Chapter 6
Quantifying Integration Architectures

Jan Algermissen

Abstract The products or services offered by enterprises today increasingly depend on information products realized by the corporate IT department. Often the time to market of a product is significantly affected by the time it takes to realize its IT-enabled aspects. In this regard, minimizing realization time within the IT department often becomes the essential factor for bringing a given product to market earlier than the competition.

This chapter proposes a methodology for determining a measure of how the integration styles of given IT systems affect the ability of these systems to adapt to changing requirements.

Introduction

The products or services offered by enterprises today increasingly depend on information products realized by the corporate IT department. Often the time to market of a product is significantly affected by the time it takes to realize its IT-enabled aspects. In this regard, minimizing realization time within the IT department often becomes the essential factor for bringing a given product to market earlier than the competition.

Realization of new functionality to be delivered by an existing IT system usually involves changing system components that are already deployed and interacting with each other to support current business processes. The amount of time and resources required to change these existing components is to a large extend determined by the amount of coupling between them and whether the kind and amount of coupling matches the given integration situation. For example, tight

J. Algermissen (✉)
NORD Software Consulting, Kriemhildstrasse 7, 22559 Hamburg, Germany
e-mail: algermissen@acm.org

E. Wilde and C. Pautasso (eds.), *REST: From Research to Practice*,
DOI 10.1007/978-1-4419-8303-9_6, © Springer Science+Business Media, LLC 2011

coupling between a provider and one single consumer is not a critical situation whereas tight coupling between a provider and a million consumers on the World Wide Web is a changeability disaster.

Given this situation, it appears useful to provide a measure for the overall integration architecture quality of an IT system that expresses how well its component connections match the circumstances of their use. Such a measure would provide a way to assess an IT department's overall resistance to evolution. It could be used to compare integration architectures or could be tracked over time to verify the success of an integration architecture management strategy.

This chapter proposes a methodology to define such a quality measure in terms of the coupling created by the applied connectors on the one hand and in terms of the specific circumstances of their use on the other.

Kinds of Change Impact

When components consume capabilities provided by other components dependency relationships are created. Changes applied to providers potentially affect the consumers and therefore have to be considered during implementation design and deployment planning.

As the basis for a detailed change impact analysis the following sections provide a classification of the various ways in which changes to one component can affect other, depending components.

The kinds of change impact are ordered according to increasing perceived complexity and cost.

Terminate or Suspend Application

Changing the functionality provided by a given component requires the deployment of a new software release realizing the associated changes. Deployments result in a temporary unavailability of the changed components. Consumers are affected by provider unavailability if the applied connectors do not provide temporal decoupling between components.

If temporal coupling exists the unavailability of the provider must be coordinated with the owners of the consumers. Depending on their nature it will be required to suspend or terminate the applications that use the affected components. For example, in automated supply chain management it could be necessary to suspend the submission of new orders and to wait for active ordering processes to terminate before deploying a new version of a provider-side component.

Configure, Build, and Deploy Consumer

To augment the capabilities of a provider it can be necessary to change aspects that do not directly affect concerns of the consumer implementation but require a configuration update or upgrading a software library. If the connector used does not protect a consumer from changes of this kind it will be necessary to reconfigure or rebuild and subsequently to re-deploy the consumer component.

Re-deploying the consumer component not only requires termination of the affected applications; consumer-side software release processes and deployment schedules lead to additional complexity for coordinating the change.

Commonly experienced consumer-side configuration- or build impacts are, for example, the need to update a configured provider address or to upgrade a database driver library.

Data Format Change

New requirements often lead to an evolution of the formats used to transfer data between communicating components. If the applied connectors neither make it possible to negotiate between new and previous data-format versions nor enable the transparent use of transforming intermediaries consumers must also be changed to be able to process the new data format.

In this case, data format changes require implementation activities on the consumer side which lead to additional coordination cost between consumer and provider owners beyond the cost of application termination and component re-deployment.

The complexity of data format changes ranges from simple (for example, adding a field to an address data structure) to highly complicated (for example, changing the data model of a data warehouse, impacting the SQL statements of hundreds business reports).

Connector Protocol Change

Connectors that are realized as a set of procedures often have an associated set of assumptions about the order in which the procedures are called or require relations between parameter values. Such rules are effectively protocols for the interaction and the communicating components must share an understanding of these protocols (Shaw and Garlan 1996).

When the provider changes the rules for the interaction (for example, adds a method that must be called before a given other method) the consumers must also be changed to account for the altered protocol.

Like changes to the data format Connector Protocol Changes result in a requirement to change the consumer implementation unless the connector used prevents protocol changes from affecting the consumer.

(REST's hypermedia constraint (Fielding 2000, pp. 100) is an example of how protocol coupling can be removed from the connector interface).

Shared Identity Change

Components use identifiers to refer to other components and the data and control elements they expose. Some architectural styles remove the need for identity from the connector interface [for example, Event-Based Integration (Fielding 2000 p. 55)] others provide dynamic identifier resolution (for example, DNS) or runtime identifier discovery (for example, hypermedia) to reduce the coupling created by shared identify.

