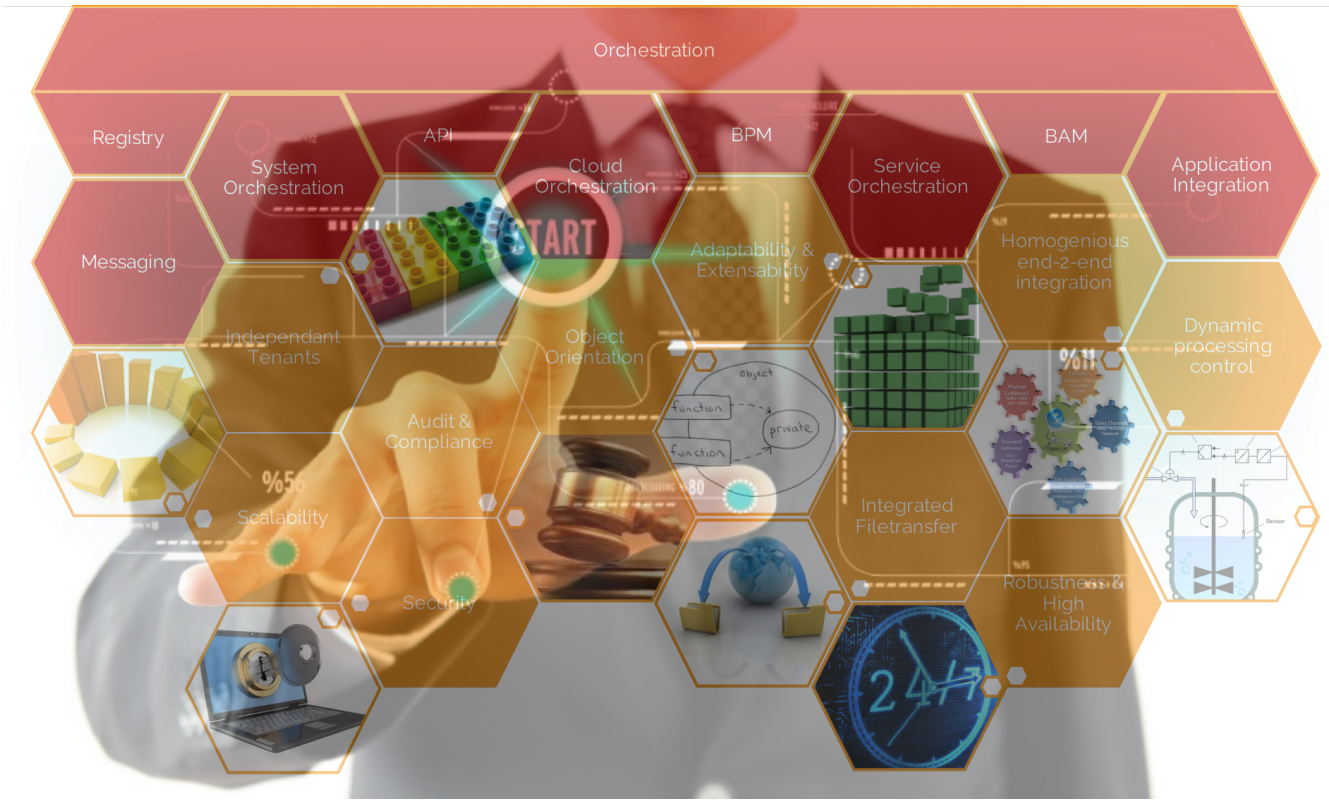
2016-04

AUTOMATION AND ORCHESTRATION FOR INNOVATIVE IT-AAS AND IOT ARCHITECTURES



thom Kunz

Smile-IT

http://www.smile-it.at

TABLE OF CONTENTS

Introduction and Summary **6**

Key findings **7**

Scalability in Automation **8**

Load scalability **8**

Expanding system resources 8

Dynamically adding physical/virtual servers to alter CPU, memory and disk space or increasing storage for workload peaks without downtime 10

Change the number of concurrently connected endpoints to one instance of the system 10

