# Fundamentals of SOA

- service-oriented
  - it represents a distinct approach for separating concerns
  - i.e., logic required to solve a large problem can be better constructed, carried out, and managed if it is decomposed into a collection of smaller, related pieces
  - Each of these pieces addresses a concern or a specific part of the problem

# Example: A City

- Let's take your average cosmopolitan city.
- It is already full of service-oriented businesses. Individual companies are service-oriented in that each provides a distinct service that can be used by multiple consumers.
- Collectively, these businesses comprise a business community.
- It makes sense for a business community not to be served by a single business outlet providing all services.
- By decomposing the community into specialized, individual outlets, we achieve an environment in which these outlets can be distributed.

- "Service-oriented architecture" is a term that represents a model in which automation logic is decomposed into smaller, distinct units of logic.
  - Collectively, these units comprise a larger piece of business automation logic. Individually, these units can be distributed.
- Even in a distributed business community, if we impose overbearing dependencies, we could inhibit the potential of individual businesses.
  - Although we want to allow outlets to interact and leverage each other's services, we want to avoid a model in which outlets form tight connections that result in constrictive inter-dependencies.
  - By empowering businesses to self-govern their individual services, we allow them to evolve and grow relatively independent from each other.

- service-oriented architecture (SOA) encourages individual units of logic to exist autonomously yet not isolated from each other.
- Units of logic are still required to conform to a set of principles that allow them to evolve independently, while still maintaining a sufficient amount of commonality and standardization.
- Within SOA, these units of logic are known as **services.**

- services encapsulate logic within a distinct context. This context can be specific to a business task, a business entity, or some other logical grouping.
- when building an automation solution consisting of services, each service can encapsulate a task performed by an individual step or a sub-process comprised of a set of steps.
- A service can even encapsulate the entire process logic.
- the larger scope represented by the services may encompass the logic encapsulated by other services.

For services to use the logic they encapsulate they can participate in the execution of business activities.
To do so, they must form distinct relationships with those that want to use them.

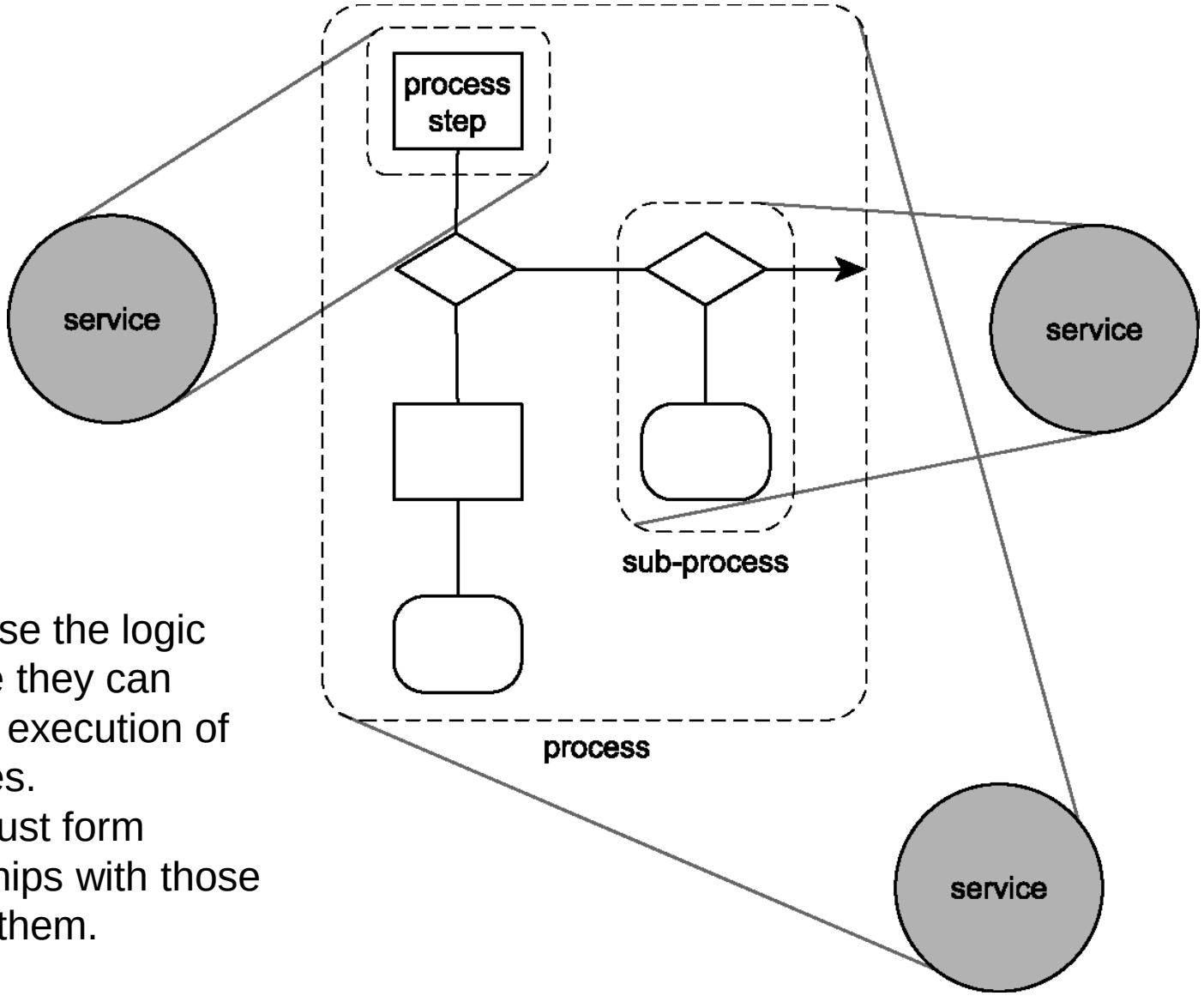


**Figure 3.1:** Services can encapsulate varying amounts of logic.

Within SOA, services can be used by other services or other programs. Regardless, the relationship between services is based on an understanding that for services to interact, they must be aware of each other. This awareness is achieved through the use of service descriptions.

A service description in its most basic format establishes the name of the service and the data expected and returned by the service. The manner in which services use service descriptions results in a relationship classified as loosely coupled.

