Analysis and Modeling with Web Services and Microservices

by Thomas Erl, Arcitura Education Inc.

The following is an excerpt from the new book “Service-Oriented Architecture: Analysis & Design for Services and Microservices (2nd Edition)”. For more information about this book, visit www.servicetechbooks.com/cd.

This chapter provides a detailed step-by-step process for modeling Web service candidates.

6.1 Web Service Modeling Process

A service modeling process can essentially be viewed as an exercise in organizing the information we gathered in Steps 1 and 2 of the parent service-oriented analysis process that was described in Chapter 4. Figure 6.1 provides a generic service modeling process suitable for Web services that can be further customized. This chapter follows this generic service modeling process by describing each step and further providing case study examples.

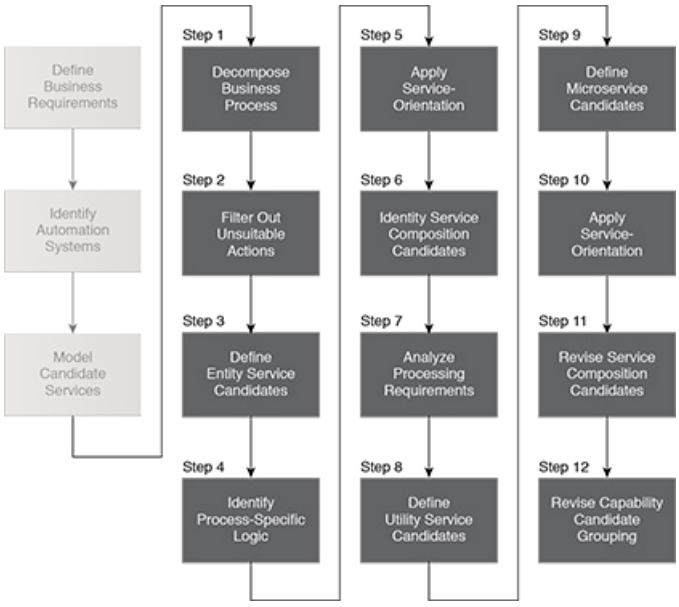

Figure 6.1 – A sample service modeling process for Web services.

Case Study Example

TLS outsources a number of its employees on a contract basis to perform various types of specialized maintenance jobs. When these employees fill out their weekly timesheets, they are required to identify what portions of their time are spent at customer sites. Currently, the amount of time for which a customer is billed is determined by an A/R clerk who manually enters hours from an appointment schedule that is published prior to the submission of timesheets.

Copyright © Arcitura Education Inc.

Discrepancies arise when employee timesheet entries do not match the hours billed on customer invoices. To address this problem and streamline the overall process, TLS decides to integrate its third-party time tracking system with its large, distributed accounting solution.

The resulting Timesheet Submission business process is shown in Figure 6.2. Essentially, every timesheet that TLS receives from outsourced employees needs to undergo a series of verification steps. If the timesheet is verified successfully, the process ends and the timesheet is accepted. Any timesheet that fails verification is submitted to a separate rejection step prior to the process ending.

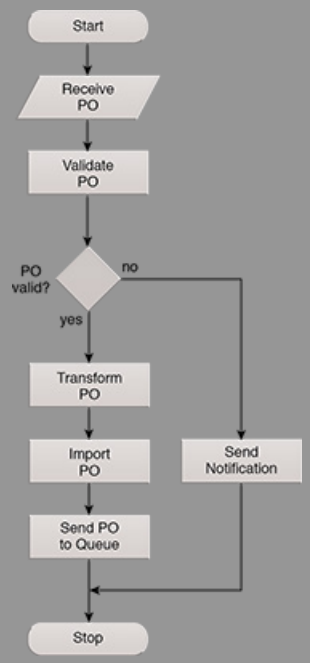

Figure 6.2 – The TLS Timesheet Submission business process.

