www.ModelingConcepts.com

Do not be afraid to ask!

A Quick Chat about SOMF
Logical Design of Service Relationship

For architects, business analysts, system analysts, software developers, modelers, team leaders, and managers

Service-Oriented Modeling Framework (SOMF) Training Series

Use the service-oriented modeling framework (SOMF) modeling capabilities for enterprise architecture, application architecture, service-oriented architecture (SOA), and Cloud Computing projects.

SOMF is empowered by Sparx Systems Enterprise Architect (EA) modeling platform

About SOMF Logical Design of Service Relationship

The logical design of service relationship is based on message exchange routes between service consumers and service providers. Namely, the message paths established to carry information and execute transactions drive the association between services and their corresponding consumers. Therefore, think about service relationship as a model for delivering and routing data by using messages between a service and a related consumer. Conceptual associations or any other business affiliations between a service and a consumer are important; however here the focus is merely on the technical requirements that drive the design of message routing and delivery.

What is a service?

Before we explain how to associate services and their corresponding consumers, let us define a service according to SOMF. The notion of a service is generalized to a higher abstraction level. Therefore, when you are modeling software, regard a service as any software asset that your organization has been constructing, acquiring, or will be building in the future. This definition of a service may include a variety of software entities such as an application, a Web service, a database trigger, a cloud, a library, an enterprise service bus (ESB), a business process, or a .NET or Java class.

Service Relationship Modeling Assets

The software assets illustrated in a SOMF service relationship diagram are services. These services are categorized in three different structural types, as depicted in Figure 1:

1. *Atomic Service*: an indivisible and fine-grained software asset, typically offers limited processes, interfaces, and capabilities. Example: Customer Information Service that provides high-level and limited account information, such as name, address, and phone number

2. *Composite Service*: a coarse-grained software asset contains internal finer-grained services. Examples: an application that encompasses smaller modules, an ESB that includes internal orchestration and business rules engines, a coarse-grained Web service that offers a large number of trading capabilities, and more

3. *Service Cluster*: a collection of atomic and/or composite services collaborates to provide a number of solutions. Example: An accounting service cluster that offers accounts receivable, accounts payable, a cloud computing service, and payroll modules

Figure 1 also illustrates a consumer, a generalized notion of any software entity that may not only provide services, but also calls other services for data and information.

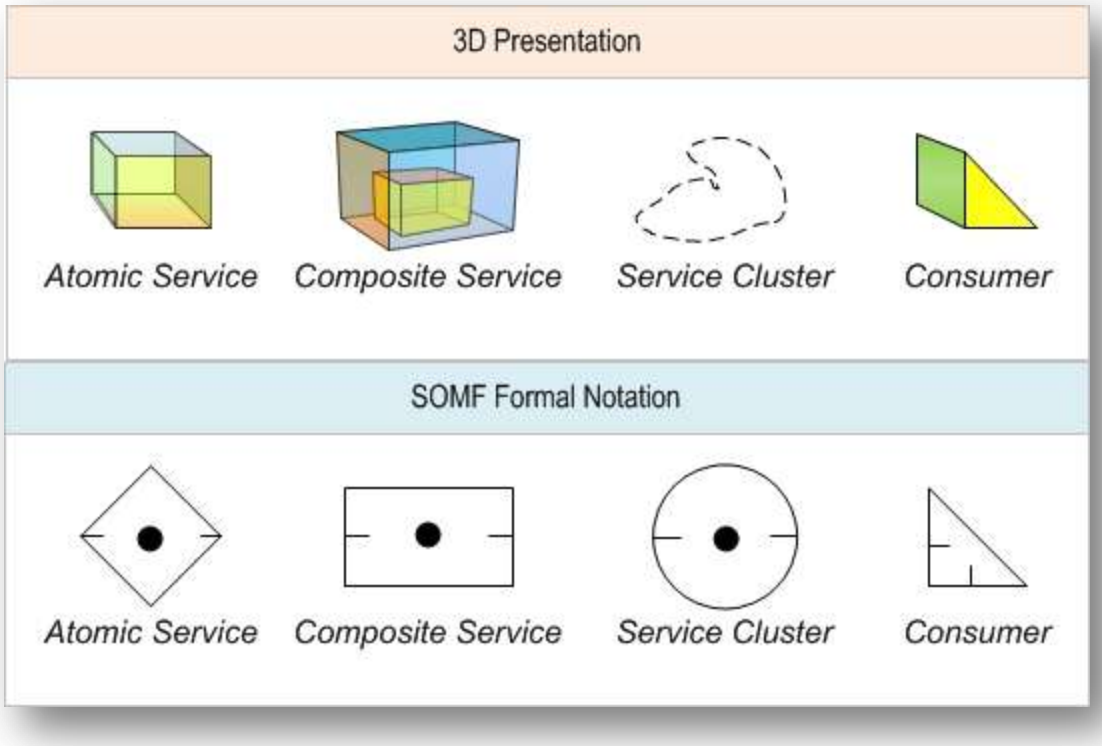

Figure 1: SOMF Logical Design Relationship Modeling Assets

SOMF Logical Design Relationship Notation

Figure 2 illustrates the notation that is used in this tutorial to demonstrate logical relationship between services and corresponding consumers. These symbols are typically used during the service-oriented logical design phase of a project to identify the message routes that must be established between services and their related consumers.