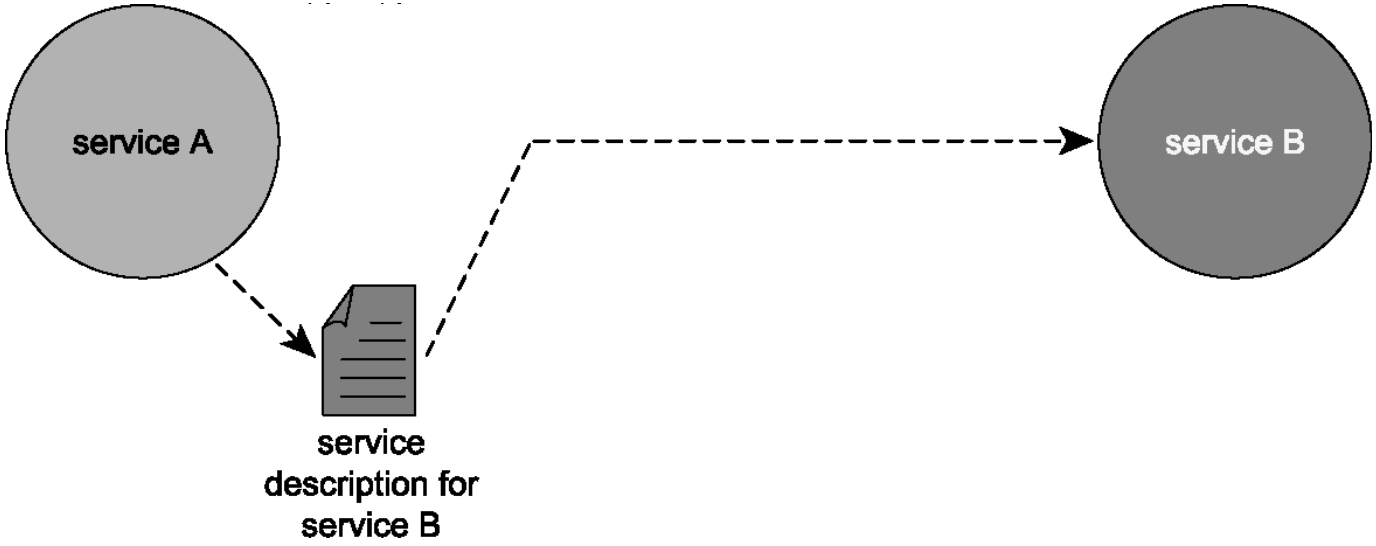


**Figure 3.2:** Because it has access to service B's service description, service A has all of the information it needs to communicate with service B.

For services to interact and accomplish something meaningful, they must exchange information. A communications framework capable of preserving their loosely coupled relationship is therefore required. One such framework is messaging.

After a service sends a message on its way, it loses control of what happens to the message thereafter. That is why we require messages to exist as "independent units of communication."
This means that messages, like services, should be autonomous.

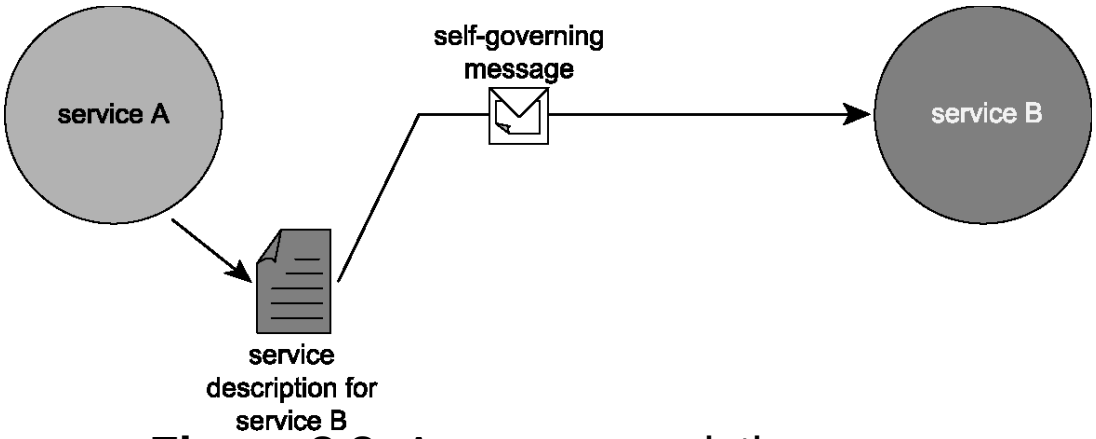


**Figure 3.3:** A message existing as an independent unit of communication.

Much like object-orientation, service-orientation has become a distinct design approach which introduces commonly accepted principles that govern the positioning and design of our architectural components

The application of service-orientation principles to processing logic results in standardized service-oriented processing logic. When a solution is comprised of units of service-oriented processing logic, it becomes what we refer to as a service-oriented solution.