Step 1: Decompose the Business Process (into Granular Actions)

We begin by taking the documented business process and breaking it down into a series of granular process steps. The business process workflow logic needs to be decomposed into its most granular representation of processing steps, which may differ from the level of granularity at which the process steps were originally documented.

Copyright © Arcitura Education Inc.

Case Study Example

Here is a breakdown of the current business process steps:

1. Receive Timesheet

2. Verify Timesheet

3. If Timesheet is Verified, Accept Timesheet Submission and End Process

4. Reject Timesheet Submission

Although it only consists of four steps at this point, there is more to this business process. The details are revealed as the TLS team decomposes the process logic. They begin with the Receive Timesheet step, which is split into two smaller steps:

1a. Receive Physical Timesheet Document

1b. Initiate Timesheet Submission

The Verify Timesheet step is actually a subprocess in its own right and can therefore be broken down into the following more granular steps:

2a. Compare Hours Recorded on Timesheet to Hours Billed to Clients

2b. Confirm That Authorization Was Given for Any Recorded Overtime Hours

2c. Confirm That Hours Recorded for Any Particular Project Do Not Exceed a Pre-Defined Limit for That Project

2d. Confirm That Total Hours Recorded for One Week Do Not Exceed a Pre-Defined Maximum for That Worker

Upon subsequent analysis, TLS further discovers that the *Reject Timesheet Submission* process step can be decomposed into the following granular steps:

4a. Update the Worker's Profile Record to Keep Track of Rejected Timesheets

4b. Issue a Timesheet Rejection Notification Message to the Worker

4c. Issue a Notification to the Worker's Manager

Having drilled down the original process steps, TLS now has a larger amount of process steps. It organizes these steps into an expanded business process workflow (Figure 6.3):

- Receive Timesheet
- Compare Hours Recorded on Timesheet to Hours Billed to Clients

If Hours Do Not Match, Reject Timesheet Submission

- Confirm That Authorization Was Given for Any Recorded Overtime Hours
- If Authorization Confirmation Fails, Reject Timesheet Submission
- Confirm That Hours Recorded for Any Particular Project Do Not Exceed a Pre-Defined Limit for That Project

Copyright © Arcitura Education Inc.

- Confirm That Total Hours Recorded for One Week Do Not Exceed a Pre-Defined Maximum for That Worker
- If Hours Recorded Confirmation Fails, Reject Timesheet Submission
- Reject Timesheet Submission
- Generate a Message Explaining the Reasons for the Rejection
- Issue a Timesheet Rejection Notification Message to the Worker

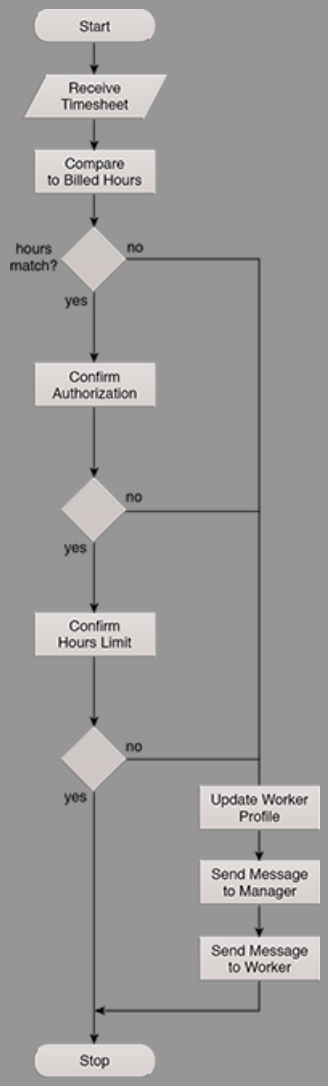

Figure 6.3 – The revised TLS Timesheet Submission business process.