- *Apparent Bidirectional*: this connector depicts a two-way message routing akin to the request/response message pattern. Typically, the consumer invokes a request and the service responses. The term "apparent" signifies the direct link between a service and a consumer, without any interception of a third party software entity.
- *Apparent Unidirectional*: a one-way solicitation or an acknowledgment message routing, during which either the consumer or a service originate a message. A response is not required by the receiving entity. Again, the term "apparent" pertains to a message route that is not intercepted by any other software entity.
- *Implied Bidirectional*: a request/response two-way message routing between a consumer and a service. The term "implied" identifies a message route scenario that is intercepted by a third party broker to deliver the message to its destination.
- *Implied Unidirectional*: a one-way solicitation or acknowledgment message routing. Again, the term "implied" signifies interception of a third party broker to deliver messages.

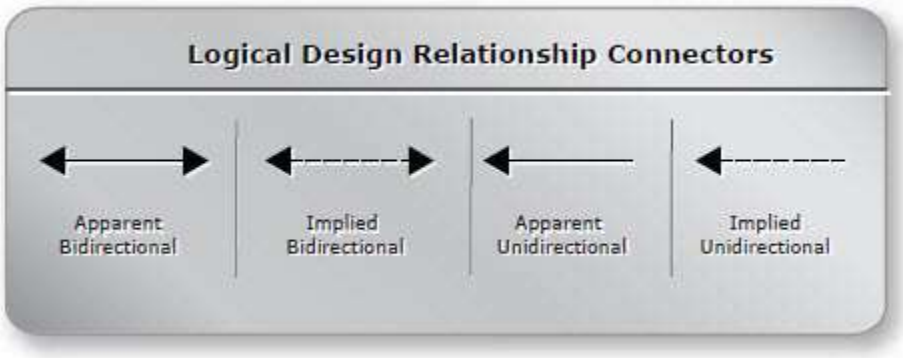

Figure 2: SOMF Logical Design Relationship Notation

Public Design Relationship

Public design relationship identifies a direct affiliation between a consumer and a service provider without the intervention or brokering by a third party software entity. This association implies that messages are exchanged directly between a service and its related consumer. No intermediaries are employed to route or deliver the involved messages.

Public Relationship: Apparent Bidirectional Notation

To depict request/response, a bidirectional relationship between a service and its affiliated consumer, use the Apparent Bidirectional notation as illustrated in Figure 3. This message transmission pattern identifies a two-direction message exchange between the Accounts Payable Atomic Service and the Payroll Composite Service. As shown, there is a direct interaction between the Accounts Payable and the Payroll services. In other words, this message exchange is public: the messages are routed without the interception and interruption of a broker service. Therefore, we name it "Public Relationship".

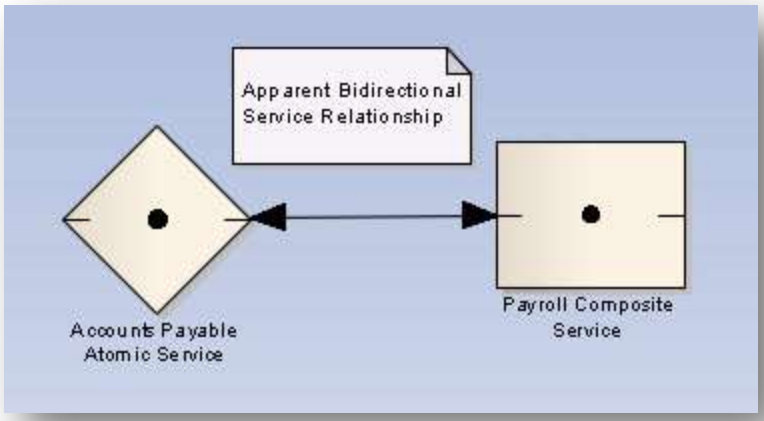

Figure 3: Apparent Bidirectional Message Exchange

Public Relationship: Apparent Unidirectional Notation

To depict a one-way relationship between a service and its affiliated consumer, use the Apparent Unidirectional notation as illustrated in Figure 4. This message transmission pattern identifies a one-direction message exchange between the Savings Account Composite Service and the Account Status Atomic Service.

Here again, the public association between the services is clear: no intercepting broker is defined. However again, the direction of the message exchange is configured as unidirectional, because the message path begins at the Savings Account and reaches its destination Account Status service. No message response is devised.