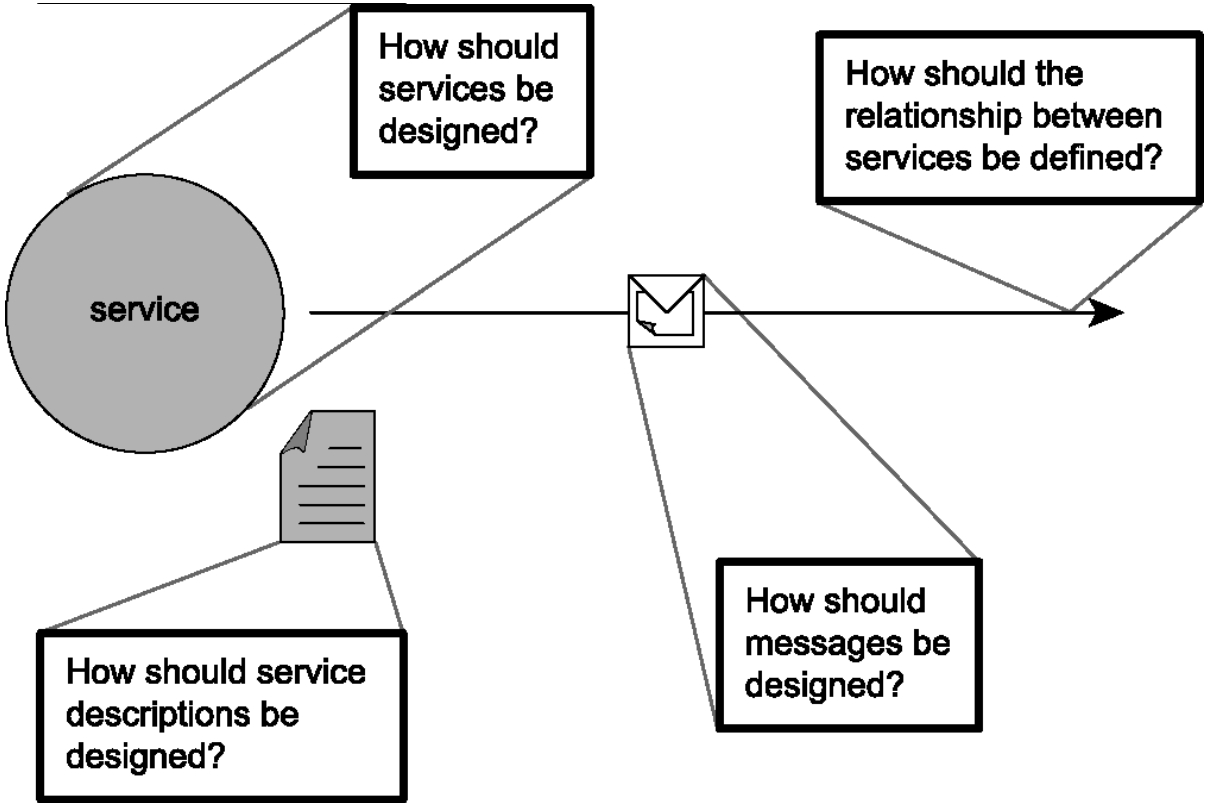


**Figure 3.4:** Service-orientation principles address design issues.

- Loose coupling
  - Services maintain a relationship that minimizes dependencies and only requires that they retain an awareness of each other.
- Service contract
  - Services adhere to a communications agreement, as defined collectively by one or more service descriptions and related documents.
- Autonomy
  - Services have control over the logic they encapsulate.
- Abstraction
  - Beyond what is described in the service contract, services hide logic from the outside world.
- Reusability
  - Logic is divided into services with the intention of promoting reuse.
- Composability
  - Collections of services can be coordinated and assembled to form composite services.
- Statelessness
  - Services minimize retaining information specific to an activity.
- Discoverability
  - Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

- the term "service-oriented" and various abstract SOA models existed before the arrival of Web services.
  - However, no one technology advancement has been so suitable and successful in manifesting SOA than Web services.

Message itself is standardized, both in format and in how it represents its payload. The use of SOAP, WSDL, XML, and XML Schema allow for messages to be fully self-contained and support the underlying agreement that to communicate, services require nothing more than a knowledge of each other's service descriptions.

The use of an open, standardized messaging model eliminates the need for underlying service logic to share type systems and supports the loosely coupled paradigm.

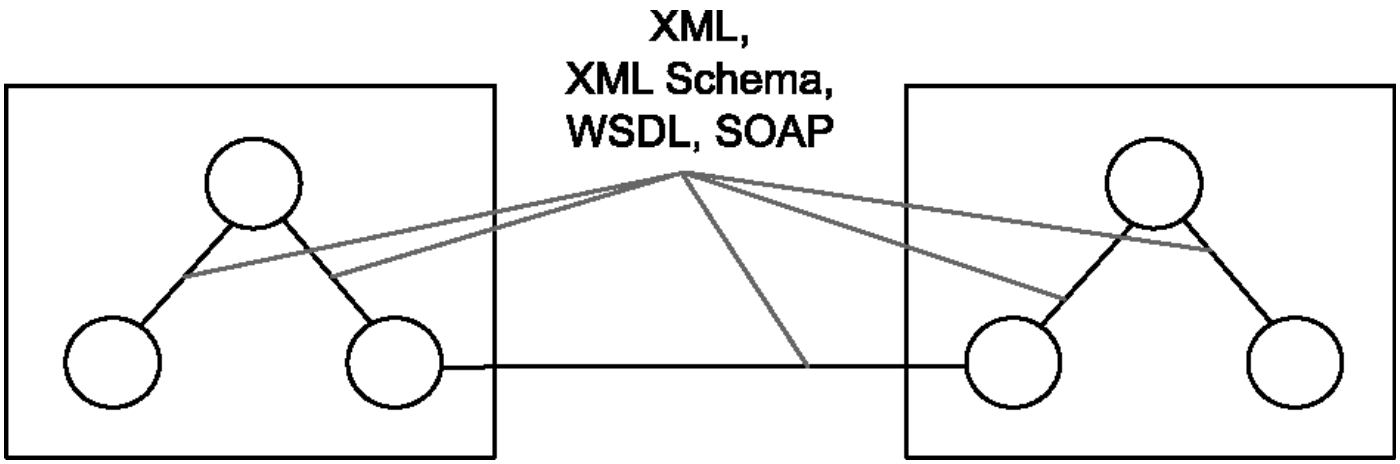


**Figure 3.5:** Standard open technologies are used within and outside of solution boundaries.

regardless of how proprietary a development environment is, as long as it supports the creation of standard Web services, it can be used to create a non-proprietary service interface layer, opening up interoperability opportunities with other, service-capable applications

allows organizations to choose best-of-breed environments for specific applications.

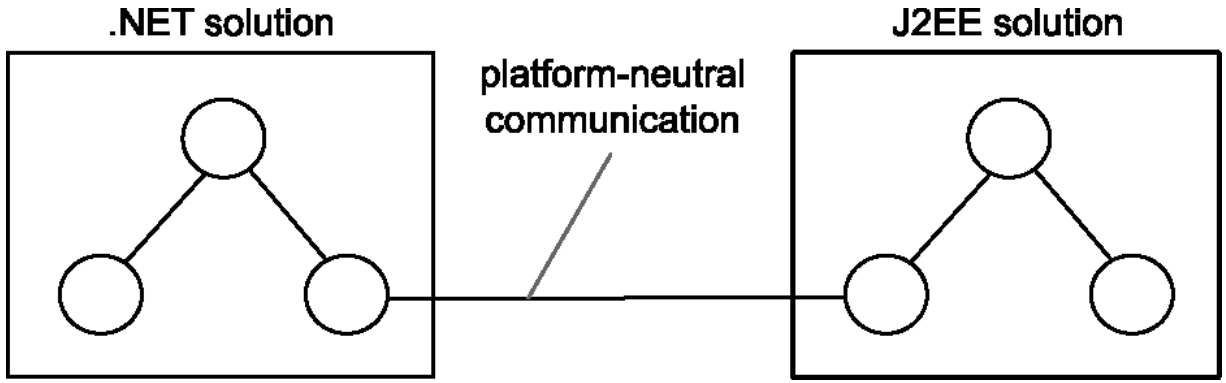


**Figure 3.6:** Disparate technology platforms do not prevent service-oriented solutions from interoperating.

SOA supports and encourages the advertisement and discovery of services throughout the enterprise and beyond. A serious SOA will likely rely on some form of service registry or directory to manage service descriptions