- Issue a Notification to the Worker's Manager
- If Timesheet Is verified, Accept Timesheet Submission and End Process

Finally, TLS further simplifies the business process logic into the following set of granular actions:

- Receive Timesheet
- Initiate Timesheet Submission
- Get Recorded Hours for Customer and Date Range
- Get Billed Hours for Customer and Date Range
- Compare Recorded Hours with Billed Hours
- If Hours Do Not Match, Reject Timesheet Submission
- Get Overtime Hours for Date Range
- Get Authorization
- Confirm Authorization
- If Authorization Confirmation Fails, Reject Timesheet Submission
- Get Weekly Hours Limit
- Compare Weekly Hours Limit with Recorded Hours
- If Hours Recorded Confirmation Fails, Reject Timesheet Submission
- Update Employee History
- Send Message to Employee
- Send Message to Manager
- If Timesheet Is verified, Accept Timesheet Submission and End Process

Step 2: Filter Out Unsuitable Actions

Some steps within a business process can be easily identified as not belonging to the potential logic that should be encapsulated by a service candidate. These can include manual process steps that cannot or should not be automated and process steps performed by existing legacy logic for which service candidate encapsulation is not an option. By filtering out these parts, we are left with the processing steps most relevant to our service modeling process.

Case Study Example

- After reviewing each of the business process steps, those that either cannot or do not belong in a service-oriented solution are removed. The following list revisits the decomposed actions. The first action is crossed out because it is performed manually by an accounting clerk.

Copyright © Arcitura Education Inc.

- Receive Timesheet
- Initiate Timesheet Submission
- Get Recorded Hours for Customer and Date Range
- Get Billed Hours for Customer and Date Range
- Compare Recorded Hours with Billed Hours
- If Hours Do Not Match, Reject Timesheet Submission
- Get Overtime Hours for Date Range
- Get Authorization
- Confirm Authorization
- If Authorization Confirmation Fails, Reject Timesheet Submission
- Get Weekly Hours Limit
- Compare Weekly Hours Limit with Recorded Hours
- If Hours Recorded Confirmation Fails, Reject Timesheet Submission
- Update Employee History
- Send Message to Employee
- Send Message to Manager
- If Timesheet Is Verified, Accept Timesheet Submission and End Process

Each of the remaining actions is considered a service capability candidate.

Step 3: Define Entity Service Candidates

Review the processing steps that remain and determine one or more logical contexts with which these steps can be grouped. Each context represents a service candidate. The contexts you end up with will depend on the types of business services you have chosen to build. For example, task services will require a context specific to the process, whereas entity services will introduce the need to group processing steps according to their relation to previously defined entities. An SOA can also consist of a combination of these business service types.

It is important that you do not concern yourself with how many steps belong to each group. The primary purpose of this exercise is to establish the required set of contexts. Equipping entity service candidates with additional capability candidates that facilitate future reuse is also encouraged. Therefore, the scope of this step can be expanded to include an analysis of additional service capability candidates not required by the current business process, but added to round out entity services with a complete set of reusable operations.

Copyright © Arcitura Education Inc.

Case Study Example

TLS business analysts support the service modeling effort by producing an entity model relevant to the Timesheet Submission business process logic (Figure 6.4).

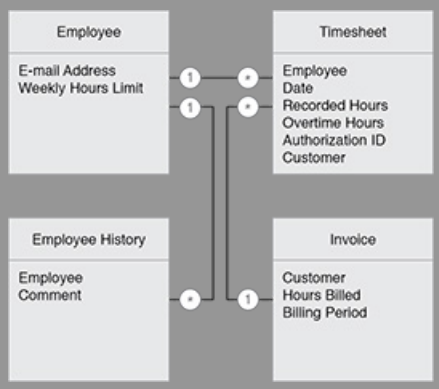

Figure 6.4 – A TLS entity model displaying business entities pertinent to the Timesheet Submission business process.