In the absence of such architectural strategies, changes to one component can lead to a requirement to update the shared identity maintained in other components. This can result in component configuration changes or programming activity if identity issues are interweaved with source code.

Common examples of Shared Identity Changes are adjusting database connection settings, changing relational table- or column names, or updating resource identifiers of HTTP-based services that expose a fixed set of URIs as part of their service description.

Communication Model Change

Some connectors do not specify how coordination is achieved between the processes of the communicating components or how communication failures are detected and handled. Instead, these aspects of the communication are deferred to the applications using the connector.

Changes to one component can significantly change the communication model and therefore lead to extensive design and implementation changes of other components.

In the case of components that communicate using a shared file, for example, concurrency control might be changed from a file locking based approach to explicit timing assumptions (component A writes at 9 AM and component B reads at 4 PM).

The possibility of communication model changes is typically found in ad-hoc enterprise integration approaches such as communicating through a shared file or database table. It is rarely (if at all) found in "modern", research-based connectors.

Table 6.1 Kinds of change impact

Change impact kind	Description
Terminate or suspend application	Is it possible that a change to one component requires termination or suspension of applications?
Configure, build, and deploy consumer	Is it possible that a change to one component requires other components to be re-deployed?
Data format change	Is it possible that a change to one component requires other components to update their implementation regarding data structure assumptions?
Connector protocol change	Is it possible that a change to one component requires other components to update their implementation to adjust to a protocol change, for example regarding the sequence of operations?
Shared identity change	Is it possible that a change to one component requires another component to change identifiers of data items or the changed component itself?
Communication model change	Is it possible that a change to one component requires other components to change the model in which they interact with the component? For example, to change from synchronous to asynchronous interactions?
Programming language change	Is it possible that a change to one component requires other components to be re-implemented in another programming language?

Programming Language Change

Certain technical realizations of several connectors prescribe the programming language (or the programming language environment) for component implementations. If, for technical or other reasons, the programming language environment of one component changes the other components must be re-implemented in a language of the new environment. The impact of such a change can be extremely broad, potentially making the change itself impossible.

The possibility of a programming language change is not a property of a given connector but rather of individual connector implementation technologies. The methodology proposed in this chapter differentiates between general connector properties and technology specific effects.

Summary of Kinds of Change Impact

Table 6.1 summarizes the kinds of change impact discussed above.

Connectors

The connector (Fielding 2000, pp. 10) used to mediate communication between components determines the amount of coupling between them. I will illustrate the development of the methodology using a set of connectors typically found in enterprise integration scenarios.

Some of the connector names below also are the names of the corresponding architectural styles. However, this chapter focuses on the connectors and the amount of change potentially exposed on connected components.

File Transfer

A File Transfer connector (Hophe and Woolf 2004a) mediates communication between components through one or several files residing in a well-known location. The components exchange data by reading and writing to these files. Process coordination is usually achieved by file naming conventions (for example, adding a suffix like.*done* to already processed files) or the presence or absence of flag files.

Except for the basic file I/O operations provided by the underlying operating system all aspects of the communication must be specified by the application using the connector. This increases coupling because all participating components have to implement the specific coordination semantics.

The notion of File Transfer Connector also applies to scenarios where some or all of the components access the shared file(s) using FTP (a typical solution for integrating with news feed providers).

Shared Database

A Shared Database connector (Hophe and Woolf 2004b) mediates communication through tables managed by a relational database system. The components exchange information by reading and writing to known tables. Similar to the *File Transfer* connector all communication aspects beyond the interaction with the database are deferred to the application using the connector. This results in tight coupling, especially because the component implementations share knowledge about the table and column names, the specific coordination model and potentially specifics of the database product used.

The use of Shared Database Connectors is very common in the financial industry, especially on the basis of data warehouse tables (sometimes leading to operational use of data gathered for analytical purposes) or on the basis of the relational persistence layer of products (instead of using the programming interface provided by the product).

Remote Procedure Call

Remote Procedure Call (Birrell and Nelson 1984) connectors mediate communication by invoking named procedures over a communication network. Remote Procedure Call connectors employ a request/response communication model. Calling a remote procedure transfers control and data from one component to another for the duration of the call. Distributed objects styles also use Remote Procedure Call connectors.

Common implementation technologies are XML-RPC (Winer 1999), Java RMI (Oracle 2009), CORBA (OMG 2008), and DCOM (Microsoft 2007). The early approach to Web services also favored an RPC-style approach (Tilkov 2005).

The primary coupling concern raised regarding Remote Procedure Call connectors is that all components must share an understanding of the specific connector protocol. Other concerns are decreased understandability because application state is distributed among the communicating components and potential performance problems.

Of particular interest in the context of unRESTful uses of REST-based architectures is the realization of Remote Procedure Call connectors by tunneling the remote method invocation through HTTP requests (Algermissen 2010a).

HTTP Type I

HTTP Type I connectors (Algermissen 2010b) constitute the most common form of architecturally incorrect use of HTTP. The server component typically specifies a set of URIs that the client has to know to communicate with the server. In addition to the set of URIs, the returned and accepted message formats are specified as well as the service specific status codes to expect. Typically, this information is provided in a human-(HTML) or machine-readable form [WADL (Hadley 2009)].

