# Core Java Pattern List

## **Core Java Design Patterns**

In core java, there are mainly three types of design patterns, which are further divided into their sub-parts:

## **1.Creational Design Pattern**

1. Factory Pattern
2. Abstract Factory Pattern
3. Singleton Pattern
4. Prototype Pattern
5. Builder Pattern.

AD

## **2. Structural Design Pattern**

1. Adapter Pattern
2. Bridge Pattern
3. Composite Pattern
4. Decorator Pattern
5. Facade Pattern
6. Flyweight Pattern
7. Proxy Pattern

## **3. Behavioral Design Pattern**

1. Chain Of Responsibility Pattern
2. Command Pattern
3. Interpreter Pattern
4. Iterator Pattern
5. Mediator Pattern
6. Memento Pattern
7. Observer Pattern
8. State Pattern
9. Strategy Pattern
10. Template Pattern
11. Visitor Pattern

# J2EE Design Patterns

# **JEE or J2EE Design Patterns**

J2EE design patterns are built for the developing the Enterprise Web-based Applications.

In J2EE , there are mainly three types of design patterns, which are further divided into their sub-parts:

### **1. Presentation Layer Design Pattern**

1. [Intercepting Filter Pattern](https://www.javatpoint.com/intercepting-filter-pattern)
2. [Front Controller Pattern](https://www.javatpoint.com/front-controller-pattern)
3. View Helper Pattern
4. Composite View Pattern

### **2. Business Layer Design Pattern**

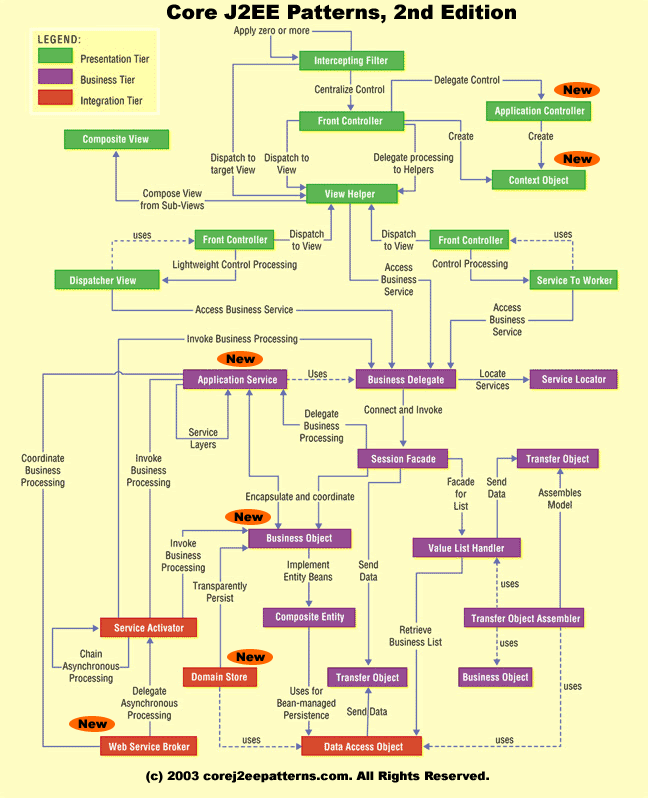
1. Business Delegate Pattern
2. Service Locator Pattern
3. Session Facade Pattern
4. Transfer Object Pattern

### **3. Integration Layer Design Pattern**

1. Data Access Object Pattern
2. Web Service Broker Pattern

The J2EE Patterns that are covered in this article are:

* [MVC Pattern](https://stackabuse.com/java-j2ee-design-patterns/#mvcpattern)
* [Business Delegate Pattern](https://stackabuse.com/java-j2ee-design-patterns/#businessdelegatepattern)
* [Composite Entity Pattern](https://stackabuse.com/java-j2ee-design-patterns/#compositeentitypattern)
* [Data Access Object Pattern](https://stackabuse.com/java-j2ee-design-patterns/#dataaccessobjectpattern)
* [Front Controller Pattern](https://stackabuse.com/java-j2ee-design-patterns/#frontcontrollerpattern)
* [Intercepting Filter Pattern](https://stackabuse.com/java-j2ee-design-patterns/#interceptingfilterpattern)
* [Service Locator Pattern](https://stackabuse.com/java-j2ee-design-patterns/#servicelocatorpattern)
* [Transfer Object Pattern](https://stackabuse.com/java-j2ee-design-patterns/#transferobjectpattern)