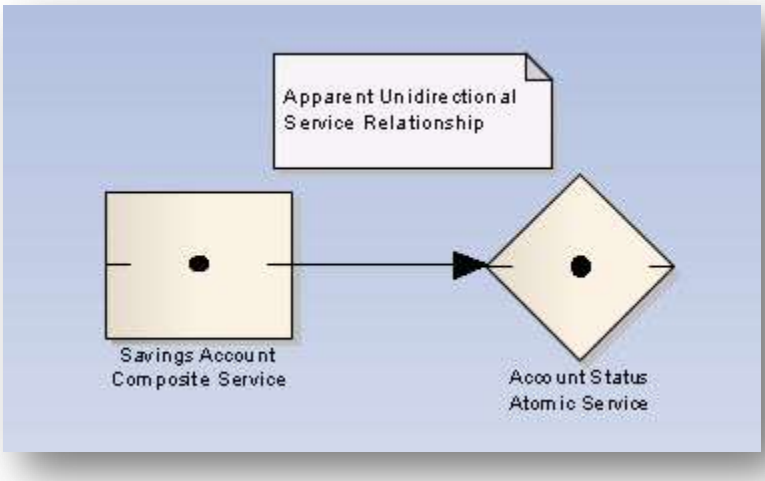

Figure 4: Apparent Unidirectional Message Exchange

Implied Design Relationship

Implied design relationship pertains to a service and consumer association that is not "direct". In other words, an intermediary software entity, a broker is configured to intercept messages that are exchanged between a service and a related consumer because of a variety of design and architecture considerations. This broker may be installed between a consumer and its corresponding service to transform data, encrypt information, or augment the content of a message by adding more data.

Implied Relationship: Implied Bidirectional Notation

The implied bidirectional notation identifies a message route that is merely logical. In essence, the message is transmitted via a broker that delivers the message to the destination entity, and then returns a response. To better understand this idea let us inspect Figure 5. The Trading Account Composite Service exchanges messages with the Checking Account Composite Service via the ESB Composite Service. This illustrates a physical implementation. However, to depict a logical relationship, the implied bidirectional notation is used to signify the "true" relationship between the Trading Account and the Checking Account services.

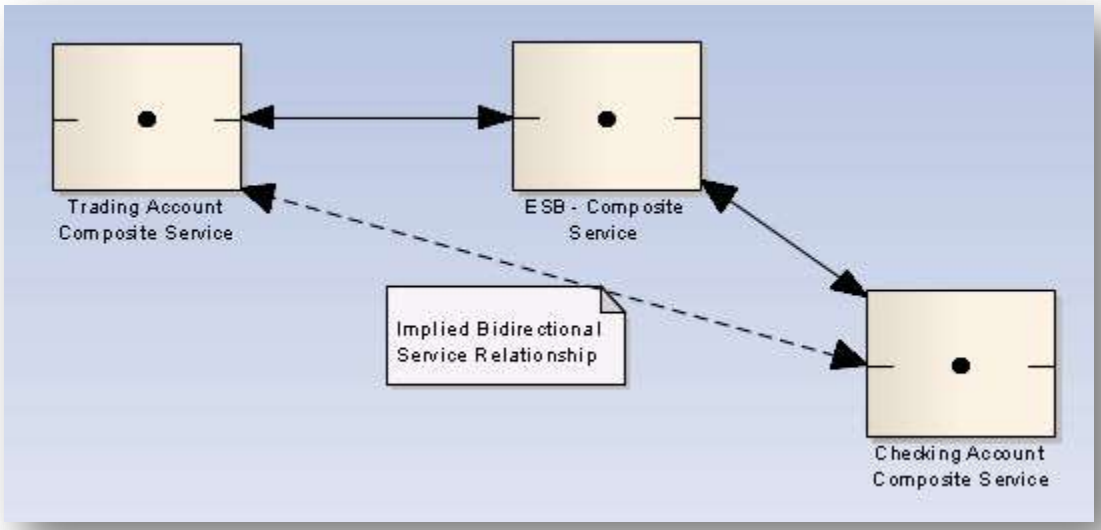

Figure 5: Implied Bidirectional Message Exchange

Implied Relationship: Implied Unidirectional Notation

In the same fashion, use the implied unidirectional to illustrate a one-way relationship between a consumer and a related service. Again, the term "implied" pertains to a logical relationship whose actual physical implementation employs a broker software entity. Figure 6 depicts this idea. Here the Technology Market News Atomic Service is physically linked to the Market News Consumer via the Market News Distributor Composite Service. However, to signify this logical

relationship between the Technology Market New Atomic Service and the Market News Consumer, the implied unidirectional notation is employed.

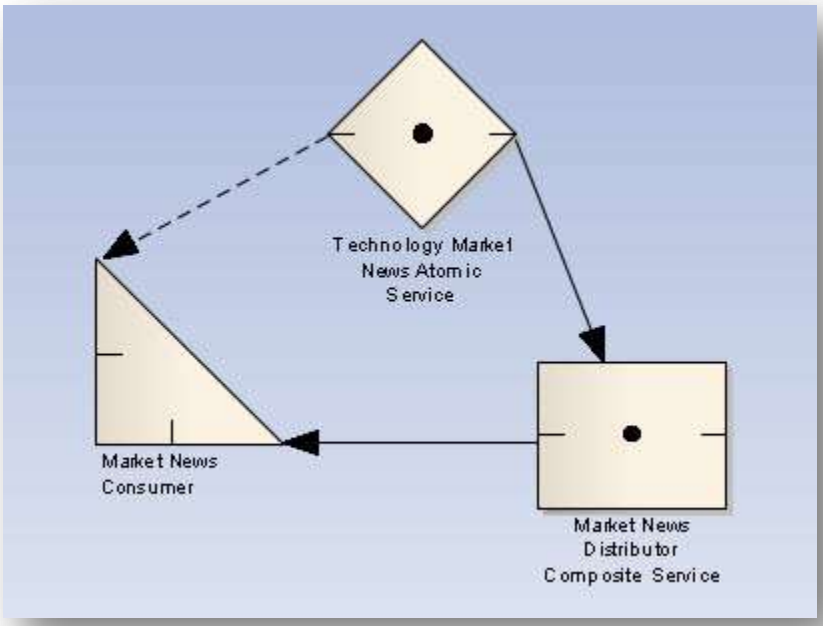

Figure 6: Implied Unidirectional Message Exchange

Isolated Containment Design Relationship

Imagine an instance where a consumer, obviously an external software entity, must exchange messages with an aggregated service, a child service that is contained within a composite service. How should such message routing be designed? The rule of thumb suggests that no direct communication should be established with aggregated or contained services. Instead, use the parent, the containing service to route messages to its internal child service.