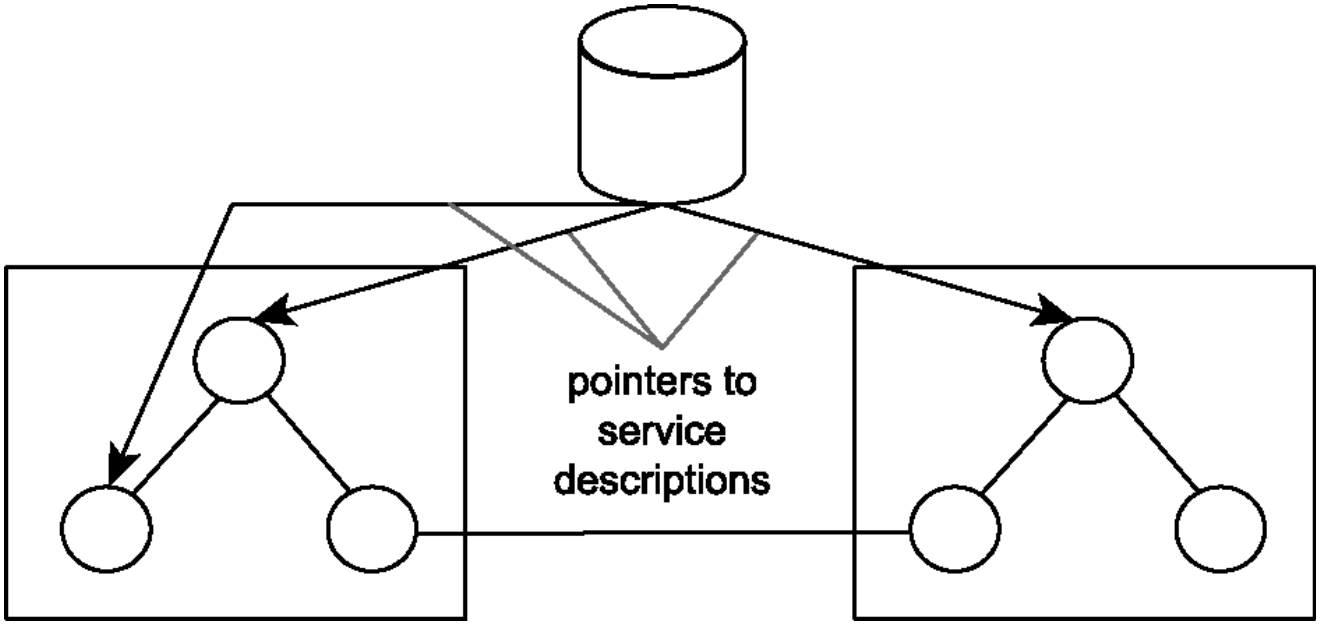


**Figure 3.7:** Registries enable a mechanism for the discovery of services.

When properly standardized, this leads to service-oriented integration architectures wherein solutions themselves achieve a level of intrinsic interoperability. Fostering this characteristic can significantly alleviate the cost and effort of fulfilling future cross-application integration requirements.

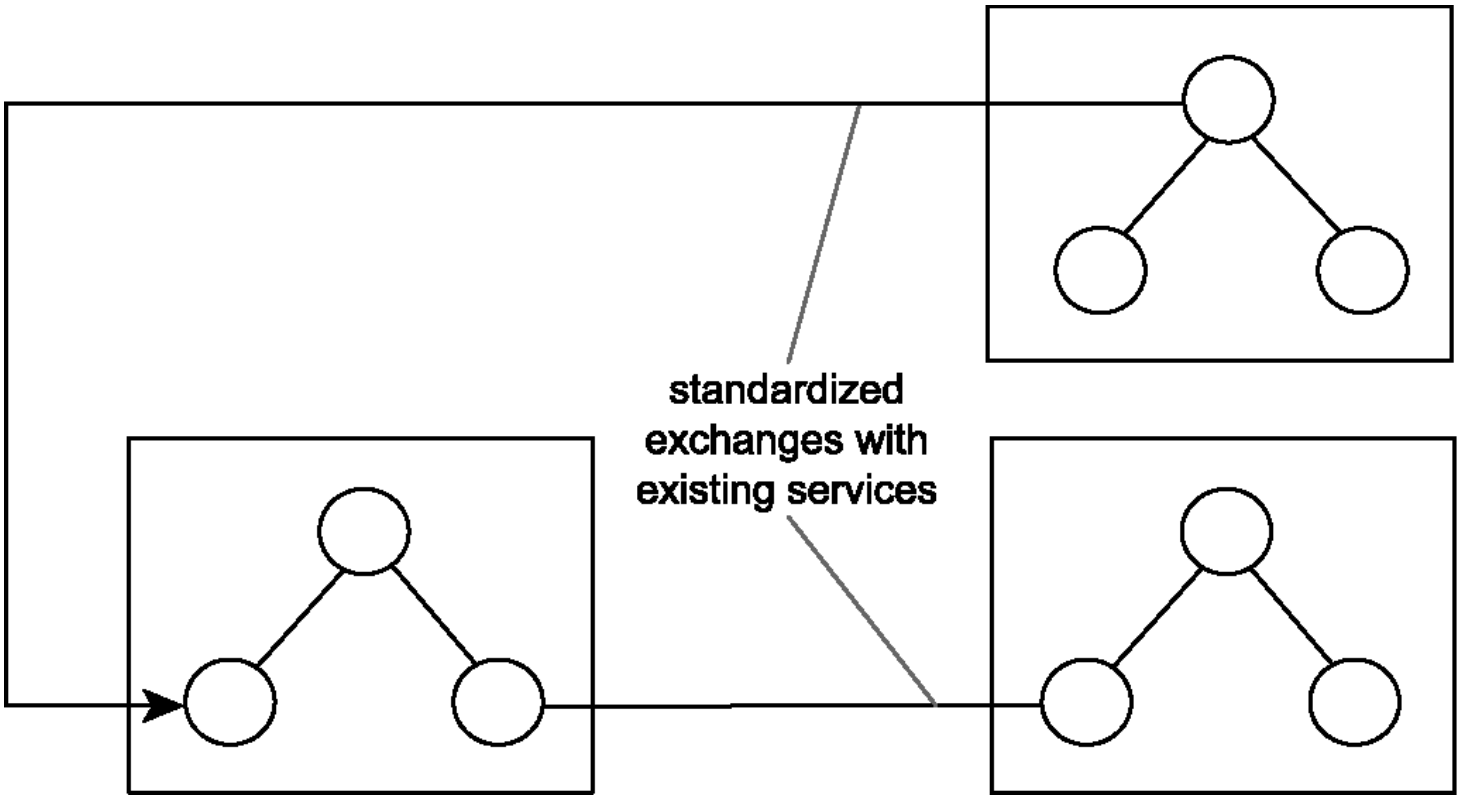


**Figure 3.8:** Intrinsically interoperable services enable unforeseen integration opportunities.

Establishing SOA within an enterprise does not necessarily require that you replace what you already have.