|  |
| --- |
| [***Pattern Relationships***](http://www.corej2eepatterns.com/PatternRelationships.htm) |

|  |
| --- |
| **Presentation Tier** |

|  |
| --- |
| [**Intercepting Filter**](http://www.corej2eepatterns.com/InterceptingFilter.htm) |
| [**Context Object**](http://www.corej2eepatterns.com/ContextObject.htm) |
| [**Front Controller**](http://www.corej2eepatterns.com/FrontController.htm) |
| [**Application Controller**](http://www.corej2eepatterns.com/ApplicationController.htm) |
| [**View Helper**](http://www.corej2eepatterns.com/ViewHelper.htm) |
| [**Composite View**](http://www.corej2eepatterns.com/CompositeView.htm) |
| [**Dispatcher View**](http://www.corej2eepatterns.com/DispatcherView.htm) |
| [**Service To Worker**](http://www.corej2eepatterns.com/ServiceToWorker.htm) |

|  |
| --- |
| **Business Tier** |

|  |
| --- |
| [**Business Delegate**](http://www.corej2eepatterns.com/BusinessDelegate.htm) |
| [**Service Locator**](http://www.corej2eepatterns.com/ServiceLocator.htm) |
| [**Session Facade**](http://www.corej2eepatterns.com/SessionFacade.htm) |
| [**Application Service**](http://www.corej2eepatterns.com/ApplicationService.htm) |
| [**Business Object**](http://www.corej2eepatterns.com/BusinessObject.htm) |
| [**Composite Entity**](http://www.corej2eepatterns.com/CompositeEntity.htm) |
| [**Transfer Object**](http://www.corej2eepatterns.com/TransferObject.htm) |
| [**T O Assembler**](http://www.corej2eepatterns.com/TransferObjectAssembler.htm) |
| [**Value List Handler**](http://www.corej2eepatterns.com/ValueListHandler.htm) |

|  |
| --- |
| **Integration Tier** |

|  |
| --- |
| [**Data Access Object**](http://www.corej2eepatterns.com/DataAccessObject.htm) |
| [**Service Activator**](http://www.corej2eepatterns.com/ServiceActivator.htm) |
| [**Domain Store**](http://www.corej2eepatterns.com/DomainStore.htm) |
| [**Web Service Broker**](http://www.corej2eepatterns.com/WebServiceBroker.htm)  List of Core J2EE Design Patterns referred from book [Core J2EE Patterns: Best Practices and Design Strategies (2nd Edition)](https://www.amazon.com/Core-J2EE-Patterns-paperback-Strategies/dp/0133807460).  All the Core J2EE Design Patterns are explained with problem, solutions, class diagram, sequence diagram,components, implementation, applicability, references etc. Presentation Tier  * [Intercepting Filter Design Pattern in Java](http://www.javaguides.net/2018/08/intercepting-filter-design-pattern-in-java.html) * [Front Controller Design Pattern in Java](http://www.javaguides.net/2018/08/front-controller-design-pattern-in-java.html) * [Application Controller Design Pattern in Java](http://www.javaguides.net/2018/08/application-controller-design-pattern-in-java.html) * [View Helper Design Pattern in Java](http://www.javaguides.net/2018/08/view-helper-design-pattern-in-java.html) * [Composite View Design Pattern in Java](http://www.javaguides.net/2018/08/composite-view-design-pattern-in-java.html) * [Context Object Design Pattern in Java](http://www.javaguides.net/2018/08/context-object-design-pattern-in-java.html)  Business Tier  * [Data Transfer Object Design Pattern in Java](http://www.javaguides.net/2018/08/data-transfer-object-design-pattern-in-java.html) * [Service Locator Design Pattern in Java](http://www.javaguides.net/2018/08/service-locator-design-pattern-in-java.html) * [Business Delegate Design Pattern in Java](http://www.javaguides.net/2018/08/business-delegate-design-pattern-in-java.html) * [Converter Design Pattern in Java](http://www.javaguides.net/2018/08/converter-design-pattern-in-java.html) * [Transfer Object Assembler Pattern in Java](http://www.javaguides.net/2018/08/transfer-object-assembler-pattern-in-java.html) * [Value List Handler Pattern in Java](http://www.javaguides.net/2018/08/value-list-handler-pattern-in-java.html)  Integration Tier  * [Data Access Object Pattern in Java](http://www.javaguides.net/2018/08/data-access-object-pattern-in-java.html) * [Service Activator Pattern in Java](http://www.javaguides.net/2018/08/service-activator-pattern-in-java.html) |