Figure 7 illustrates such an aggregated scenario. The Banking Accounts Composite Service contains two finer-grained atomic services: Checking Account and Savings Account. Note that this is an informal representation of a composite structure. Figure 8, however, uses a formal notation to depict an isolated containment design relationship by which the Banking Consumer communicates directly with the Banking Accounts Composite Service that routes two-way messages to its child services: Checking and Savings accounts.

Note that it is also possible to add the implied bidirectional notation to indicate logical relationships that are being established between the Banking Consumer with its corresponding Savings and Checking atomic services.

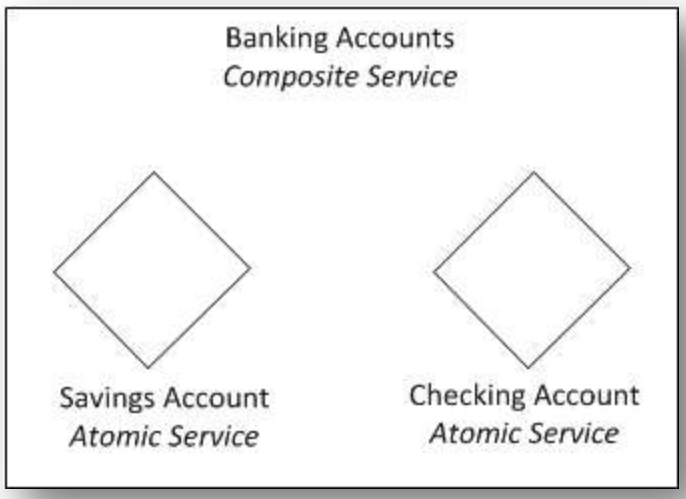

Figure 7: Aggregation Scenario

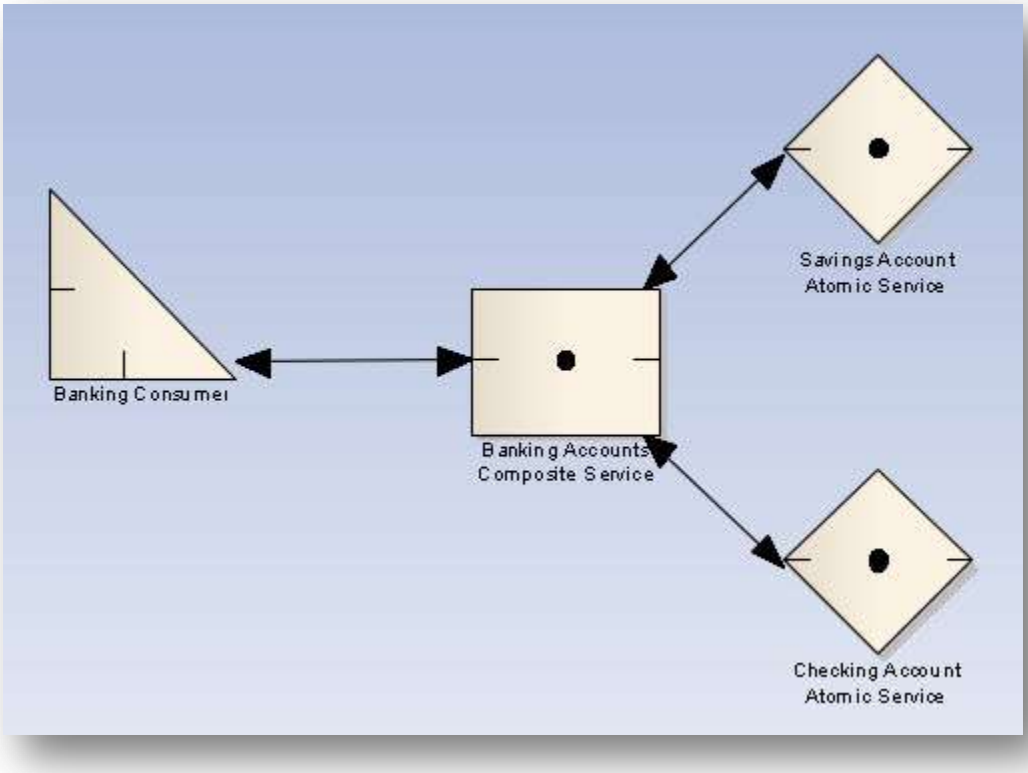

Figure 8: Isolated Containment Design Relationship

Internal Design Relationship

Internal design relationship depicts service associations within an aggregating composite service or service cluster. These relationships can signify bidirectional or unidirectional message exchange directions. Again, remember, this pattern of message routing should only be contained within an aggregating entity.