Endpoint reconnection following an outage 10

Administrative scalability **10**

Organizational unit support 11

Endpoint connection from different network segments 11

Seamless automation expansion for newly added resources 11

Functional scalability **12**

Enhance functionality through customized solutions 12

Template-based implementation and template transfer 12

Customizable template libraries with graphical modeler 13

Throughput **13**

High Availability and Robustness **14**

Multi-Server / Multi-Process architecture **14**

Transactions 14

Endpoint Failover 15

Disaster Recovery **16**

Key Performance Indicators **16**

Efficiency through object orientation **17**

SOLID object orientation **17**

"SOLID" IT automation 17

IT-aaS example 19

Patterns benefitting from object reusability **19**

Inheritance 19

Abstraction versus Specialization 19

Maintainability 19

Adaptability & Extensibility **21**

Hot plug-in of new integrations **21**

Leverage industry standards **22**

Dynamic data validation & exchange **22**

Integrated File Transfer **23**

Support of multiple different file transfer protocols **23**

Standard FT protocols **24**

Endpoint-to-endpoint File Transfer **24**

Process integration **24**

Audit & Compliance **26**

Traceability **26**

Statistics **26**

Version Management **27**

Monitoring **27**

Full Audit Trails **27**

Compliance Through Standardization **27**

Independent tenants **28**

Multi tenancy versus multi-client **28**

Multi-tenancy 28

Multi-client ... 28
Importance of Multi tenancy and Multi Client ... 28
Independent units within one system ... 28
Segregation of duties ... 29
Security ... 30
User security ... 30
Authentication ... 30
Personnel data base ... 30
Authorization & Access ... 31
API ... 31
Object level security ... 32
Separation of concern ... 32
Communication Security ... 32
Homogeneous end-to-end integration ... 34
ONE UX for all integrations ... 34
ONE platform for all automation ... 34
Functional integration with target systems ... 35
Dynamic processing control ... 36
Dynamic just-in-time execution ... 36
Pre/post processing of automation tasks ... 37
Easy-to-use extensible scripting ... 37
From Automation to Orchestration ... 38
What is Orchestration? ... 39
System versus Service Orchestration ... 41
Differences between system and service orchestration ... 43
System orchestration key features ... 43

Main requirements for a service orchestrator **44**

System Orchestrator Architecture **45**

Logical architecture **46**

Conclusion **48**

INTRODUCTION AND SUMMARY

Recent customer projects in the field of architectural cloud computing and IoT frameworks revealed one clear fact repeatedly: Automation remains the utter key to success – not only technically for the framework and/or platform to remain performant, scalable and highly available but also for the business, which is using the respective IT environment, in order to remain in advantage compared to competition as well as stay on the leading edge of innovation.

On top of the automation building block within a cloud framework, an Orchestration solution ensures that atomic automation "black boxes" together form a platform for successful business execution.

As traditional IT landscapes take their leap into adopting (hybrid) cloud solutions for IaaS or maybe PaaS, automation and orchestration – in the same way – has to move from job scheduling or workload automation to more sophisticated IT Ops or DevOps tasks such as:

- Provisioning of infrastructure and applications
- Orchestration and deployment of services
- Data consolidation
- Information collection and reporting
- Systematic forecasting and planning

In a time of constrained budgets, IT must always look to manage resources as efficiently as possible. One of the ways to accomplish that goal is through use of an IT solution that automates mundane tasks, orchestrates the same to larger solution blocks and eventually frees up enough IT resources to focus on driving business success.

This paper "Automation and "Orchestration for Innovative IT-aaS and IoT Architectures" therefore is targeted at

- explaining key criteria for a resilient, secure and scalable automation solution fit for the cloud
- clearly identifying the separation between "automation" and "orchestration"
- providing IT decision makers with a set of criteria to selecting the right solutions for their need of innovation

The first section of the paper will list architectural key criteria for automation solutions and explain their relevance for cloud frameworks as well as innovative IT landscapes in general.

The second section deals with Orchestration. It will differentiate system orchestration from service orchestration, explain key features and provide decision support for choosing an appropriate solution.

KEY FINDINGS

- Traditional “old style” integration capabilities – such as: file transfer, object orientation or audit readiness - remain key criteria even for a cloud-ready automation platform.
- In an era where cloud has become a commodity, just like the internet as such, service centered IT landscapes demand for a maximum of scalability and adaptability as well as multi-tenancy in order to be able to create a service-oriented ecosystem for the advancement of the businesses using it.
- Security, maximum availability, and centralized management and control are fundamental necessities for transforming an IT environment into an integrated service center supporting business expansion, transformation, and growth.
- Service orchestration might be the ultimate goal to achieve for an IT landscape, but system orchestration is a first step towards creating an abstraction layer between basic IT systems and business-oriented IT-services.