While the HTTP methods are used according to their specification there exists tight coupling between provider and consumer around the service aspects described in the service description document. The amount of coupling is comparable to the amount of coupling in the case of Remote Procedure Call connectors.

A prominent example of a service exposing this kind of connector is the Twitter API (Twitter 2009).

Message Oriented Middleware

Message Oriented Middleware connectors enable components to communicate through the exchange of asynchronous messages. Usually Message Oriented Middleware implementation technologies provide the property of reliable message delivery and thus temporal decoupling between components.

Message Oriented Middleware constrains the component interface to a single method with the semantic of "processThis" (Baker 2005).

Examples of Message Oriented Middleware connectors are Java Message Service (Sun 2008), Microsoft MQ (Microsoft 2009), Document-Oriented Web Services (Baker 2005), and the Simple Mail Transfer Protocol (Klensin 2001).

Event-Based Integration

In the Event-based Integration style (Fielding 2000, p. 55), components communicate through event publishing. Providers announce or broadcast (usually typed) events and the event system will invoke consumer components that have registered for certain types of events.

A key property of event-based integration is decreased coupling because the communicating components do not need to be aware of each other's identity. In addition, event-based integration provides temporal decoupling if the underlying messaging technology provides reliable messaging.

Common implementation technologies for Event-Based Integration connectors include Java Message Service (Sun 2008), WS-Eventing (Box et al. 2006), XMPP Publish-Subscribe (Millard et al. 1999), PubSubHubbub (Fitzpatrick and Slatkin 2010).

HTTP Type II

HTTP Type II (Algermissen 2010c) connectors are another common misuse of HTTP. HTTP Type II is an improvement compared to HTTP Type I because it exposes less service specific information and thereby reduces coupling. HTTP Type II uses only HTTP standard status codes and specific media types.

Nevertheless, similar to HTTP Type I, the URI space, the accepted and returned media types as well as the predefined set of possible response status codes are specified by a design time artifact (for example, a WADL document).

The possible change impacts are reduced because HTTP Type II connectors do not expose specific data formats and can take advantage of HTTP redirection and to some extend content negotiation.

An example of a service that exposes this kind of connector is the Google Calendar API (Google 2010).

REST

The REST architectural style (Fielding 2000) emphasizes overall system simplicity and decoupling of components. Components communicate through the exchange of

self-describing messages in a request–response interaction style. Uniform connector semantics, self-describing messages and the runtime-only exposure of control structures (Fielding 2000, pp. 100) remove all coupling between individual clients and servers.

HTTP 1.1 (Fielding et al. 1999) is the implementing technology of REST.

Change Impact Potential of the Connectors

In the case of most connectors, changes to provider components have an impact on their consumers. However, neither imposes every connector the same kinds of impact nor requires every kind of impact the same amount of resources to address it. To achieve a realistic comparison between connectors, it is essential to examine which connectors potentially lead to which kinds of impact and whether a given impact is very likely to occur or is just an architectural possibility.

The following table shows for every examined connector the possible change impacts. The symbol *o* indicates that it is architecturally possible that the given change impact occurs; that it is not possible to entirely protect consumers from this impact and still apply any kind of desired change to the provider. The symbol + indicates that the given change is relatively common and the symbol ++ indicates that the given change impact is very likely to occur, for example, because it is the usual consequence of the best practices when changing the provider.

The absence of a symbol indicates that a connector is inherently capable of enabling any kind of new provider capability without imposing the given impact on any consumer (Table 6.2).

Table 6.2 Change impact potential of examined connectors

	Terminate or suspend application	Configure, compile, and deploy consumer	Data format change	Connector protocol change	Shared identity change	Communication model change
File transfer	++	++	++	++	++	++
Shared database	++	++	++	++	++	++
Remote procedure call	++	++	+	++	+	
HTTP Type I	++	++	++	+	++	
Message oriented middleware	+	+	+		o	
Event-based integration	+	+	+		o	
HTTP Type II REST	o	o		o	o	

The most problematic connectors are *File Transfer* and *Shared Database*. Both cannot protect the communicating components from any kind of change impact. In addition, all change impacts are equally likely to occur since there are no patterns or best practices that make any of the changes less likely. These connectors are closely related to an "ad-hoc" or "do what works" problem solving style – when new requirements arise, the necessary changes are made without architectural guidance.

Remote Procedure Call and *HTTP Type I* show similar impact potential. *Remote Procedure Call* emphasizes communication through method semantics and change impact tends more towards connector protocol changes. *HTTP Type I* on the other hand emphasizes communication through data format semantics and impact potential tends more towards data format changes.

Message Oriented Middleware and *Event Based Integration* generally have less impact potential due to their ability of temporal decoupling and their focus on uniform method semantics. Both make it less likely that a possible change actually impacts a consumer.

HTTP Type I, *Message Oriented Middleware* and *Event Based Integratio* all focus on data centric communication. However, use of standard document formats, for example, UBL (Bosak and McGrath 2006) seems to be more widely practiced with the latter two than with *HTTP Type I* based designs.

The impact potential of *HTTP Type II* is even lower because the possibility of HTTP redirect- and content negotiation mechanisms makes it much easier to realize new provider functionality without impacting consumers in any way. However, impact is still possible because coupling between provider and consumer exists around the exposed service description.