Obviously, the incorporation of SOA with previous platforms can lead to a variety of hybrid solutions. However, the key aspect is that the communication channels achieved by this form of service-oriented integration are all uniform and standardized

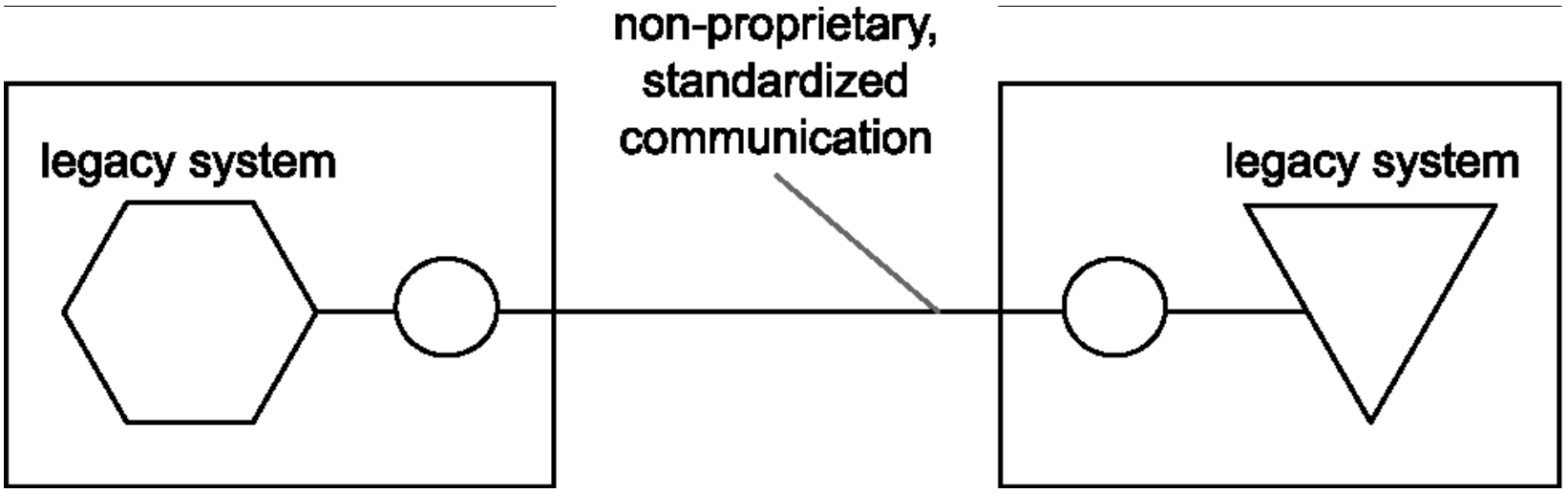


**Figure 3.9:** Services enable standardized federation of disparate legacy systems.

WS-* platform allows for the creation of streamlined and optimized service-oriented architectures, applications, services, and even messages.

A broader example of composability is represented by the second-generation Web services framework that is evolving out of the release of the numerous WS-* specifications. The modular nature of these specifications allows an SOA to be composed of only the functional building blocks it requires.

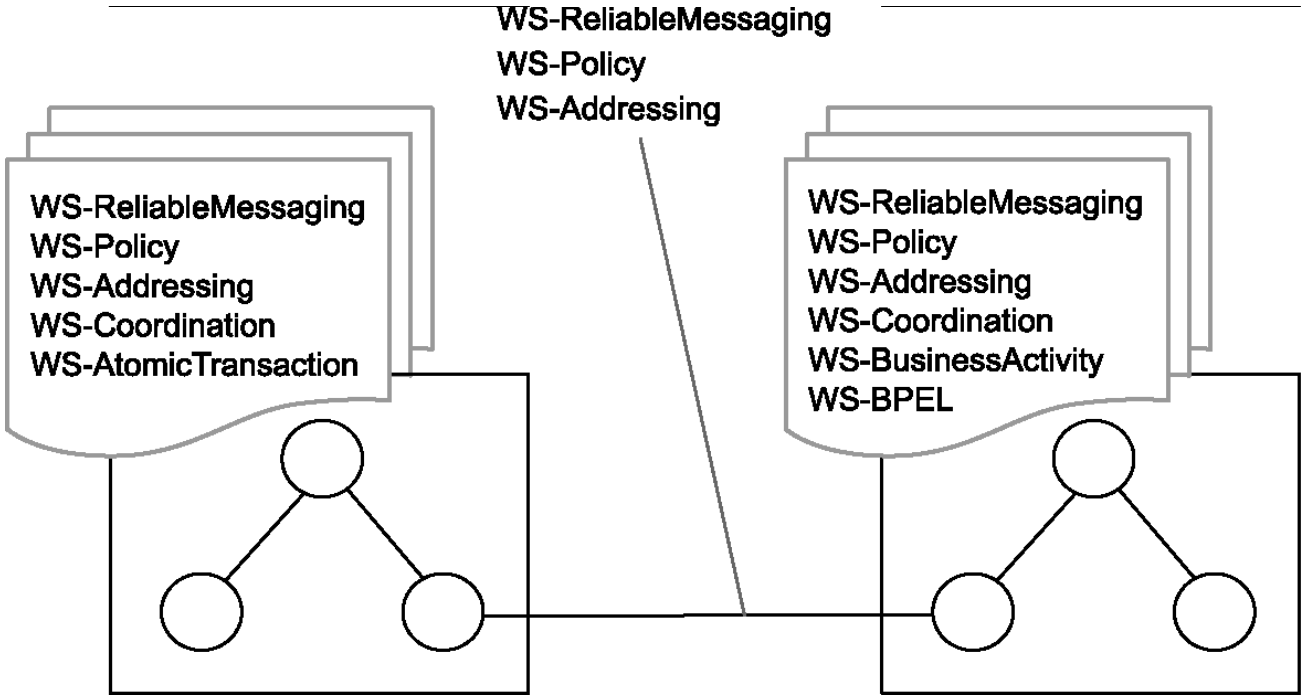


**Figure 3.10:** Different solutions can be composed of different extensions and can continue to interoperate as long as they support the common extensions required.