TARGET AUDIENCE

Who should be reading this paper:

- Technical decision makers
- Cloud and solution architects in the field of innovative IT environments
- IT-oriented pre- and post-sales consultants

SCALABILITY IN AUTOMATION

Scalability criteria are often referred to as “load scalability”, actually meaning the ability of a solution to automatically adopt to increasing and decreasing consumption or execution demands. While the need for this capability is obviously true for an automation solution, there’s surely more that belongs into the field of scalability. Hence, the following aspects will be discussed within this chapter:

- load scalability
- administrative scalability
- functional scalability
- throughput

LOAD SCALABILITY

Back at a time where cloud computing was still striving for the one and only definition of itself, one key cloud criteria became clear very quickly: Cloud should be defined mainly by virtually infinite resources to be consumed by whatever means. In other words, cloud computing promised to transform the IT environment into a landscape of endlessly scalable services.

Today, whenever IT executives and decision makers consider the deployment of any new IT service, scalability is one of the major requirements. Scalability offers the ability for a distributed system to easily expand and contract its resource pool to accommodate heavier or lighter loads or number of inputs. It also determines the ease with which a system or component can be modified, added, or removed to accommodate changing load.

Most IT decisions today are based on the question of whether a solution can meet this definition. This is why when deciding on an automation solution - especially when considering it to be the automation, orchestration and deployment layer of your future (hybrid) IT framework - scalability should be a high priority requirement. In order to determine a solution’s load scalability capabilities one must examine the system’s architecture and how it handles the following key functions:

EXPANDING SYSTEM RESOURCES

A scalable automation solution architecture allows adding and withdrawing resources seamlessly, on demand, and with no downtime of the core system.

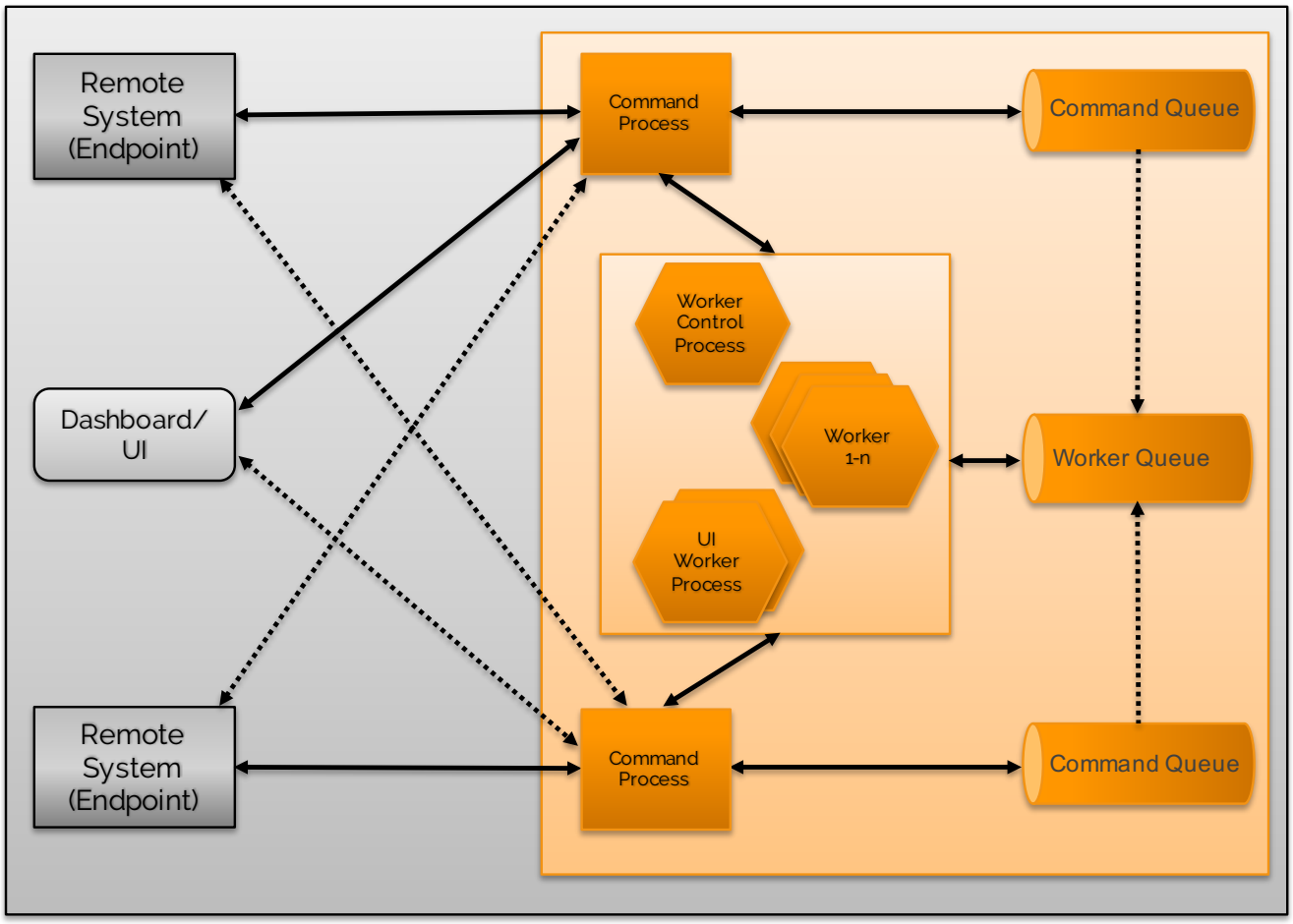
The importance of a central management architecture will be discussed later in this paper; for now it is sufficient to understand that centralized engine architecture develops its main load scalability characteristics through its technical process architecture.

