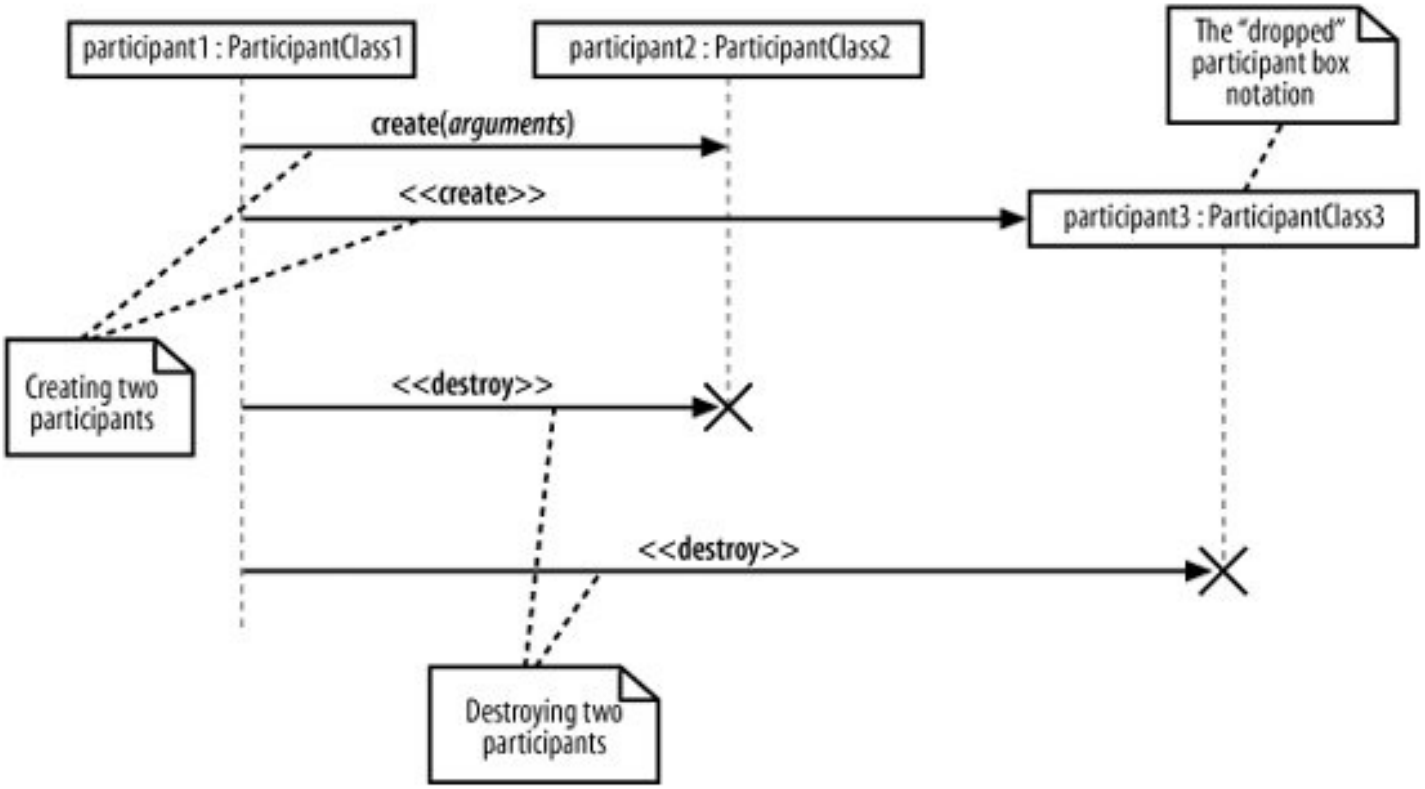

messageCaller : MessageCaller
messageReceiver : MessageReceiver
messageReceiver2 : MessageReceiver2
foo()
bar()
bar()

The Return Message

- Return Message adalah bagian opsional dari notasi yang dapat digunakan pada akhir bar aktivasi untuk menunjukkan bahwa aliran kontrol aktivasi kembali ke participant.