SOA establishes an environment that promotes reuse on many levels.

Collections of services that form service compositions can themselves be reused by larger compositions.

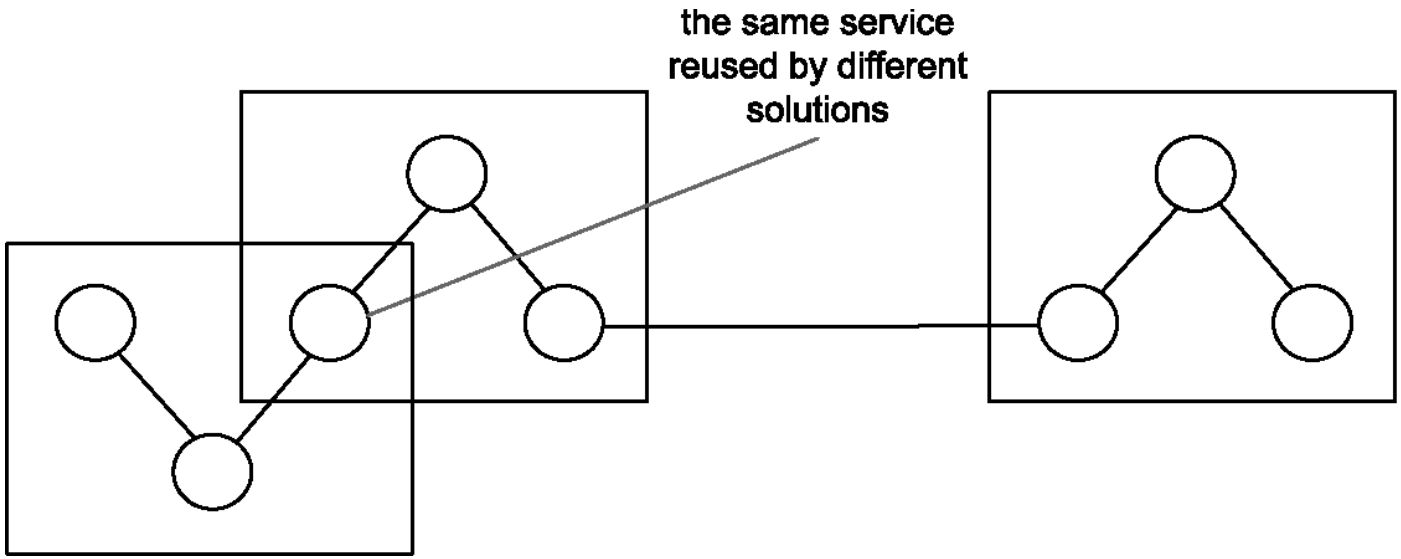


**Figure 3.11:** Inherent reuse accommodates unforeseen reuse opportunities.

When expressing encapsulated functionality through a service description, SOA encourages you to think beyond immediate, point-to-point communication requirements. When service logic is properly partitioned via an appropriate level of interface granularity, the scope of functionality offered by a service can sometimes be extended without breaking the established interface

Because the loosely coupled relationship fostered among all services minimizes inter-service dependencies, extending logic can be achieved with significantly less impact.

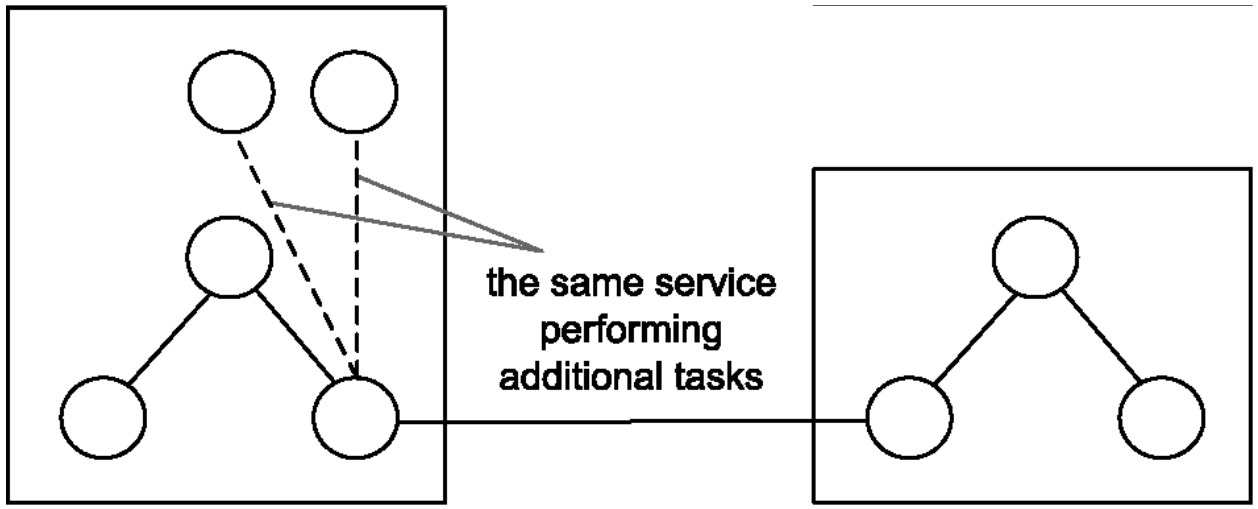


**Figure 3.12:** Extensible services can expand functionality with minimal impact.

services can be designed to express business logic. BPM models, entity models, and other forms of business intelligence can be accurately represented through the coordinated composition of business-centric services.