REST is the only connector that makes provider evolution completely independent from consumer impact. Any kind of evolution can be achieved without causing any impact on consumers. This is achieved by architecturally removing the need for any service specific descriptions. Without such descriptions all coupling between consumer and provider is removed, enabling impact-free evolution.

The Significance of Component Usage

So far I have presented an approach to capture and compare the effect that connectors have on the amount of coupling between components. However, the coupling created by a connector is not sufficient to determine how difficult it is to change the providing component. We also need to consider how many consumers are connected through a given connector and how easy it is to coordinate change activity with them. It is the combination of both, connector coupling and the circumstances of connector use that determine the effect on the consumer.

If the applied connector matches the circumstances of its use the effect on the changeability of the provider will be within reasonable bounds but a connector mismatch can effectively make provider evolution impossible.

For example, tight coupling between a provider and a single consumer that are owned by a single authority is usually not considered an obstacle to evolving the provider component. On the other hand, any kind of coupling between a provider and many consumers that are owned by authorities beyond the provider's control present a situation where changing the provider must be considered impossible.

In the following sections, I will introduce the concepts of *Agency Distance* and *Consumer Cardinality* to denote how easily coordination between component owners can be established, and to express how many consumers are tied to a provider using a given connector.

Agency Distance

Components are owned and operated by human authorities. An agency boundary (Khare and Taylor 2004) denotes the set of components that are owned by a single authority and for which consensus and coordination can be enforced. When a change to a provider component imposes a corresponding change to its consumers the component owners must coordinate their related adaptation activities. The overall cost (and likelihood) of achieving this coordination depends on which kind of connector has been used and also on whether the component owners reside within the same agency boundary or not.

Because of this significance of how far apart the component owners are I use the term *Agency Distance* to capture whether one must cross an agency boundary when connecting components.

The owners of any two communicating components can reside within a single agency boundary or within separate ones. For the latter case, two possibilities exist: the separate agencies can either be rather closely related, for example, as business partners, or almost completely unrelated as are, for example, an online retailer and its potentially millions of customers.

Therefore I differentiate the three agency distances shown in Table 6.3.

Table 6.3 Agency distances

Name	Description	Example
Same	Communicating components are owned by the same authority.	An application connected to its private database, integrated business systems of a single organizational unit.
Near	Communicating components are owned by different agencies, but coordination between the agencies is possible though often undesirable. Especially when integrating with business partners.	Data warehouse integration between subsidiaries and a higher entity, Supply chain management between business partners, e.g. suppliers and manufacturers.
Far	Communicating components are owned by different agencies. Coordination between the agencies is either impossible or highly undesirable.	Online retailer and its customers, a payment gateway offering its service to its customers, a content provider offering its services to subscribers worldwide.

Table 6.4 Consumer cardinalities

Name	Description	Example
0	No consumers at all.	A provider component not yet taken into production or not yet made accessible.
1	Exactly one consumer. No expectation or intention that there will be other consumers.	Two systems connected for synchronization. An application and its "private" database.
n	Very few (e.g. <4), exactly known consumers, no expectation or intention that this number will significantly increase	Inventory system used by a few other business applications.
N	Many, usually unknown, uncontrollable consumers, coordination impossible or very expensive.	Online retailer and its customers, a data warehouse and business systems that use it, a system that supports on-the-road salesmen.

Consumer Cardinality

A dependency relationship exists between provider- and consumer components. The more consumers use a given connector the more difficult it becomes to change the provider because the impact of the change needs to be coordinated with the owners of the consumers. An extremely large number of consumers belonging to many different owners makes it effectively impossible to coordinate a change.

We can differentiate three kinds of consumer cardinalities with regard to how strongly they affect a provider's ability to change (Table 6.4).

Especially in enterprise integration contexts the consumer cardinality n has a strong tendency towards changing to a cardinality of N because it is often the case that it is not exactly known how many system actually depend on a given provider. There are, for example, situations when changing a data warehouse is practically impossible because several hundreds of business reports directly depend on the database schema. Thus, it is important to emphasize that the consumer cardinality of N does not only apply to large scale, public services on the World Wide Web but is frequently found inside enterprise IT systems.

Connector Usage

The two preceding sections introduced the concepts of *Agency Distance* and *Consumer Cardinality* to differentiate several contexts in which connectors can be used. This differentiation is important because it is not the nature of a connector alone that determines how difficult it is to change a component on which other components depend. In fact, it is the balance between the coupling created by a connector on the one hand and the actual circumstances of how it is being used on

Table 6.5 Connector usage

	Same	Near	Far
1	1-same	1-near	–
n	*n*-same	*n*-near	–
N	–	*N*-near	*N*-far

the other that determines how a given exposed connector impacts the changeability of a component.

To classify different contexts of how connectors can be used we combine the notions of Agency Distance and Consumer Cardinality. This combination results in the six Connector Usage patterns shown in Table 6.5.

The combinations 1-far, *n*-far, and *N*-same are not included because they do not exist in reality. For example, by definition you cannot have an unknown, uncontrollable number of consumers within a single agency boundary.

Table 6.6 illustrates the different connector usages with examples.