# MSA

# EA Design Patterns

# Catalog of Patterns of Enterprise Application Architecture

Last Significant Update: [January 2003](https://martinfowler.com/eaaCatalog/#version-list)

*A short summary of the patterns in*[*Patterns of Enterprise Application Architecture (P of EAA)*](https://martinfowler.com/books/eaa.html)*.*

These pages are a brief overview of each of the patterns in P of EAA. They aren't intended to stand alone, but merely as a quick aide-memoire for those familiar with them, and a handy link if you want to refer to one online. In the future I may add some post-publication comments into the material here, but we'll see how that works out.

Many of these diagrams demonstrate the rather poor GIF output of Visio. The nice diagrams were redrawn for me by David Heinemeier Hansson

**Domain Logic Patterns:**[Transaction Script](https://martinfowler.com/eaaCatalog/transactionScript.html) (110), [Domain Model](https://martinfowler.com/eaaCatalog/domainModel.html) (116), [Table Module](https://martinfowler.com/eaaCatalog/tableModule.html) (125), [Service Layer](https://martinfowler.com/eaaCatalog/serviceLayer.html) (133).

**Data Source Architectural Patterns:** [Table Data Gateway](https://martinfowler.com/eaaCatalog/tableDataGateway.html) (144), [Row Data Gateway](https://martinfowler.com/eaaCatalog/rowDataGateway.html) (152), [Active Record](https://martinfowler.com/eaaCatalog/activeRecord.html) (160), [Data Mapper](https://martinfowler.com/eaaCatalog/dataMapper.html) (165).

**Object-Relational Behavioral Patterns:** [Unit of Work](https://martinfowler.com/eaaCatalog/unitOfWork.html) (184), [Identity Map](https://martinfowler.com/eaaCatalog/identityMap.html) (195), [Lazy Load](https://martinfowler.com/eaaCatalog/lazyLoad.html) (200)

**Object-Relational Structural Patterns:** [Identity Field](https://martinfowler.com/eaaCatalog/identityField.html) (216), [Foreign Key Mapping](https://martinfowler.com/eaaCatalog/foreignKeyMapping.html) (236), [Association Table Mapping](https://martinfowler.com/eaaCatalog/associationTableMapping.html) (248), [Dependent Mapping](https://martinfowler.com/eaaCatalog/dependentMapping.html) (262), [Embedded Value](https://martinfowler.com/eaaCatalog/embeddedValue.html) (268), [Serialized LOB](https://martinfowler.com/eaaCatalog/serializedLOB.html) (272), [Single Table Inheritance](https://martinfowler.com/eaaCatalog/singleTableInheritance.html) (278), [Class Table Inheritance](https://martinfowler.com/eaaCatalog/classTableInheritance.html) (285), [Concrete Table Inheritance](https://martinfowler.com/eaaCatalog/concreteTableInheritance.html) (293), [Inheritance Mappers](https://martinfowler.com/eaaCatalog/inheritanceMappers.html) (302).

**Object-Relational Metadata Mapping Patterns:** [Metadata Mapping](https://martinfowler.com/eaaCatalog/metadataMapping.html) (306), [Query Object](https://martinfowler.com/eaaCatalog/queryObject.html) (316), [Repository](https://martinfowler.com/eaaCatalog/repository.html) (322).