To demonstrate the internal design relationship scenario, Figure 7 is used again to illustrate a composite service and its two aggregated atomic services. Figure 9, on the other hand, depicts the internal association between these entities, by which two-way apparent bidirectional message exchange pattern is established between the parent Banking Accounts Composite Service and its related child services: Savings Account Atomic Service and Checking Account Atomic Service.

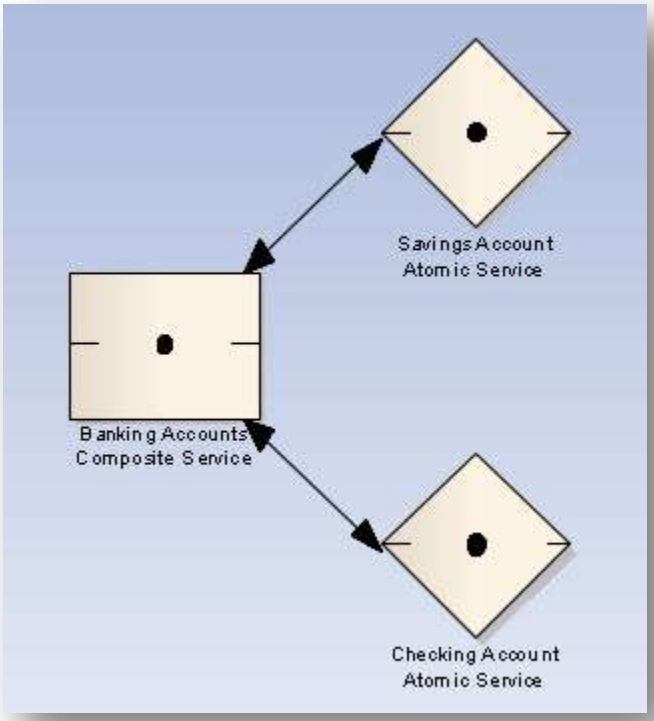

Figure 9: Internal Design Relationship

Relationship Cardinality

The term "cardinality" pertains to the number of links a service may maintain with its peer services or related consumers. These associations, once again, depict message paths that must be established to enable transactions between the message exchange parties. There are four distinct patterns for relationship cardinality:

- One-to-one
- One-to-many
- Many-to-one
- Many-to-many

These cardinality patterns are discussed in the sections that follow.

One-to-one

The one-to-one cardinality pattern identifies message exchange configuration between a consumer and a related service. Figure 10 illustrates a one-to-one scenario in which an Equity Trading Service Cluster exchanges apparent bidirectional messages with its peer Stock Lookup Atomic Service.

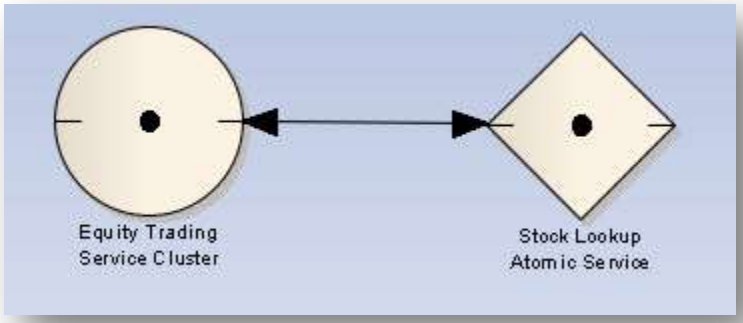

Figure 10: One-to-One Cardinality Pattern

One-to-Many

It is common to find a single service or a consumer that each of which exchanges messages with a number of services or consumers – here a single entity is "using" many entities. Figure 11 illustrates such case. This cardinality pattern depicts a central Business News Atomic Service that distributes news to related news agencies: Real Estate News Atomic Service, Stock Market News Atomic Service, Commodity Market News Atomic Service, and Auto Industry News Atomic Service. Note that the focus here is on a single entity, the Business News Atomic Service. In this case, the message path is apparent unidirectional. However, a bidirectional message route can be established as well, as long as the focus is on a single entity that must communicate with peer entities or partners. Again, the keyword here is "using".