If the social or business circumstances of an integration scenario make coordination particularly difficult the connector usages *n-near* and *N-near* should be treated as *n-far* and *N-far,* respectively.

Connector Suitability

The goal of the previous sections was to establish the foundation for quantifying the impact of a given "connection" on how easily and quickly the providing component can be changed.

We have examined two aspects of "connections":

1. The potential change impact on the consumer created by the nature of the connector used.
2. The agency distance and consumer cardinality of the actual use of the connector.

In the following I propose an approach to determine a connector suitability value based on cascading sets of rules that compare the potential change impact of a connector with the circumstances of its use and select one value from a set of connector suitability values.

Connector Suitability Values

A connector suitability value expresses how appropriate the choice of the given connector is compared to the context in which it is used. The following table describes the four suitability values (Table 6.7) used in this chapter.

I have chosen a set of only four values because it is hardly possible to exactly determine connector suitability at a finer granularity. I have also chosen not to define

Table 6.6 Example connector usages

Connector usage	Description	Example
1-same	Exactly one consumer. No expectation or intention that there will be other consumers. Provider and consumer reside within the same agency boundary.	Content Management System and its own database. Two systems connected to synchronize a certain kind of asset.
1-near	Exactly one consumer located in a closely related agency boundary, for example, a subsidiary or a special business partner. Communication serves a special need and is not a general offering of one party. There is no intention to provide the connector to other consumers.	Human Resource department providing data, for example, about open positions, to be included on the corporate Web site.
n-same	Very few consumers; all located within the same agency boundary.	Integration between systems that exist for a single common purpose, for example, Content Management.
n-near	Very few consumers located in closely related agency boundaries. Typically between organizational units of a single enterprise or between few special, selected business partners. There might be an intention to reuse the provided services for other contexts, but the goal is not to offer the service as a product.	Systems integration between organizational units inside a single enterprise or between subsidiaries. Systems integration between business partners to help conducting business (not services sold to partners).
N-near	Very many consumers but located in related agency boundaries. Though coordination is possible because consumers tend to be related through contracts, for example, subscriptions it is highly undesirable for practical and social reasons. Best to be proactively treated as N-far.	Supply-chain-management related service, consumed by many business partners. Online payment gateway offered as a service.
N-far	Very many (potentially millions) or consumers residing in highly unrelated administrative domains. Coordination is impossible.	Online retailer and its customers.

an odd number of values in order to avoid a medium value that would act as a meaningless “catch-all”-value for undecided cases. An even number of possibilities enforces explicit reasoning.

If you find that in your context a higher granularity can be meaningfully supported the methodology can be adapted accordingly by changing the rule sets below. However, to enable comparison between systems or to track quality improvements over time the particular value set and the rules you define are required to remain stable.

Table 6.7 Connector suitability values

Name	Symbol	Definition
Perfect	++	The connector matches the circumstances of its use. There is no possibility to improve the evolvability of the providing component by using a different connector.
Reasonable	+	The connector does not harm the evolvability of the component. It is a good match but if resources permit the system would benefit from changing to a perfectly matching connector.
Problematic	−	The connector does not match the circumstances of its use, but there is also no immediate risk that the mismatch might prevent any kind of system evolution. A necessary change can be difficult and costly but it will be possible to achieve. If resources are available and no critical suitability exists the integration quality of the system will benefit from reducing the number of connectors with problematic suitability.
Critical	–	The connector absolutely does not fit the circumstances of its use. The amount of mismatch can make necessary business-level evolution impossible (for example, rolling out a new product). Critical connector suitability conditions should be improved as soon as resources permit.

Assigning Suitability Values

The proposed methodology uses cascading sets of rules to determine the suitability value of a connector in a given context of use. The advantage of an approach that uses subsequent application of rules is that it enables a refinement of the suitability value selection at different conceptual levels.

The value selection at the first level is based on the architectural analysis of the various connectors without considering any particularities of connector implementation technologies.

The second level allows for refinement of the general suitability values to reflect implementation technology specific properties. For example, Java RMI and Web Services, both Remote Procedure Call connectors, have identical general suitability values but differ significantly when taking the technical properties into account. While Java RMI requires all components to be implemented in the same programming language environment Web services provide language independence. Using the former potentially leads to a programming language change impact while the latter does not. Implementation technology suitability rules can reflect these differences by refining suitability values accordingly.

At the third level it is possible to further adjust the suitability values based on specifics of the application domain or to reflect skills and preferences pertaining to the context or department in which the methodology is used.

After subsequent application of the various cascading levels a suitability value is obtained that can be used to compare different components or to analyze a certain component over time.

General Suitability Value Rules for 1-same

The general suitability of a connector for *1-same* connector usage is determined according to the following rules:

- No connector is *critical* because even broad changes to the single one consumer do not lead to a situation that can make business level requirements impossible to realize.
- Connectors that do not prevent against communication model changes are *reasonable* to indicate that there is a possibility of improvement.
- All other connectors are *perfect.*

General Suitability Value Rules for n-same

The general suitability of a connector for *n-same* connector usage is determined according to the following rules:

- No connector is *critical* because within the same agency boundary and a reasonably small number of consumers it will be possible to handle any given change impact. Even if that implies a complete re-implementation of consumers.
- A connector that has the potential impact of changing the communication model is *problematic*. While it is not immediately critical, this situation is a potential problem and a more suitable connector should be used. Especially if the number of consumers grows or if the agency distance changes, for example, in the course of a company merger.
- Connectors that focus on document-oriented communication, aim for temporal decoupling, and have uniform method semantics are a *perfect* match.
- More tightly coupling connectors are *reasonable* but should be replaced. Especially if the number of consumers is expected to grow.