Partitioning business logic into services that can then be composed has significant implications as to how business processes can be modeled

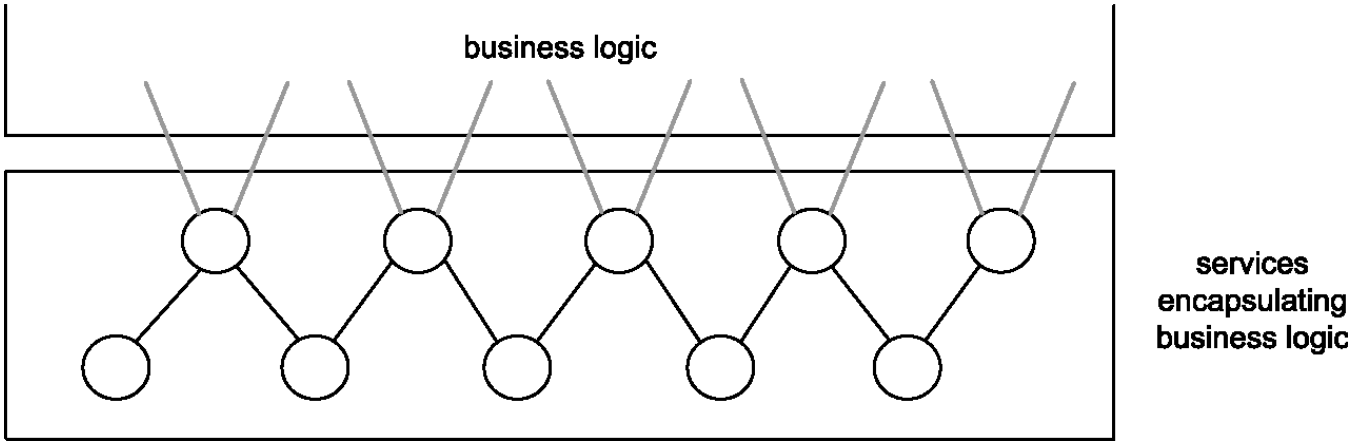


**Figure 3.13:** A collection (layer) of services encapsulating business process logic.

One of the characteristics that tends to evolve naturally through the application of service-oriented design principles is that of abstraction. Typical SOAs can introduce layers of abstraction by positioning services as the sole access points to a variety of resources and processing logic.

When applied through proper design, abstraction can be targeted at business and application logic. For example, by establishing a layer of endpoints that represent entire solutions and technology platforms, all of the proprietary details associated with these environments disappear

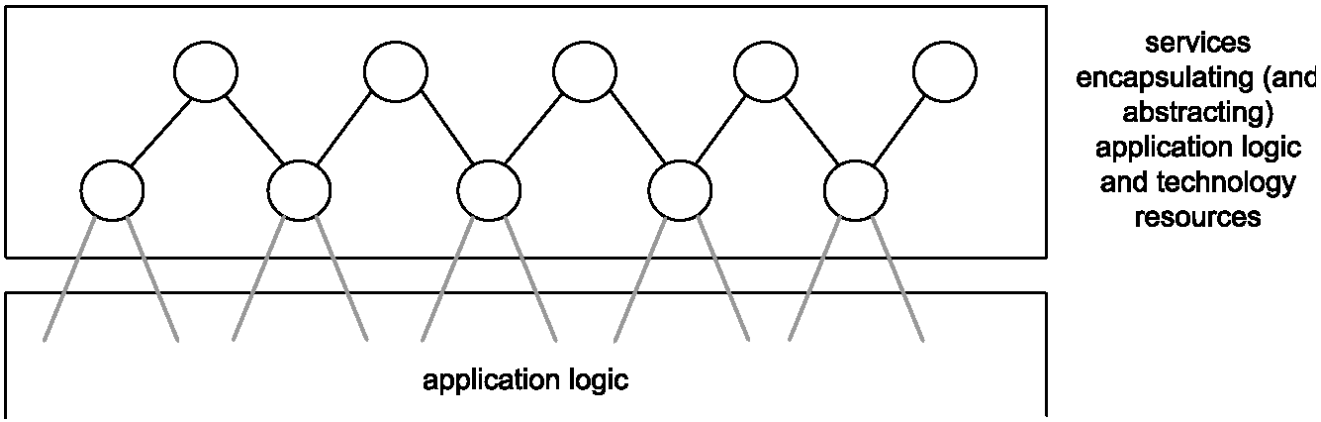


**Figure 3.14:** Application logic created with proprietary technology can be abstracted through a dedicated service layer.

Services only require an awareness of each other, allowing them to evolve independently.