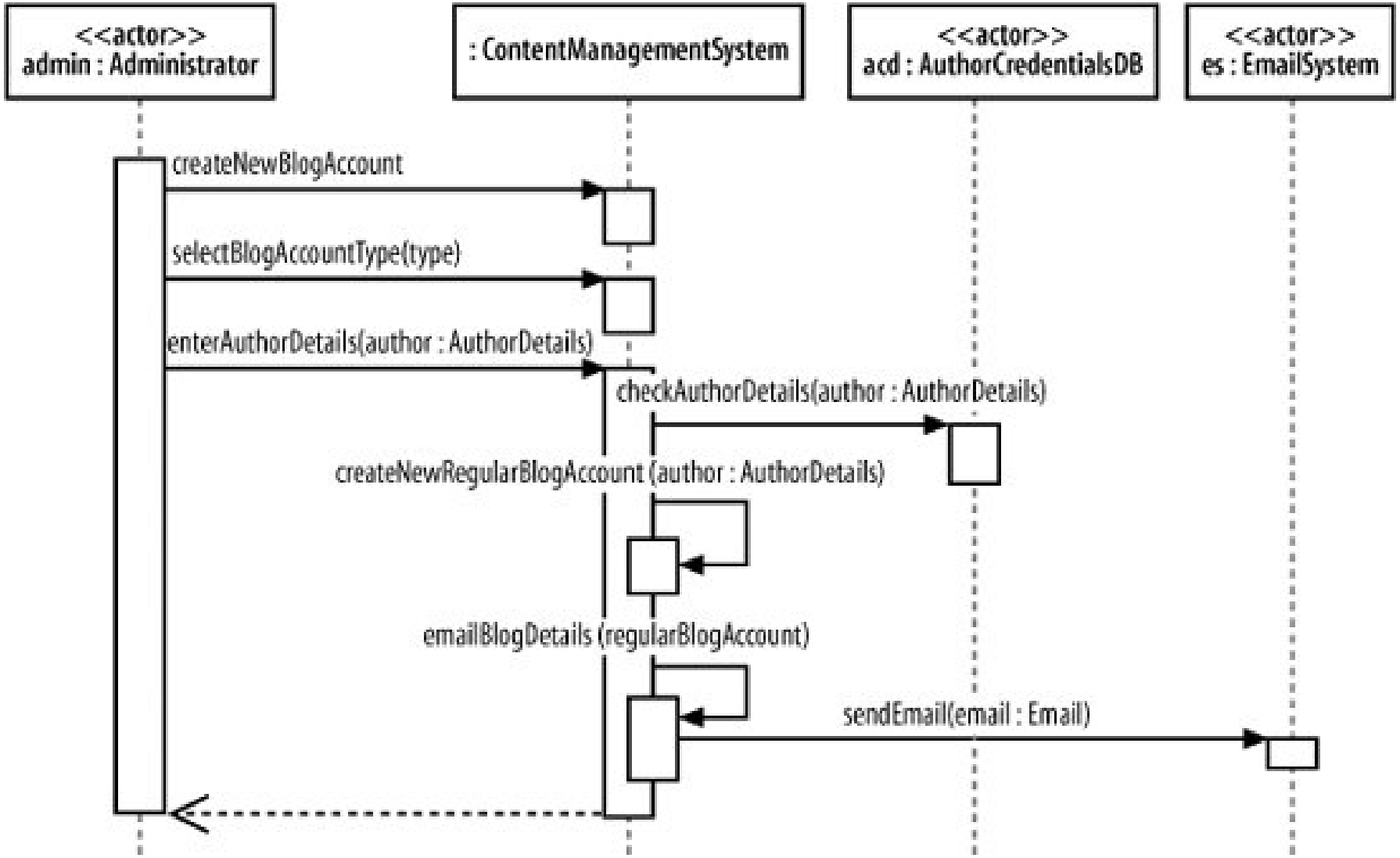
- Kita tidak harus menggunakan Return Message kadang-kadang mereka benar-benar dapat membuat diagram terlalu sibuk dan membingungkan.

Participant Creation and Destruction Messages



Membuat Sequence dari Usecase

Main Flow	Step	Action
	1	The Administrator asks the system to create a new blog account.
	2	The Administrator selects the regular blog account type.
	3	The Administrator enters the author's details.
	4	The author's details are checked using the Author Credentials Database.
	5	The new regular blog account is created.
	6	A summary of the new blog account's details are emailed to the author.


<<actor>>
admin : Administrator
: ContentManagementSystem
<<actor>>
acd : AuthorCredentialsDB
<<actor>>
es : EmailSystem
createNewBlogAccount
selectBlogAccountType(type)
enterAuthorDetails(author : AuthorDetails)
checkAuthorDetails(author : AuthorDetails)
createNewRegularBlogAccount (author : AuthorDetails)
emailBlogDetails (regularBlogAccount)
sendEmail(email : Email)

Referensi

- Bennett, McRobb and Farmer (2005)
- OReilly.Learning.UML.2.0.Apr.2006