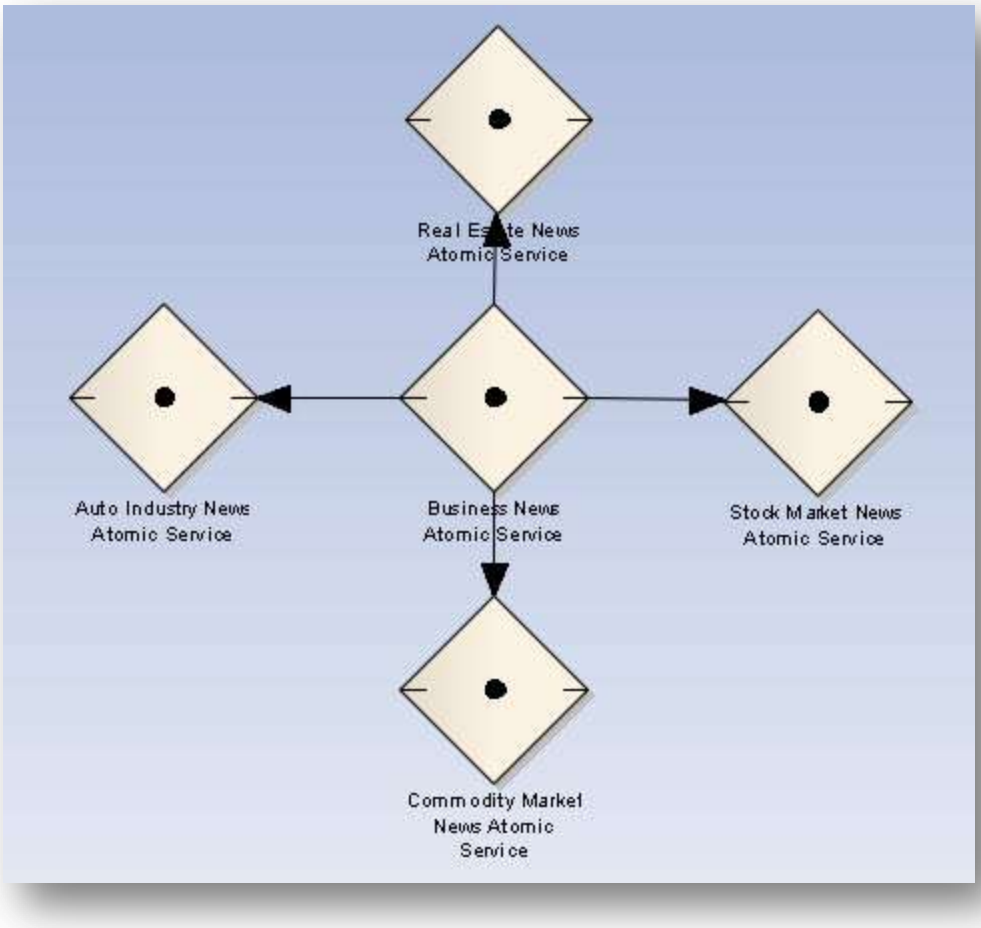

Figure 11: One-to-Many Cardinality Pattern

Many-to-One

The many-to-one cardinality pattern identifies associations of two or more software entities with a single entity – that is, many entities "using" a single entity. The single entity may be one service or one consumer. This pattern is akin to the one-to-many association discussed earlier, however, the focus should be more on the "many" entities that are devised to exchange messages with a single service or consumer. Figure 12 illustrate a many-to-one scenario. Here, three different services are "using" the Employee Lookup Atomic Service: Employee Benefits, Employee Education, and Employee Payroll.

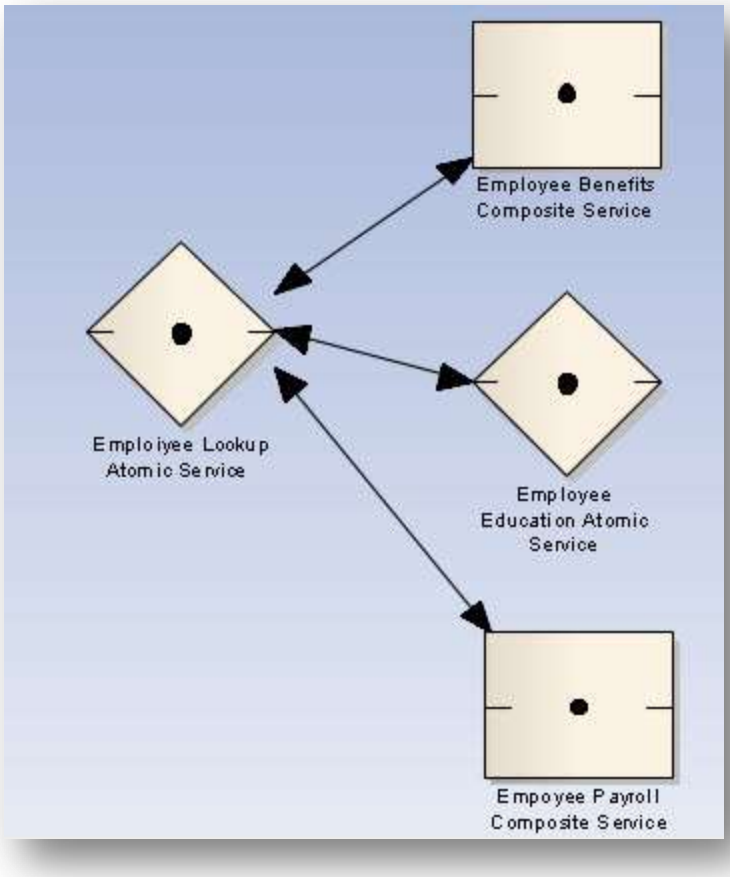

Figure 12: Many-to-One Cardinality Pattern

Many-to-Many

Finally, the many-to-many cardinality pattern depicts a network in which services and consumers exchange messages. In this case, there is no recognized, distinct, or stylize manner by which services communicate with their corresponding consumers. Figure 13 clarifies this pattern. Note that the four insurances: home, car, life, and liability that are represented by services that communicate with each other via a number of message paths. We name this configuration many-to-many since it resembles a network grid.