The TLS team studies this model, along with the list of granular service capability candidates identified during the previous analysis step. They subsequently identify the service capability candidates considered agnostic. All those classified as nonagnostic are bolded, as follows:

- **Initiate Timesheet Submission**
- Get Recorded Hours for Customer and Date Range
- Get Billed Hours for Customer and Date Range
- **Compare Recorded Hours with Billed Hours**
- **If Hours Do Not Match, Reject Timesheet Submission**
- Get Overtime Hours for Date Range
- Get Authorization
- **Confirm Authorization**
- **If Authorization Confirmation Fails, Reject Timesheet Submission**
- Get Weekly Hours Limit
- **Compare Weekly Hours Limit with Recorded Hours**
- **If Hours Recorded Confirmation Fails, Reject Timesheet Submission**

Copyright © Arcitura Education Inc.

- Update Employee History
- Send Message to Employee
- Send Message to Manager
- **If Timesheet Is Verified, Accept Timesheet Submission and End Process**

First, the Timesheet entity is reviewed. It is decided that this entity warrants a corresponding entity service candidate simply called “Timesheet.” Upon analysis of its attributes, TLS further determines that the following service capability candidates should be grouped with the entity service candidate:

- Get Recorded Hours for Customer and Date Range
- Get Overtime Hours for Date Range
- Get Authorization

However, upon subsequent analysis, it is determined that the first two capability candidates could be made more reusable by removing the requirement that a date range be the only query criteria. Although this particular business process will always provide a date range, business analysts point out that other processes will want to request recorded or overtime hours based on other parameters. The result is a revised set of capability candidates, as shown in Figure 6.5.

Analysts then take a look at the Invoice entity. They again agree that this entity deserves representation as a standalone entity service candidate. They name this service “Invoice” and assign it the following capability candidate:

- Get Billed Hours for Customer and Date Range

When the service-orientation principle of Service Reusability is again considered, the analysts decide to expand the scope of this service candidate by altering the function of the chosen capability candidate and then by adding a new one, as shown in Figure 6.6. Now service consumers can retrieve invoice-related customer information and billed hours information separately.

The Employee and Employee History entities are reviewed next. Because they are closely related to each other, it is decided that they can be jointly represented by a single entity service candidate called “Employee.” Two service capability candidates are assigned, resulting in the service candidate definition displayed in Figure 6.7.

The TLS team considers also adding a Send Notification service capability candidate to the Employee service candidate, but then determines that this functionality is best separated into a utility service candidate.

Copyright © Arcitura Education Inc.

Figure 6.5 – The Timesheet service candidate.

Figure 6.6 – The Invoice service candidate.

Figure 6.7 – The Employee service candidate.

As a result, the remaining two actions are put aside for now until utility services are defined, later in this process:

- Send Message to Employee
- Send Message to Manager

Step 4: Identify Process-Specific Logic

Any parts of the business process logic remaining after we complete Step 3 will need to be classified as non-agnostic or specific to the business process. Common types of actions that fall into this category include business rules, conditional logic, exception logic, and the sequence logic used to execute the individual business process actions.

Note that not all non-agnostic actions necessarily become service capability candidates. Many process-specific actions represent decision logic and other forms of processing that are executed within the service logic.

Note

There may be sufficient information about the identified non-agnostic logic to determine whether any part of this logic may be suitable for encapsulation by one or more microservices. In this case, microservice candidates can be defined as part of this step together with task service candidates. However, it is recommended that you wait until Step 9 to formally define the necessary microservice(s) for this solution because upcoming service modeling steps can identify additional non-agnostic logic and can further assist with the definition of solution implementation and processing requirements.

Case Study Example

The following actions are considered non-agnostic because they are specific to the Timesheet Submission business process:

- Initiate Timesheet Submission
- Compare Recorded Hours with Billed Hours
- If Hours Do Not Match, Reject Timesheet Submission
- Confirm Authorization
- If Authorization Confirmation Fails, Reject Timesheet Submission
- Compare Weekly Hours Limit with Recorded Hours
- If Hours Recorded Confirmation Fails, Reject Timesheet Submission
- If Timesheet Is Verified, Accept Timesheet Submission and End Process

The Initiate Timesheet Submission action forms the basis of a service capability candidate, as explained in the upcoming Timesheet Submission task service candidate description. The remaining actions are bolded to indicate that they represent logic that is carried out within the Timesheet Submission task service, upon execution of the Initiate Timesheet Submission action, which is renamed to the Start service capability candidate (Figure 6.8).