As an inherent characteristic, such an architecture is comprised of thoroughly separated working processes each having the same basic capabilities but - depending on the system's functionality - acting differently (i.e. each serving a different execution function). A worker process could –

depending on technical automation requirements – be assigned to some of the following tasks (see figure below for an overview of such an architecture):

- worker process management (this would be kind-of a "worker control process" capability needed to be built out redundantly in order to allow for seamless handover in case of failure)
- access point for log-on requests to the central engine (one of "Worker 1..n" below)
- requests from a user interface instance ("UI worker process")
- workload automation synchronization ("Worker 1..n")
- Reporting ("Worker 1..n")
- Integrated authentication with remote identity directories ("Worker 1..n")

At the same time, command and communication handling should be technically separated from automation execution handling and be spread across multiple "Command Processes" as well - all acting on and providing the same capabilities. This will keep the core system responsive and scalable in case of additional load.



The architecture described in the figure above represents these core differentiators when it comes to one of the following load change scenarios:

DYNAMICALLY ADDING PHYSICAL/VIRTUAL SERVERS TO ALTER CPU, MEMORY AND DISK SPACE OR INCREASING STORAGE FOR WORKLOAD PEAKS WITHOUT DOWNTIME

With the above architecture, changing load means simply adding or withdrawing physical (virtualized/cloud-based) resources to the infrastructure running the core automation system. With processes acting redundantly on the "separation of concern" principle, it is either possible to provide more resources to run other jobs or add jobs to the core engine (even when physically running on a distributed infrastructure).

This should take place without downtime of the core automation system, ensuring not only rapid reaction to load changes but also high resource availability (to be discussed in a later chapter).

CHANGE THE NUMBER OF CONCURRENTLY CONNECTED ENDPOINTS TO ONE INSTANCE OF THE SYSTEM

At any time during system uptime it might become necessary to handle load requirements by increasing the number of system automation endpoints (such as agents) connected to one specific job instance. This is possible only if concurrently acting processes are made aware of changing endpoint connections and are able to re-distribute load among other running jobs seamlessly without downtime. The architecture described above allows for such scenarios where a separated, less integrated core engine would demand reconfiguration when adding endpoints over a certain number.

ENDPOINT RECONNECTION FOLLOWING AN OUTAGE

Even if the solution meets the criteria of maximum availability, outages may occur. A load scalable architecture is a key consideration when it comes to disaster recovery. This involves the concurrent boot-up of a significant number of remote systems including their respective automation endpoints. The automation solution therefore must allow for concurrent re-connection of several thousand automation endpoints within minutes of an outage in order to resume normal operations.

ADMINISTRATIVE SCALABILITY

While load scalability is the most commonly discussed topic when it comes to key IT decisions, there are other scalability criteria to be considered as differentiating criteria in deciding on an appropriate automation solution. One is "Administrative Scalability" defined as the ability for an increasing number of organizations or users to easily share a single distributed system.[1]

[1] Sources: Wikipedia - Scalability and Paper “On System Scalability”, Weinstock/Goodenough, March 2006

ORGANIZATIONAL UNIT SUPPORT

A system is considered administratively scalable when it is capable of:

- Logically separating organizational units within a single system. This capability is generally understood as "multi-client" or “multi-tenancy”.
- Providing one central administration interface (UI + API) for system maintenance and onboarding of new organizations and/or users.