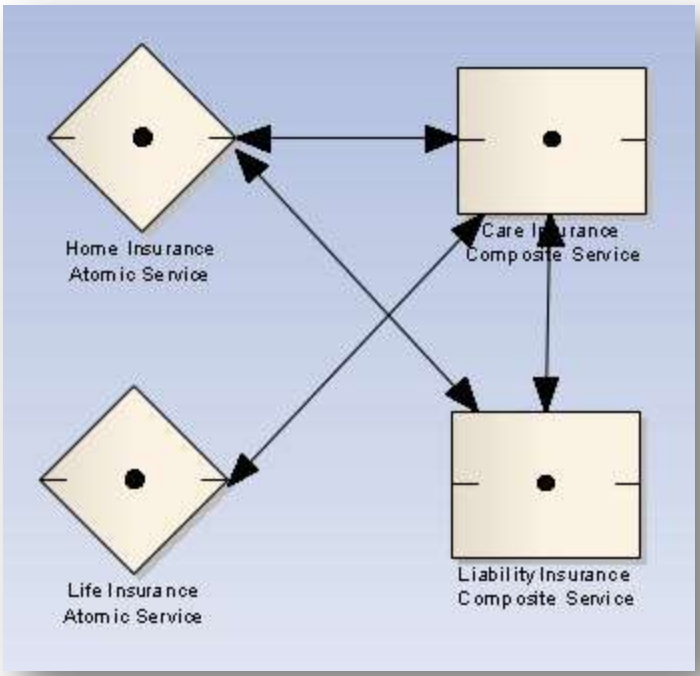

Figure 13: Many-to-Many Cardinality Pattern

Message Synchronization

Message synchronization pertains to the order and the simultaneous aspects of message delivery. In other words, when designing service relationships ask the guiding questions: "Should a service or a consumer exchange messages at the same time with its peers?", "What should be the order of message exchange when designing transactions? ", "Should any order to message delivery be applied?".

The sections that follow identify three major synchronization aspects that you should be aware of when designing service relationship: in-order, same-time, and any-order.

In-Order

The in-order synchronization pattern should be applied when a message exchange requires a certain order to be completed – known as synchronous. This sequential or blocking processing is typically vital to executing proper transactions that take place between consumers and related services. Figure 14 exemplifies the in-order pattern. Note that the Loan Interest Rates Atomic Service exchanges messages in a sequential manner with three services: (1) 15 Year Interest Rates Composite Service, (2) 30 Year Interest Rates Atomic Service, and (3) publishes the interest rates to the final destination – Real Estate Service Cluster.

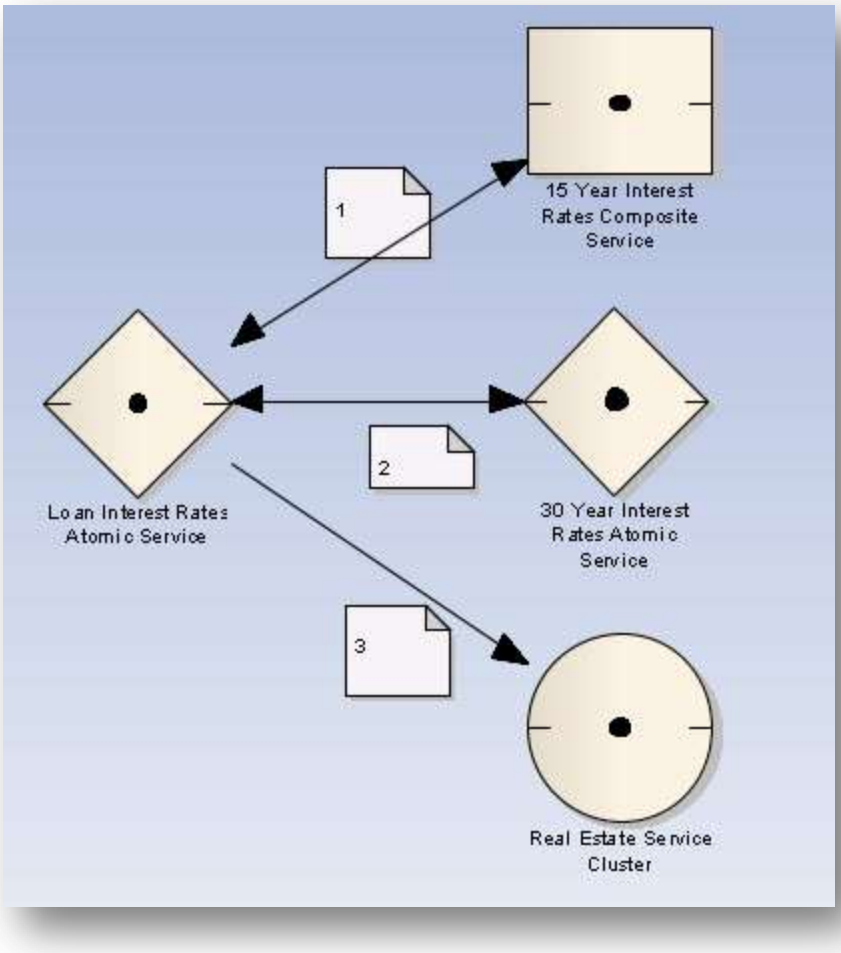

Figure 14: In-Order Synchronization Pattern

Same-Time

The same-time synchronization should be applied to simultaneously execute a transaction or complete a message exchange session between a consumer and corresponding service. This synchronization pattern implies that message exchanges can be performed asynchronously. Figure 15 illustrates this concept. Note that the Equity Trading Service Cluster delivers a unidirectional message to two services simultaneously: Equity Trading Market News Composite Service and Stock Price Atomic Service.