**Web Presentation Patterns:** [Model View Controller](https://martinfowler.com/eaaCatalog/modelViewController.html) (330), [Page Controller](https://martinfowler.com/eaaCatalog/pageController.html) (333), [Front Controller](https://martinfowler.com/eaaCatalog/frontController.html) (344), [Template View](https://martinfowler.com/eaaCatalog/templateView.html) (350), [Transform View](https://martinfowler.com/eaaCatalog/transformView.html) (361), [Two-Step View](https://martinfowler.com/eaaCatalog/twoStepView.html) (365), [Application Controller](https://martinfowler.com/eaaCatalog/applicationController.html) (379).

**Distribution Patterns:** [Remote Facade](https://martinfowler.com/eaaCatalog/remoteFacade.html) (388), [Data Transfer Object](https://martinfowler.com/eaaCatalog/dataTransferObject.html) (401)

**Offline Concurrency Patterns:** [Optimistic Offline Lock](https://martinfowler.com/eaaCatalog/optimisticOfflineLock.html) (416), [Pessimistic Offline Lock](https://martinfowler.com/eaaCatalog/pessimisticOfflineLock.html) (426), [Coarse Grained Lock](https://martinfowler.com/eaaCatalog/coarseGrainedLock.html) (438), [Implicit Lock](https://martinfowler.com/eaaCatalog/implicitLock.html) (449).

**Session State Patterns:** [Client Session State](https://martinfowler.com/eaaCatalog/clientSessionState.html) (456), [Server Session State](https://martinfowler.com/eaaCatalog/serverSessionState.html) (458), [Database Session State](https://martinfowler.com/eaaCatalog/databaseSessionState.html) (462).

**Base Patterns:** [Gateway](https://martinfowler.com/eaaCatalog/gateway.html) (466), [Mapper](https://martinfowler.com/eaaCatalog/mapper.html) (473), [Layer Supertype](https://martinfowler.com/eaaCatalog/layerSupertype.html) (475), [Separated Interface](https://martinfowler.com/eaaCatalog/separatedInterface.html) (476), [Registry](https://martinfowler.com/eaaCatalog/registry.html) (480), [Value Object](https://martinfowler.com/eaaCatalog/valueObject.html) (486), [Money](https://martinfowler.com/eaaCatalog/money.html) (488), [Special Case](https://martinfowler.com/eaaCatalog/specialCase.html) (496), [Plugin](https://martinfowler.com/eaaCatalog/plugin.html) (499), [Service Stub](https://martinfowler.com/eaaCatalog/serviceStub.html) (504), [Record Set](https://martinfowler.com/eaaCatalog/recordSet.html) (508)

**Conversation Patterns**

[HOME](https://www.enterpriseintegrationpatterns.com/index.html) [PATTERNS](https://www.enterpriseintegrationpatterns.com/eaipatterns.html) [RAMBLINGS](https://www.enterpriseintegrationpatterns.com/ramblings.html) [ARTICLES](https://www.enterpriseintegrationpatterns.com/articles.html) [TALKS](https://www.enterpriseintegrationpatterns.com/talks.html) [DOWNLOAD](https://www.enterpriseintegrationpatterns.com/downloads.html) [BOOKS](https://www.enterpriseintegrationpatterns.com/books1.html) [CONTACT](https://www.enterpriseintegrationpatterns.com/gregor.html)

[**Conversation Patterns**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/index.html)