Copyright © Arcitura Education Inc.

Figure 6.8 – The Timesheet Submission service candidate with a single service capability that launches the automation of the Timesheet Submission business process.

Step 5: Apply Service-Orientation

This step gives us a chance to make adjustments and apply key service-orientation principles. Depending on the insight we may have as to the specific nature of logic that will be required within a given service candidate, we may have an opportunity to further augment the scope and structure of service candidates. Principles such as Service Loose Coupling (293), Service Abstraction (294), and Service Autonomy (297) may provide suitable considerations at this stage.

Note

The application of the Service Autonomy (297) principle in particular may raise considerations that could introduce the need for some of the identified logic to be encapsulated within microservices. In this case, microservice candidates can be defined as part of this step and will be subject to further review during Step 9, when microservices are formally defined.

Step 6: Identify Service Composition Candidates

Identify a set of the most common scenarios that can take place within the boundaries of the business process. For each scenario, follow the required processing steps as they exist now.

This exercise accomplishes the following:

- Provides insight as to how appropriate the grouping of your process steps is
- Demonstrates the potential relationship between task and entity service layers
- Identifies potential service compositions
- Highlights any missing workflow logic or processing steps

Ensure that, as part of your chosen scenarios, you include failure conditions that involve exception handling logic. Note also that any service layers you establish at this point are still preliminary and still subject to revisions during the design process.

Copyright © Arcitura Education Inc.

Case Study Example

Figure 6.9 displays a preliminary service composition candidate comprised of task and entity service candidates. This composition model is the result of various composition scenarios mapped out by the TLS team to explore different success and failure conditions when carrying out the automation of the Timesheet Submission process.

As a result of mapping different service activities within the boundaries of this service composition candidate, TLS feels confident that no further non-agnostic process logic is missing from what it has identified so far.

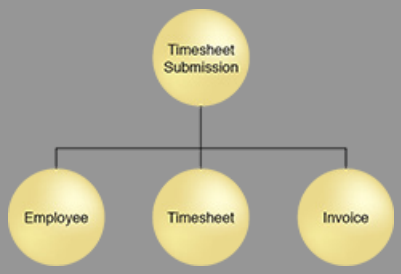

Figure 6.9 – A look at the service composition candidate hierarchy that is formed as various service interaction scenarios are explored during this stage.

Step 7: Analyze Processing Requirements

By the end of Step 6, you will have created a business-centric view of your services layer. This view could very well include both utility and business service candidates, but the focus so far has been on representing business process logic.

This and the upcoming steps ask us to identify and dissect the underlying processing and implementation requirements of service candidates. We do this to abstract any further technology-centric service logic that may warrant the introduction of microservices or that may add to the utility service layer. To accomplish this, each processing step identified so far is required to undergo a mini-analysis.

Specifically, what we need to determine is:

- What underlying processing logic needs to be executed to process the action described by a given service capability candidate.
- Whether the required processing logic already exists or whether it needs to be newly developed.
- What resources external to the service boundary the processing logic may need to access—for example, shared databases, repositories, directories, legacy systems, etc.
- Whether any of the identified processing logic has specialized or critical performance and/or reliability requirements.

Copyright © Arcitura Education Inc.

- Whether the identified processing logic has any specialized or critical implementation and/or environmental requirements.

Note that any information gathered during Step 2 of the parent service-oriented analysis process covered in Chapter 4 will be referenced at this point.

Case Study Example

Upon assessing the processing requirements for the identified service candidates and the overall business process logic, the TLS team can confirm that the Send Message to Employee and Send Message to Manager actions will need to be encapsulated as part of a utility service layer. Based on the information available about the known processing requirements and the eventual service implementation environment, they cannot identify any further utility-centric logic.