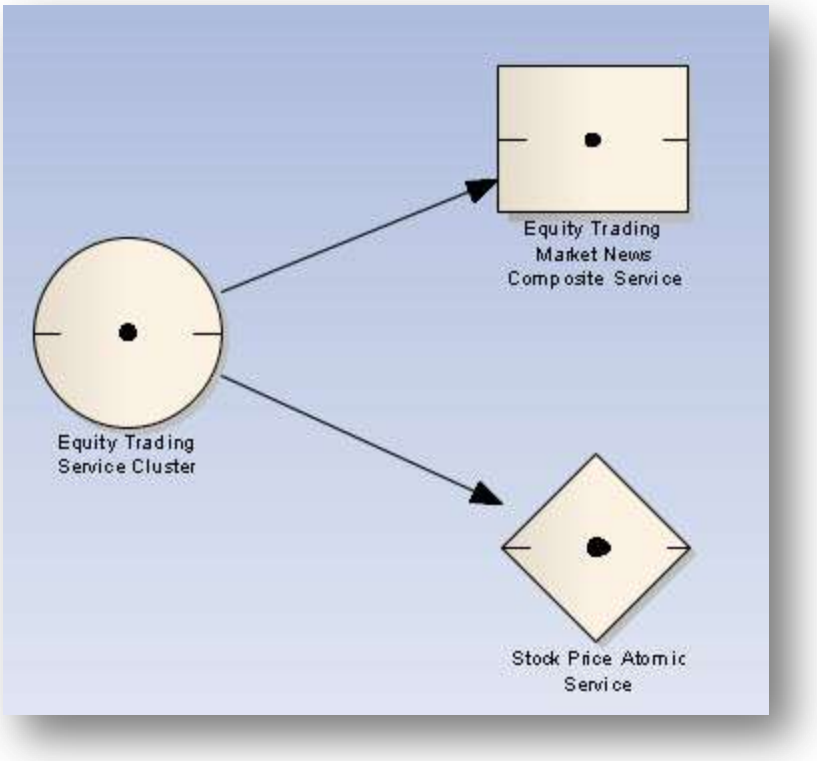

Figure 15: Same-Time Synchronization Pattern

Any-Order

The any-order synchronization pattern pertains to asynchronous or synchronous message exchanges as described in the previous sections.

Tagging Intermediaries

As discussed in the previous sections, an intermediary is a broker, a software entity that is positioned between a consumer and a service to provide mediation activities. These activities can be data or message transformation, applying security, enriching a message, monitoring security, message delivery, and message workload management. This list is not final and the opportunities for message mediation are typically vast.

When designing service relationship and message routes it would be important to tag intermediaries. Tagging means identifying what kind of services a message broker offers. Consider a number of common tags that can be used to identify intermediary contributions:

- T: Message Transformer
- E: Content Enricher

- G: Gateway Enabler
- M: Transaction Monitor
- L: Service Locator
- R: Message Router
- F: Message Filter
- A: Content Aggregator

Figure 16 exmplifies the intermediary tagging idea. As apparent, the ESB Composite Service is tagged with four different offerings that are performed during message exchange sessions: (E) Content Enricher, (M) Transaction Monitor, (L) Service Locator, and (R) Message Router.

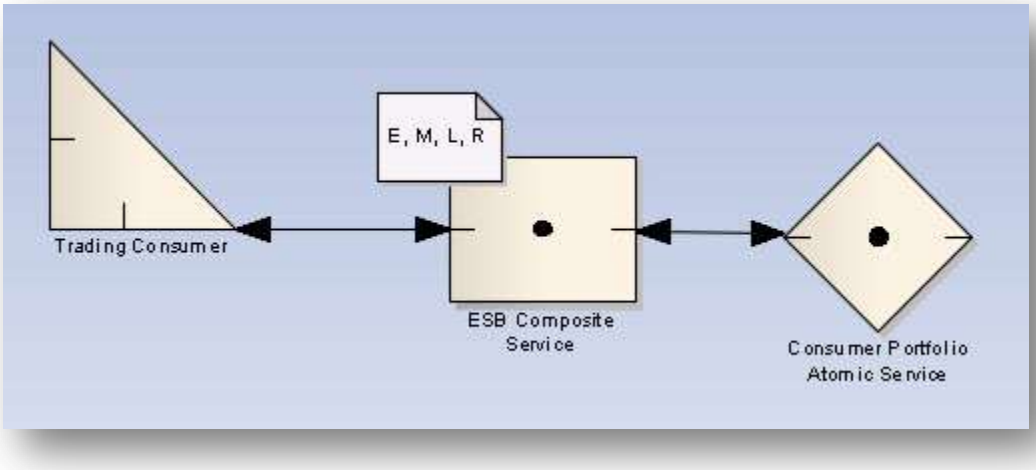

Figure 16: Tagging Intermediaries

SOMF Literature

To learn more about service logical design and service relationship topics refer to these three books on service-oriented modeling: