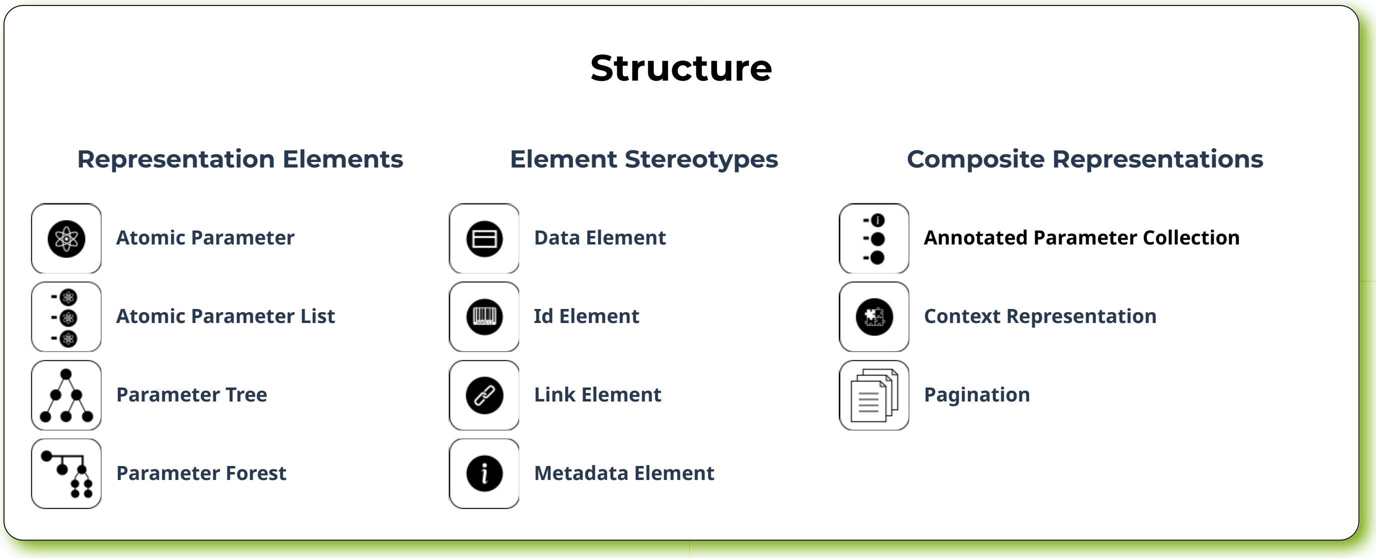
Structure Patterns

What is the adequate number of API message parameters and how should these parameters be structured?

← Responsibility Patterns   Quality Patterns →

Category Overview

The Structure Patterns category comprises the following patterns:



API contracts contain the unique address of one or more API endpoints (e.g., a URI in SOAP or RESTful HTTP) and the identifier of the operation (e.g., the service name of a SOAP Web service call or the name of a RESTful HTTP resource) plus the message structures (i.e., the request message and response message of the operation).

This category of our pattern language deals with such aspects of interface representation structure design. It addresses the following design issue:

What is an adequate number of API message parameters? How should these parameters be structured? How can they be grouped and annotated with usage instructions?

For instance, in a RESTful HTTP context this design issue can be interpreted as how the URI, form, and body parameters transported with the message are structured and how many parameters are transported. In a RESTful HTTP API usually the request body is used for data sent to or received from the server (e.g., in JSON, XML, or another MIME type), and the query parameters of the URL are used to specify the exact data requested.

In a WSDL/SOAP context, we can interpret this design issue as how the SOAP message parts should be organized and which data types should be used to define the corresponding elements in an XML schema.

The following Structure Patterns have been published so far:

Representation Elements

	PATTERN: ATOMIC PARAMETER
Problem	How can an API provider define a single, primitive data element as parameter in a request message or a response message?

	PATTERN: ATOMIC PARAMETER
Solution	To exchange a simple, unstructured data element (such as a number, a string, or a boolean value), define only a single scalar parameter for a message.

	PATTERN: ATOMIC PARAMETER LIST
Problem	How can the API provider define multiple primitive data elements as parameters in a request message or a response message?
Solution	To transmit two or more simple, unstructured data elements, define these message parameters as multiple Atomic Parameters (such as numbers, strings, or boolean values) and arrange them in an ordered list.