During the review of the non-agnostic process logic that is currently within the scope of the Timesheet Submission task service, architects realize that a discrepancy exists in processing requirements. In particular, the Confirm Authorization action encompasses logic that is required to access a proprietary clearance repository. This interaction has significantly greater SLA requirements than the rest of the non-agnostic process logic in relation to performance and failover.

Keeping this logic grouped with the other logic that is part of the Timesheet Submission task service could risk this logic not executing as per its required metrics. Therefore, it is suggested that it be separated into one or more microservice candidates that would eventually benefit from the type of highly autonomous implementation that could guarantee the required performance and failover demands.

Step 8: Define Utility Service Candidates

In this step we break down each unit of agnostic processing logic into a series of granular actions. We need to be explicit about the labeling of these actions so that they reference the function they are performing. Ideally, we would not reference the business process step for which a given function is being identified.

Group these processing steps according to a pre-defined context. With utility service candidates, the primary context is a logical relationship between capability candidates. This relationship can be based on any number of factors, including:

- Association with a specific legacy system
- Association with one or more solution components
- Logical grouping according to type of function

Various other issues are factored in after service candidates are subjected to the service-oriented design process. For now, this grouping establishes a preliminary utility service layer.

Copyright © Arcitura Education Inc.

Case Study Example

Subsequent to assessing processing requirements for logic that may qualify for the utility service model, the TLS team revisits the Send Message to Employee and Send Message to Manager actions and groups them into a new reusable utility service, simply called Notification.

To make the service candidate more reusable, the two capability candidates are consolidated into one as shown in Figure 6.10.

Figure 6.10 – The Notification service candidate.

Note

Modeling utility service candidates is notoriously more difficult than entity service candidates. Unlike entity services where we base functional contexts and boundaries upon already-documented enterprise business models and specifications (such as taxonomies, ontologies, entity relationships, and so on), there are usually no such models for application logic. Therefore, it is common for the functional scope and context of utility service candidates to be continually revised during iterations of the service inventory analysis cycle.

Step 9: Define Microservice Candidates

We now turn our attention to the previously identified non-agnostic processing logic to determine whether any unit of this logic may qualify for encapsulation by a separate microservice. As discussed in Chapter 4, the microservice model can introduce a highly independent and autonomous service implementation architecture that can be suitable for units of logic with particular processing demands.

Typical considerations can include:

- Increased autonomy requirements
- Specific runtime performance requirements
- Specific runtime reliability or failover requirements
- Specific service versioning and deployment requirements

It is important to note that, due to their specialized implementation needs, the use of SOAP-based Web services may not be suitable for microservices, even when they are identified as part of a Web services-centric service modeling process. SOA architects are given the option to build microservices using alternative implementation technologies, which may introduce disparate or proprietary communication protocols.

SOA Patterns

The Dual Protocols pattern provides a standardized manner of supporting primary and secondary communication protocols with the same service inventory.

Case Study Example

The Confirm Authorization action that is part of the Timesheet Submission task service candidate logic is separated to form the basis of the Confirm Authorization microservice candidate (Figure 6.11), a REST service that executes this logic via a Confirm capability candidate.

For more information on service modeling steps distinct to REST services, see Chapter 7.

Figure 6.11 – The Confirm Authorization service candidate.

Step 10: Apply Service-Orientation

This step is a repeat of Step 7, provided here specifically for any new utility service candidates that may have emerged from the completion of Steps 8 and 9.

Step 11: Revise Service Composition Candidates

Revisit the original scenarios you identified in Step 6 and run through them again, this time incorporating the new utility service and capability candidates as well. This will result in the mapping of elaborate activities that bring expanded service compositions to life. Be sure to keep track of how business service candidates map to underlying utility service candidates during this exercise.

Copyright © Arcitura Education Inc.

Case Study Example

With the introduction of the Notification utility service and the Verify Timesheet microservice, the complexion of the Timesheet Submission composition hierarchy changes noticeably, as illustrated in Figure 6.12.