General Suitability Value Rules for 1-near

The general suitability rules for *1-near* connector usage are the same as those for *1-same* connector usage because the fact that there is only a single potentially impacted consumer makes all kinds of changes achievable with reasonable cost and coordination complexity.

General Suitability Value Rules for n-near

The general suitability of a connector for *n-near* connector usage is determined according to the following rules:

- Connectors that have an impact potential of changing the communication model or the connector protocol are *critical*. Change coordination activity not only involves several owners of consumers but also has to be established across agency boundaries. This combination can make the required changes effectively impossible.
- HTTP Type I is a *problematic* (but not critical) connector. Data format and shared identity oriented changes are costly, but still manageable in an *n-near* context.
- *HTTP Type I*, *Message Oriented Middleware* and *Event-Based Integration* can be considered *reasonable* if they are used at all in this context.
- REST and HTTP Type II are *perfect* connectors.

General Suitability Value Rules for N-near

The general suitability of a connector for *N-near* connector usage is determined according to the following rules:

- *REST* and *HTTP Type II* are *perfect* connectors. The change impact potential of an *HTTP Type II* connector can usually be controlled to only lead to changes that can be coordinated with the customers of a service. Google's HTTP-exposed services are a good example of this.
- *Message Oriented Middleware*, *Event-Based Integration*, and *HTTP Type I* are *problematic* (but not critical) connectors. Data format and shared identity oriented changes are costly, but still manageable in an *N-near* context.
- All other connectors are *critical* because their change impact potential is too large for typical B2C or large-scale B2B integration contexts. It is likely either not possible to achieve the necessary coordination or not desirable to ask the *N-near* consumers to consider it. Also, the potentially large number of consumers requires the impact to be easy to document and relatively easy to adapt to. Both can be achieved with *Message Oriented Middleware*, *Event Based Integration*, and *HTTP Type I* but not with *Remote Procedure Call*, *Shared Database* or *File Transfer*.

General Suitability Value Rules for N-far

The general suitability of a connector for *N-far* connector usage is determined according to the following rules:

- *REST* is the *perfect* connector for this connector use.
- All other connectors are *critical* because it is impossible to achieve coordination with consumers in an *N-far* connector usage context.

Table 6.8 Connector suitability

	1-same	*n*-same	1-near	*n*-near	*N*-near	*N*-far
File transfer	+	−	+	–	–	–
Shared database	+	−	+	–	–	–
Remote procedure call	++	+	++	–	–	–
HTTP Type I	++	+	++	+	−	–
Message-oriented middleware	++	++	++	+	−	–
Event-based integration	++	++	++	+	−	–
HTTP Type II	++	++	++	++	++	–
REST	++	++	++	++	++	++

General Connector Suitability Summary

The following table summarizes the connector suitability values determined by applying the general suitability rules stated above (Table 6.8).

Integration in a 1-same context is not problematic. Regarding the overall ability to change the system the specific connector used for such point-to-point integrations does not have much influence. However, this situation immediately changes if the number of consumers grows. In this case, system owners should be advised to change the connector before increasing the number of consumers.

On the contrary connector uses that are either public facing or address large-scale B2B scenarios require *REST* or at least *HTTP-Type II* connectors to avoid obstruction of the evolvability of the system.

Connectors that emphasize the exchange of document-oriented messages (as opposed to procedure calls) and aim for temporal decoupling are suitable even for integration with many, not directly controllable consumers.

The connectors with the best suitability, especially in connector usage contexts found in large enterprise IT scenarios, are *Message Oriented Middleware*, *Event-Based Integration*, *HTTP Type II*, and *REST* connectors. They especially support growing numbers of consumers – a property specifically not shared with Remote Procedure Call connectors.

It is important to note that the suitability values and the conclusions drawn from the analysis refer to system evolvability only. They are not intended to express how well a given connector matches the interaction requirements of the components that form a given application. Performance- or real-time considerations, for example, can mandate the use of Event-Based integration connectors although the suitability rules provided yield a negative suitability value. [On the other hand, it is questionable whether publish/subscribe interactions can be economically sustained in *N-near* or *N-far* scenarios (Fielding 2008)].

Table 6.9 Technology specific suitability of RPC connectors

	1-same	1-near	*n*-same	*n*-near	*N*-near	*N*-far
Remote procedure call	++	+	++	–	–	–
Java RMI	++	+	−	–	–	–
DCOM	++	+	−	–	–	–
CORBA	++	+	++	–	–	–
WS-*(RPC)	++	+	++	–	–	–
RPC-URI tunneling	++	+	–	–	–	–

Additional Rules for Connector Technologies

In addition to the implementation technology, independent rules above further rules can be applied to capture implementation technology properties. The rules presented below are provided as examples and should be extended to cover the properties of the actual technologies used.