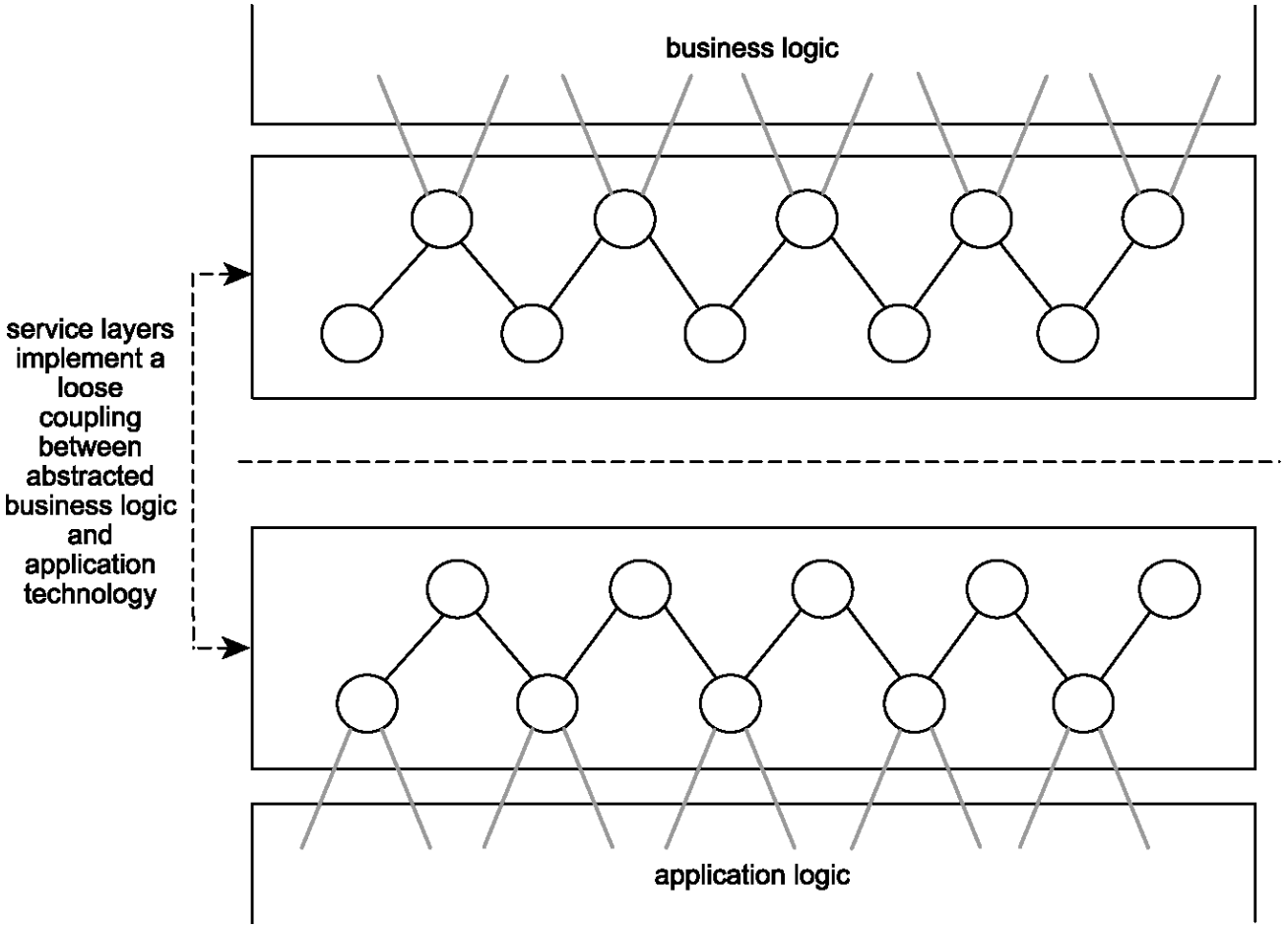


**Figure 3.15:** Through the implementation of service layers that abstract business and application logic, the loose coupling paradigm can be applied to the enterprise as a whole.

Whether the result of an internal reorganization, a corporate merger, a change in an organization's business scope, or the replacement of an established technology platform, an organization's ability to accommodate change determines the efficiency with which it can respond to unplanned events.

Change in an organization's business logic can impact the application technology that automates it.
Change in an organization's application technology infrastructure can impact the business logic automated by this technology.
The more dependencies that exist between these two parts of an enterprise, the greater the extent to which change imposes disruption and expense.

By leveraging service business representation, service abstraction, and the loose coupling between business and application logic provided through the use of service layers, SOA offers the potential to increase organizational agility

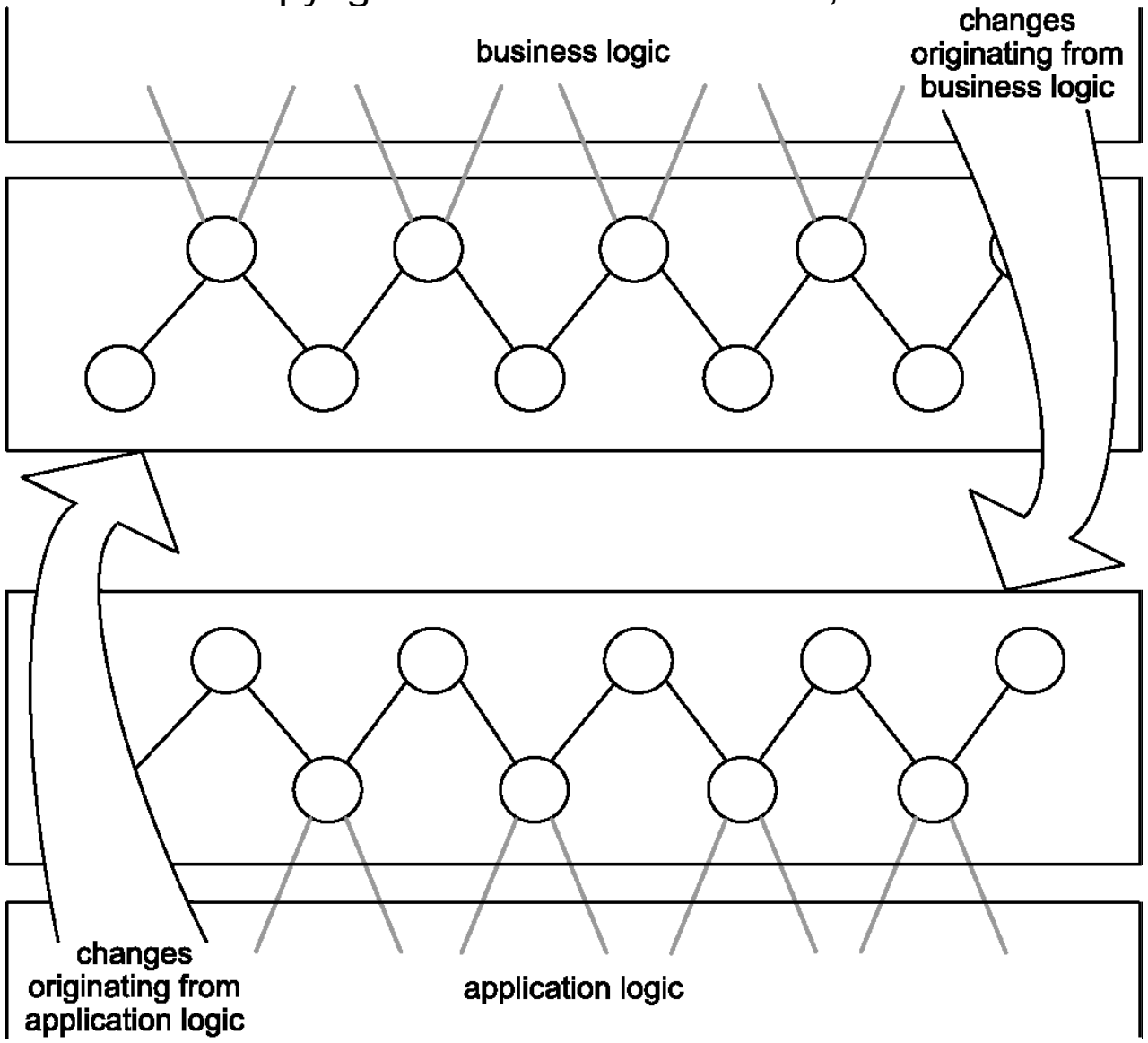


**Figure 3.16:** A loosely coupled relationship between business and application technology allows each end to more efficiently respond to changes in the other.

Contemporary SOA represents an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services.

SOA can establish an abstraction of business logic and technology that may introduce changes to business process modeling and technical architecture, resulting in a loose coupling between these models.

SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise.

SOA is ideally standardized throughout an enterprise, but achieving this state requires a planned transition and the support of a still evolving technology set.