* **Introduction**
  + [Overview](https://www.enterpriseintegrationpatterns.com/patterns/conversation/index.html)
  + [Preface](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Preface.html)
  + [Table of Contents](https://www.enterpriseintegrationpatterns.com/patterns/conversation/toc.html)
  + [Introduction](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Introduction.html)
* [**Describing Conversations**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/DescribingIntro.html)
  + [Choreography](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Choreography.html)
  + [Orchestration](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Orchestration.html)
  + [Hypermedia State](https://www.enterpriseintegrationpatterns.com/patterns/conversation/HyperMedia.html)
  + [Conversation Vocabulary](https://www.enterpriseintegrationpatterns.com/patterns/conversation/VocabularyIntro.html)
* [**Discovery**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/DiscoveryIntro.html)
  + [Dynamic Discovery](https://www.enterpriseintegrationpatterns.com/patterns/conversation/DynamicDiscovery.html)
  + [Advertise Availability](https://www.enterpriseintegrationpatterns.com/patterns/conversation/ParticipantsAdvertise.html)
  + [Consult Directory](https://www.enterpriseintegrationpatterns.com/patterns/conversation/ConsultDirectory.html)
  + [Referral](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Referral.html)
  + [Leader Election](https://www.enterpriseintegrationpatterns.com/patterns/conversation/LeaderElection.html)
* [**Starting a Conversation**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/EstablishingIntro.html)
  + [Three-Way Hand Shake](https://www.enterpriseintegrationpatterns.com/patterns/conversation/ThreeWayHandShake.html)
  + [Acquire Token First](https://www.enterpriseintegrationpatterns.com/patterns/conversation/AcquireToken.html)
  + [Rotate Tokens](https://www.enterpriseintegrationpatterns.com/patterns/conversation/RotateTokens.html)
  + [Verify Identity](https://www.enterpriseintegrationpatterns.com/patterns/conversation/VerifyIdentity.html)
  + [User Grants Access](https://www.enterpriseintegrationpatterns.com/patterns/conversation/UserGrantsAccess.html)
* [**Basic Conversations**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/BasicIntro.html)
  + [Fire-and-Forget](https://www.enterpriseintegrationpatterns.com/patterns/conversation/FireAndForget.html)
  + [Asynchronous Request-Response](https://www.enterpriseintegrationpatterns.com/patterns/conversation/RequestResponse.html)
  + [Request-Response with Retry](https://www.enterpriseintegrationpatterns.com/patterns/conversation/RequestResponseRetry.html)
  + [Polling](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Polling.html)
  + [Subscribe-Notify](https://www.enterpriseintegrationpatterns.com/patterns/conversation/SubscribeNotify.html)
  + [Quick Acknowledgment](https://www.enterpriseintegrationpatterns.com/patterns/conversation/QuickAck.html)
* [**Intermediaries**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/MultiIntro.html)
  + [Proxy](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Proxy.html)
  + [Relay](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Relay.html)
  + [Load Balancer](https://www.enterpriseintegrationpatterns.com/patterns/conversation/LoadBalancer.html)
  + [Scatter-Gather](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Aggregator.html)
* [**Managing Distributed Systems**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/ManagementIntro.html)
  + [Heartbeat](https://www.enterpriseintegrationpatterns.com/patterns/conversation/HeartBeat.html)
* [**Resource Management**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/ResourceIntro.html)
  + [Incremental State Build-up](https://www.enterpriseintegrationpatterns.com/patterns/conversation/IncrementalStateBuild.html)
  + [Lease](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Lease.html)
  + [Renewal Reminder](https://www.enterpriseintegrationpatterns.com/patterns/conversation/RenewalReminder.html)
* [**Ensuring Consistency**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/EnsuringConsistencyIntro.html)
  + [Ignore Error](https://www.enterpriseintegrationpatterns.com/patterns/conversation/IgnoreError.html)
  + [Compensating Action](https://www.enterpriseintegrationpatterns.com/patterns/conversation/CompensatingAction.html)
  + [Tentative Operation](https://www.enterpriseintegrationpatterns.com/patterns/conversation/TryConfirmCancel.html)
  + [Coordinated Agreement](https://www.enterpriseintegrationpatterns.com/patterns/conversation/CoordinatedAgreement.html)
* **Appendices**
  + [Bibliography](https://www.enterpriseintegrationpatterns.com/patterns/conversation/bib.html)

|  |  |  |
| --- | --- | --- |
|  | **Overview**[Conversation Patterns](https://www.enterpriseintegrationpatterns.com/patterns/conversation/index.html) » Introduction | [**Contents**](https://www.enterpriseintegrationpatterns.com/patterns/conversation/toc.html)[Next ⇒](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Preface.html) |

## Conversations between Loosely Coupled Systems

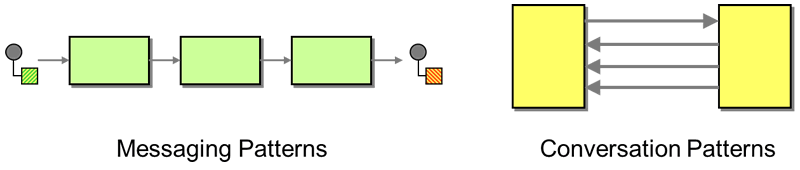
*Work in progress. Last update: Jan 2017*

Asynchronous [*Messaging*](https://www.enterpriseintegrationpatterns.com/patterns/messaging/Messaging.html) is a great way for distributed systems to communicate while maintaining loose coupling. Sending a single message isn't always enough for a meaningful exchange, though: a request expects a response; a certificate exchange may be needed for initial authentication; an order message is supposed to lead to a payment message, and if something goes wrong, additional messages may be needed to handle or correct errors. In these situations, participants engage in *Conversations*, an exchange of related messages. Examples of conversations occur at many levels: traditional web services are orchestrated into larger services, REST services provide referrals or use HATEOAS to trigger subsequent interactions, low-level networking protocols discover services. Conversations also occur in real world: coworkers schedule a suitable meeting time by exchanging their availability, vendors issue offers to be followed by orders, shipments, and payments.

Designing robust conversations is no simple task, though: lost messages, participants' state getting out of sync, timeouts, and lots of other annoying stuff can happen in distributed systems that communicate over unreliable networks. Developers may have a 1000-page TCP/IP tome on the shelf, but are quietly happy that someone else already figured this out for them. Conversations quickly sneak up on developers, though, even when things start out pretty simple: a basic [*Asynchronous Request-Response*](https://www.enterpriseintegrationpatterns.com/patterns/conversation/RequestResponse.html) interaction may foresee re-sending the request if no response is received within a specific time window. Doing so may result in a duplicate response being received if the original response was simply delayed, forcing the requestor to detect and ignore duplicate responses. The requestor likely also shouldn't send indefinite retry messages as this may overburden the system. Quickly the simple sending of a message turned up a number of considerations that govern how multiple messages can be exchanged: we need to decide the rules of the conversation.

## EIP 2: Conversation Patterns

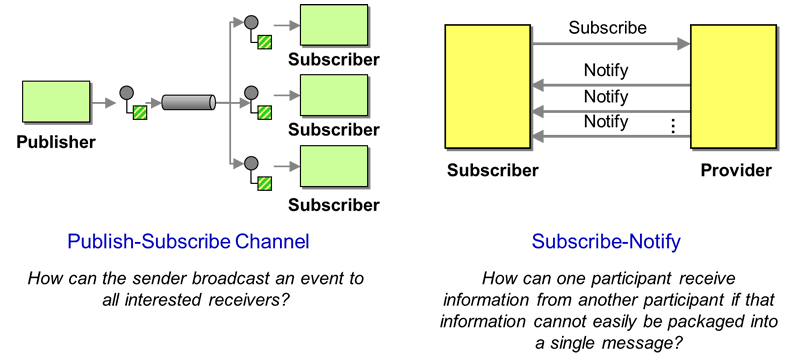
[Conversation Patterns](https://www.enterpriseintegrationpatterns.com/patterns/conversation/toc.html) assist developers in designing robust interchanges between loosely coupled components just like [Messaging Patterns](https://www.enterpriseintegrationpatterns.com/patterns/messaging) did for stateless message exchanges. Just as before, each design pattern represents a "mind-sized" chunk of information that not only describes a concrete solution but also elaborates the trade-offs and forces that motivate it. Building on the idea of the pattern icons, informal sketches capture the essence of the conversation while real-life examples provide a precise protocol specification.

  
*Comparing conversation and messaging patterns*

Conversation Patterns highlight that [Enterprise Integration Patterns](https://www.enterpriseintegrationpatterns.com/) consist of more than just messaging patterns: Conversation Patterns are set to become Volume 2 of the "Enterprise Integration Patterns" book series. Each pattern language's base vocabulary allows tackling different kinds of design problems. Whereas messaging patterns follow the flow of a message through a series of components, conversation patterns take the complementary viewpoint by following the stateful exchange of messages between components over time. The introduction of conversation state in EIP Vol. 2 allows the discussion new topics like error handling or resource management, something that the messaging patterns in EIP Vol. 1 weren't able to do.

## A Comparison: Subscriptions

A simple example highlights the complementary viewpoint that messaging and conversation patterns take.

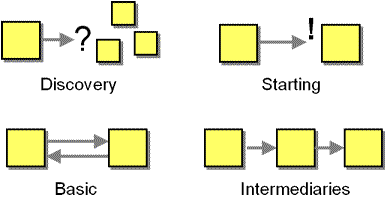
  
*Comparing conversation and messaging patterns*

A [*Publish-Subscribe Channel*](https://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html), shown on the left, is a basic messaging pattern that describes how messages are routed to a set of subscribers. The pattern considers a single message to be distributed across multiple subscribers, discussing how each subscriber receiving its own copy of the message makes adding new subscribers side-effect free. A related, but distinct conversation pattern, takes the opposite viewpoint: it looks at how a single subscriber receives notification messages over time, tacitly assuming that multiple subscribers can do so at the same time. Discussing the flow of messages over time raises important questions, such as how long the flow of messages should continue, which in turn leads to patterns like [*Lease*](https://www.enterpriseintegrationpatterns.com/patterns/conversation/Lease.html) expiration.

## The Pattern Language

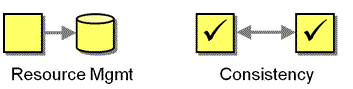
A pattern language structures a collection of patterns and guides the developer towards a meaningful solution (see [[POSA5](http://www.amazon.com/exec/obidos/ASIN/0471486485/enterpriseint-20)]). The Conversation Patterns language distinguishes *foundational* patterns that deal with technical aspects, such as finding a conversation partner, from *application-level* patterns that serve higher-level goals, such as managing resources or negotiating agreements. Both groups of patterns rely on a common vocabulary to [describe and compose conversations](https://www.enterpriseintegrationpatterns.com/patterns/conversation/DescribingIntro.html).

The *foundational* patterns roughly follow the life cycle of setting up a conversation:



* [*Discovery*](https://www.enterpriseintegrationpatterns.com/patterns/conversation/DiscoveryIntro.html): Before services can interact, they have to identify one or more conversation partners.
* [*Starting a Conversation*](https://www.enterpriseintegrationpatterns.com/patterns/conversation/EstablishingIntro.html): Once the conversation partners are known, a conversation has to be established, including authentication, handshaking, and negotiating conversation parameters.
* [*Basic Conversations*](https://www.enterpriseintegrationpatterns.com/patterns/conversation/BasicIntro.html) take place between two participants and a small number of message types. They establish a core vocabulary for the description of more complex conversations.
* [*Intermediaries*](https://www.enterpriseintegrationpatterns.com/patterns/conversation/MultiIntro.html) are simple conversations that have messages flow through a third party.

Based on these foundational conversation patterns, we can tackle common distributed system problems at the *application level*, organized by type of consideration:



* [*Resource Management*](https://www.enterpriseintegrationpatterns.com/patterns/conversation/ResourceIntro.html): Acquiring resources from a service represents a form of coupling: if the requestor disappears unexpectedly, the resource provider continues to hold unneeded resources. Resource management patterns deal with resource allocation and deallocation in loosely coupled systems.
* [*Ensuring Consistency*](https://www.enterpriseintegrationpatterns.com/patterns/conversation/EnsuringConsistencyIntro.html): Things don't always go according to plan in distributed systems where messages can be duplicated and lost, or conversation partners may disappear altogether. These patterns deal with error conditions and help participants reach a consistent state.
* *Reaching Agreement*: Most business situations depend on multiple independent parties reaching an agreement through offers, counter-offers, negotiations and deals.

All patterns are listed in the [Table of Contents](https://www.enterpriseintegrationpatterns.com/patterns/conversation/toc.html).

## Real Life Conversations

Conversations between components often resemble conversations in real life. That's the case because real life is inherently unreliable and asynchronous. Humans have therefore learned to interact by exchanging asynchronous messages, such as leaving a voice mail, mailing a letter, or sending an email. A series of such messages forms a conversation, which serves an overarching purpose, such as negotiating a deal or determining a place for dinner. Real-life conversations can span hours, days, or even months, and have to deal with many of the same issues as distributed systems do, including duplicate messages ("I really need that TPS report"), messages crossing in transit ("Ignore this notice if you've already sent your payment"), and coordinating multiple independent resources ("You got the money? You got the goods?"). Therefore, one can learn a good bit about distributed system design from real life.

sn

# Messaging Patterns