Suitability Rules for Remote Procedure Call Connector Technologies

- Some implementation technologies of Remote Procedure Call connectors require the same programming language or programming language environment for the communicating components. These connector technologies are *problematic* when used with more than one consumer because the risk of having to change the implementation of several consumers is undesirable. Examples are RMI and DCOM.
- RPC URI Tunneling is *critical* for any use that involves more than one consumer due to the architectural complexity it involves (for example, the use of idempotent methods to tunnel non-idempotent operations) (Table 6.9).

Suitability Rules for Message Oriented Middleware Connector Technologies

- Some implementation technologies of Message Oriented Middleware connectors require the same programming language or programming language environment for the communicating components. These connector technologies are *problematic* when used with more than one consumer because the risk of having to change the implementation of several consumers is undesirable. Examples are JMS and MSMQ.
- SMTP is a *perfect* connector for large scale messaging (provided the properties of SMTP, for example, its high latency, fit your other architectural needs). The combination of SMTP with other internet standards such as MIME headers and media types are an effective way to address the integration problems resulting from *N-near* and *N-far* usage contexts (Table 6.10).

Table 6.10 Technology specific suitability for MOM connectors

	1-same	1-near	*n*-same	*n*-near	*N*-near	*N*-far
Message-oriented middleware	++	++	++	+	−	–
JMS	++	++	−	−	−	–
MSMQ	++	++	−	−	−	–
WS-* (Doc-Style)	++	++	++	+	−	–
SMTP	++	++	++	++	++	+

Table 6.11 Technology specific suitability for EBI connectors

	1-same	1-near	*n*-same	*n*-near	*N*-near	*N*-far
Message-oriented middleware	++	++	++	+	−	–
JMS	++	++	−	−	−	–
WS-Eventing	++	++	++	+	−	–
XMPP Pub-Sub	++	++	++	+	−	–
PubSubHubbub	++	++	++	++	++	+

Suitability Rules for Event-Based Integration Connector Technologies

- Some implementation technologies of Event-Based Integration connectors require the same programming language or programming language environment for the communicating components. These connector technologies are *problematic* when used with more than one consumer because the risk of having to change the implementation of several consumers is undesirable. An example is JMS.
- The PubSubHubbub implementation of Event-Based Integration connectors combines a publish-subscribe protocol with elements of REST, primarily self-describing messages and uniform interface semantics. It relies on open standards and therefore reduces the potential change impact on consumers. It is a *perfect* match for all usage contexts except for *N-far*. PubSubHubbub constrains notification server behavior in a way that can be problematic in an *N-far* environment (Algermissen 2009) thus it is only **reasonable** for these kinds of uses (Table 6.11).

Additional Rules for Local Suitability Tuning

When the proposed methodology is applied in a specific IT environment it might be desired to further tune the suitability values to reflect domain- or business specific nuances. Such a fine-tuning might, for example, be used to express certain technological capabilities of the associated IT departments or aspects of service level agreements with business partners.

Further sets of rules to be applied subsequent to the general and technology specific suitability rules enable such adjustments if necessary.

Component Change Resistance

Components can expose several different connectors in different connector usage scenarios. For example, a monitoring component might expose an *Event-Based Integration* connector to efficiently distribute monitoring events. At the same time it can also expose a *REST*-based API to enable other components to access monitoring reports or configuration settings (Fielding et al. 2010).

Each of the exposed connectors affects how difficult it is to change the component. Assuming that both connectors in the example are *perfectly* matching their context of use the component would exhibit maximum changeability. Its resistance to change would be minimized.

If, however, the component would also expose a *File Transfer* connector to integrate with several systems in two IT departments of a recently acquired company (an *n-near* connector usage) the component's resistance to being changed would be dramatically higher. Given that the component's overall suitability cannot be better than the worst suitability value of all exposed connectors the resulting component suitability value is the minimum value of all exposed connectors.

In the example case, the additional connector would lower the component's value from *perfect* to *critical* (the value of *File Transfer* for the *n-near* context).

Decreasing component change resistance is a primary goal of integration architecture management because it is a precondition for reducing the amount of time necessary to add new functionality to a system.

Integration Architecture Quality

IT systems exist to support business level processes. Associated use cases are realized as networked applications composed of communicating software components.

Integration architecture management aims to optimize the suitability of the applied connectors to maintain an evolvable system. The key property of interest of an integration architecture is how efficiently the integrated system can be changed to respond to new business requirements.

The notion of *component change resistance* described in the previous section can be used to capture this essential IT system property as a measurable entity. The following table assigns evenly distributed percentage values to the four connector suitability values with 100% indicating a *perfect* match (Table 6.12).

Table 6.12 Perfect-suitability percentage

Suitability value	Match percentage (%)
Perfect	100
Reasonable	66
Problematic	33
Critical	0

Using these percentage values we can express the overall suitability of the connectors exposed by a component as a fraction of the desired optimum of a perfect match (100%). For example, the component mentioned in the previous chapter had an original suitability of 100%. When we added the File Transfer connector to integrate with several near-distance consumers the suitability of the component dropped to 0% (critical).