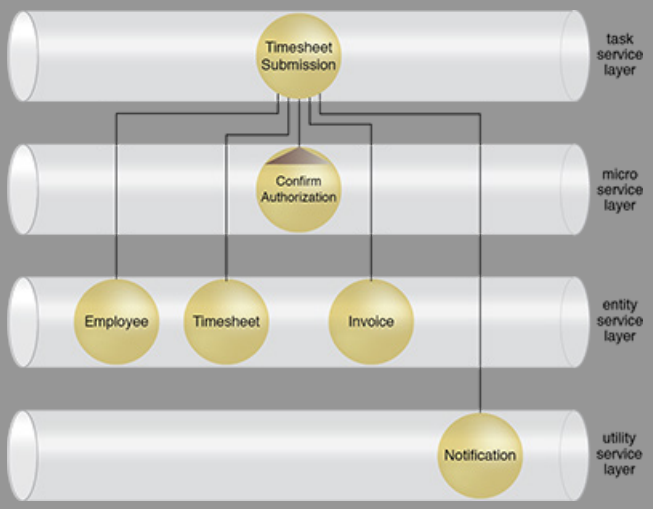

Figure 6.12 – The revised service composition candidate incorporating the new utility service and microservice.

Step 12: Revise Capability Candidate Grouping

Performing the mapping of the activity scenarios from Step 11 will usually result in changes to the grouping and definition of service capability candidates. It may also highlight any omissions in any further required processing steps, resulting in the addition of new service capability candidates and possibly even new service candidates.

Note

This process description assumes that this is the first iteration through the service modeling process. During subsequent iterations, additional steps need to be incorporated to check for the existence of relevant service candidates and service capability candidates.

Copyright © Arcitura Education Inc.

Thomas Erl

Thomas Erl is a best-selling IT author and founder of Arcitura™ Education Inc. Thomas has been the world's top-selling service technology author for over seven years and is the series editor of the Prentice Hall Service Technology Series from Thomas Erl (www.servicetechbooks.com). With more than 300,000 copies in print worldwide, his books have become international bestsellers and have been formally endorsed by senior members of major IT organizations, such as IBM, Microsoft, Oracle, Intel, Accenture, IEEE, HL7, MITRE, SAP, CISCO, HP, and many others.

Several of his books, including Cloud Computing Design Patterns*,* Cloud Computing: Concepts, Technology & Architecture*,* SOA Design Patterns*,* SOA Principles of Service Design*, and* SOA Governance*, were authored in collaboration with the IT community and have contributed to the definition of cloud computing technology mechanisms, the service-oriented architectural model and service-orientation as a distinct paradigm. His more recent title,* Service-Oriented Architecture: Analysis & Design for Services and Microservices*, formally positions and introduces new patterns for the Microservice architectural model as part of SOA.*

As CEO of Arcitura™ Education Inc. and in cooperation with SOA School, Cloud School and Big Data Science School, Thomas has led the development of curricula for the internationally recognized SOA Certified Professional (SOACP), Cloud Certified Professional (CCP) and Big Data Science Certified Professional (BDSCP) accreditation programs, which have established a series of formal, vendor-neutral industry certifications obtained by thousands of IT professionals around the world.

Thomas is the founding member of the SOA Manifesto Working Group and author of the Annotated SOA Manifesto (www.soa-manifesto.com). For 10 years, he was the editor of The Service Technology Magazine, and he further oversees the SOAPatterns.org, CloudPatterns.org and BigDataPatterns.org initiatives, which are dedicated to the on-going development of master pattern catalogs for service-oriented architecture, cloud computing and Big Data.

Thomas has toured more than 20 countries as a speaker and instructor, and regularly participates in international conferences. More than 100 articles and interviews by Thomas have been published in numerous publications, including The Wall Street Journal *and* CIO Magazine*.*

Copyright © Arcitura Education Inc.