	PATTERN: PARAMETER FOREST
Problem	How can the API provider define repetitive or nested parameter data structures that cannot be represented well in a single tree structure?
Solution	Define the parameter representation of a message as multiple simple and composite data structure representations, including scalars, lists, and complex types like trees, arranged in an ordered list of those structures.

	PATTERN: PARAMETER TREE
Problem	How can the API provider define tree data structures in the parameters of a message? How can the API provider define repetitive or nested data elements in the parameters of a message?
Solution	Define the parameter representation of a message based on a single root data element that contains one or more subordinate composite data structures such as tuples, arrays, or trees.

Composite Representations

	PATTERN: CONTEXT REPRESENTATION
Problem	How can API consumer and provider exchange context information without relying on the remoting protocols (for instance, application protocols such as HTTP or transport protocols such as TCP)? In particular, how can identity information and quality properties in a request be made visible to related subsequent requests?
Solution	Combine and group all Metadata Elements that carry the desired information into a custom representation element in request and/or response messages. Do not transport this single Context Representation in protocol headers, but place it in the message payload. Position and mark the consolidated Context Representation element so that it is easy to find and distinguish from other Data Elements. Separate global from local context in a call chain by structuring the Context Representation accordingly.

	PATTERN: PAGINATION
Problem	How can an API provider deliver large sequences of structured data without overwhelming clients?
Solution	Divide large response data sets into manageable and easy-to-transmit chunks (also known as pages). Send one chunk of partial results per response message and inform the client about the total and/or remaining number of chunks. Provide optional

	PATTERN: PAGINATION
	filtering capabilities to allow clients to request a particular selection of results. For extra convenience, include a reference to the next page from the current one.

Element Stereotypes

	PATTERN: DATA ELEMENT
Problem	How can domain/application-level information be exchanged between API clients and API providers without exposing provider-internal data definitions in the API? How can API client and API provider be decoupled from a data management point of view (i.e., format autonomy be achieved)?
Solution	Define a dedicated/designated vocabulary, a Published Language in Domain-Driven Design (DDD) terms, consisting of Data Elements for request and response messages that wrap all or parts of the domain model in the system that exposes the API (for instance, aggregates, entities and value objects when practicing DDD).

	PATTERN: ID ELEMENT
Problem	How can instances of API elements be distinguished from each other? API elements requiring identification may be endpoints, operations, and parts of request and response message content.
Solution	Introduce a special type of Data Element, a unique Id Element, to identify API endpoints, operations, and message representation elements that have to be distinguished from each other, for instance because they have a lifecycle. Use this Id Element consistently throughout API description and implementation. Decide whether the Id Element is globally unique or only valid within the context of this particular API.

	PATTERN: LINK ELEMENT
Problem	How can API endpoints and operations be referenced in request and response message payloads? How can globally unique, network-accessible pointers to API endpoints and their operations be included in request and response messages? How can these pointers be used to allow clients to drive provider-side state transitions and operation invocation sequencing?
Solution	Include a special type of Id Element, a Link Element, to the signatures of request and response messages. These Link Elements act as a textual pointers (i.e., hyperlinks) to other endpoints and operations. Let a Metadata Element indicate the nature of the relationship (i.e., specify its semantic type).

	PATTERN: METADATA ELEMENT
Problem	How can messages be enriched with additional information so that receivers can interpret the message content correctly, without having to hard-code assumptions about the data semantics?
Solution	Introduce one or more Metadata Elements with their name and type to explain and enhance the other representation elements that appear in request and response messages (e.g., the Data Elements in Atomic Parameter Lists, Parameter Trees and Parameter Forests). Populate the values of the Metadata Elements thoroughly and consistently; process them as and if needed to steer interoperable message consumption and processing.

Earlier versions of a subset of these patterns are featured in our EuroPLoP 2017 paper Interface Representation Patterns: Crafting and Consuming Message-Based Remote APIs. A free PDF version of this paper can be downloaded here.

← Responsibility Patterns

Quality Patterns →

← BACK TO HOMEPAGE

Microservice API Patterns