The mapping of suitability values to percentages makes it possible to calculate the average suitability of a set of components. For example, if we have a system consisting of five components with the individual suitability values of 100%, 66%, 66%, 0%, and 100% the average suitability can be expressed as $332/5 = 66.4\%$. This percentage roughly maps to an average *reasonable* quality of the integration architecture.

The better the individual components score the more adaptable the overall IT system is to requirements for new functionality needed by its stakeholders.

Conclusion

Communication between components causes coupling and this coupling acts as an obstacle against change. This effect needs to be controlled to maintain the ability of an IT system to realize new requirements.

Two forces affect coupling: the nature of the connectors that mediate communication and the number and "distance" of consumers. Choosing connectors that match the circumstances of their use is an essential means to maintain a competitive ability to realize new requirements. Extensive use of mismatched connectors can cause the reactivity of an IT system to stall, leading to critical impact on business evolution.

Most of the problems commonly experienced with integration scenarios are the result of a mismatch between the nature of a given connector and the circumstances of its use.

References

Algermissen J (2009) Message #21263 on atom-syntax mailing list. http://www.imc.org/atom-syntax/mail-archive/msg21263.html. Accessed November 2010

Algermissen J (2010a) Classification of HTTP-based APIs. http://www.nordsc.com/ext/classification_of_http_based_apis.html#uri-rpc. Accessed November 2010

Algermissen J (2010b) Classification of HTTP-based APIs. http://www.nordsc.com/ext/classification_of_http_based_apis.html#http-type-one. Accessed November 2010

Algermissen J (2010c) Classification of HTTP-based APIs. http://www.nordsc.com/ext/classification_of_http_based_apis.html#http-type-two. Accessed November 2010

Baker M (2005) Towards truly document oriented Web services. http://www.coactus.com/blog/2005/07/towards-truly-document-oriented-web-services/. Accessed November 2010

Birrell AD, Nelson BJ (1984) Implementing remote procedure call. *ACM Transactions on Computer Systems*, 2, 1984, pp. 39–59
Bosak J, McGrath T (2006) Universal business language 2.0. OASIS. http://docs.oasis-open.org/ubl/cs-UBL-2.0/UBL-2.0.html. Accessed November 2010
Box D et al. (2006) Web services eventing (WS-Eventing). W3C. http://www.w3.org/Submission/WS-Eventing/. Accessed November 2010
Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine
Fielding RT (2008) Economies of scale. Fielding, R.T. http://roy.gbiv.com/untangled/2008/economies-of-scale. Accessed November 2010
Fielding RT (2010) Fielding, Roy Thomas, Message #15819 on rest-discuss mailing list. http://tech.groups.yahoo.com/group/rest-discuss/message/15819. Accessed November 2010
Fielding RT, Gettys J, Mogul JC, Nielsen HF, Masinter L, Leach P, Berners-Lee T (1999) Hypertext Transfer Protocol – HTTP/1.1. *Internet RFC 2616*
Fitzpatrick B, Slatkin B (2010) PubSubHubbub Core 0.3 – Working Draft. Google Inc. http://code.google.com/apis/pubsubhubbub/. Accessed November 2010
Google (2010) Google calendar API. http://code.google.com/apis/calendar. Accessed November 2010
Hadley M (2009) Web application description language. http://www.w3.org/Submission/wadl. Accessed November 2010
Hophe G, Woolf B (2004a) Enterprise Integration Patterns. Pearson Education. p. 43
Hophe G, Woolf B (2004b) Enterprise Integration Patterns. Pearson Education. p. 47
Khare R, Taylor RN (2004) Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems, in 26th International Conference on Software Engineering (ICSE), (Edinburgh, Scotland, 23–28 May 2004)
Klensin J (2001) Simple mail transfer protocol. http://www.ietf.org/rfc/rfc2821.txt. Accessed November 2010
Microsoft (2007) Distributed component object model. Microsoft. http://msdn.microsoft.com/library/cc201989.aspx. Accessed November 2010
Microsoft (2009) Message queuing MSMQ. Microsoft. http://msdn.microsoft.com/en-us/library/ms711472(VS.85).aspx. Accessed November 2010
Millard P, Saint-Andre P, Meijer R (1999) XEP-0060 Publish-Subscribe. XMPP Standards Foundation. http://xmpp.org/extensions/xep-0060.html. Accessed November 2010
OMG (2008) Common object request broker architecture. Object Management Group (OMG). http://www.corba.org/. Accessed November 2010
Oracle (2009) Java remote method invocation. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html. Accessed November 2010
Shaw M, Garlan D (1996) Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, Englewood Cliffs, NJ, USA. p. 169
Sun (2008) Java Message Service (JMS). http://www.sun.com/software/products/message_queue/index.xml. Accessed November 2010
Tilkov S (2005) RPC style web services. http://www.innoq.com/blog/st/2005/05/18/rpcstyle_web_services.html. Accessed November 2010
Twitter (2009) The Twitter API. http://apiwiki.twitter.com/Twitter-API-Documentation. Accessed November 2010
Winer D (1999) XML remote procedure calls. http://www.xmlrpc.com/spec. Accessed November 2010