ENDPOINT CONNECTION FROM DIFFERENT NETWORK SEGMENTS

Another aspect of administrative scalability is the ability of an automation solution to seamlessly connect endpoints from various organizational sources.

In large enterprises, multiple clients (customers) might be organizationally and network-wise separated. Organizational units are well hidden from each other or are routed through gateways when needing to connect. However, the automation solution is normally part of the core IT system serving multiple or all of these clients. Hence, it must allow for connectivity between processes and endpoints across the aforementioned separated sources. The established secure client network delineation must be kept in place, of course. One approach for the automation solution is to provide special dedicated routing (end)points capable of bridging the network separation via standard gateways and routers but only supporting the automation solution’s connectivity and protocol needs.

SEAMLESS AUTOMATION EXPANSION FOR NEWLY ADDED RESOURCES

While the previously mentioned selection criteria for automation systems are based on "segregation," another key decision criteria is based on harmonization and standardization.

An automation system can be considered administratively scalable when it is capable of executing the same, one-time-defined automation process on different endpoints within segregated environments.

The solution must be able to:

- Add an organization and its users and systems seamlessly from any segregated network source.
- Provide a dedicated management UI including those capabilities which is solely accessible securely by organization admin users only.

and at the same time

- Define the core basic automation process only once and distribute it to new endpoints based on (organization-based) filter criteria.

The architecture thereby allows for unified process execution (implement once, serve all), administrative scalability and efficient automation.

FUNCTIONAL SCALABILITY

Functional Scalability, defined as the ability to enhance the system by adding new functionality at minimal effort[2], is another type of scalability characteristics that shall be included in the decision-making process.

The following are key components of functional scalability:

ENHANCE FUNCTIONALITY THROUGH CUSTOMIZED SOLUTIONS

Once the basic processes are automated, IT operations staff can add significant value to the business by incorporating other dedicated IT systems into the automation landscape. Solution architects are faced with a multitude of different applications, services, interfaces, and integration demands that can benefit from automation.

A functionally scalable automation solution supports these scenarios out-of-the-box with the ability to:

- Introduce new business logic to existing automation workflows or activities by means of simple and easy-to-adopt mechanisms without impacting existing automation or target system functions.
- Allow for creation of interfaces to core automation workflows (through use of well-structured APIs) in order to ease integration with external applications.
- Add and use parameters and/or conditional programming/scripting to adapt the behavior of existing base automation functions without changing the function itself.

TEMPLATE-BASED IMPLEMENTATION AND TEMPLATE TRANSFER

A functionally scalable architecture also enables the use of templates for implementing functionality and sharing/distributing it accordingly.

Once templates have been established, the automation solution should provide for a way to transfer these templates between systems or clients. This could either be supported through scripting or solution packaging. Additional value-add (if available): Share readily-tested implementations within an automation community.

Typical use-cases include but are not limited to:

- Development-test-production deployment of automation packages.
- Multi-client scenarios with well-segregated client areas with similar baseline functionality.

[2] Sources: Wikipedia - Scalability or Computer Business Research: Scalability

CUSTOMIZABLE TEMPLATE LIBRARIES WITH GRAPHICAL MODELER

In today's object and service based IT landscapes, products that rely on scripting are simply not considered functionally scalable. When using parameterization, conditional programming, and/or object reuse through scripting only, scaling to augment the automation implementation would become time-consuming, complex, and unsustainable. Today's functionally scalable solutions use graphical modelers to create the object instances, workflows, templates, and packages that enable business efficiency and rapid adaptation to changing business requirements.

THROUGHPUT

Finally, consider the following question as a decision basis for selecting a highly innovative, cloud-ready, scalable automation solution:

What is the minimum and maximum workflow load for execution without architectural change of the solution? If the answer is: 100 up to 4-5 Mio concurrent jobs per day without change of principle setup or architecture, one's good to go.

In other words: Scalable automation architectures support not only the aforementioned key scalability criteria but are also able to handle a relatively small scale of concurrent flows equally to an utterly large scale. The infrastructure footprint needed for the deployed automation solution must obviously adapt accordingly.

HIGH AVAILABILITY AND ROBUSTNESS

High Availability and Robustness are critical elements of any automation solution. Since an automation solution forms the backbone of an entire IT infrastructure, one needs to avoid any type of downtime. In the rare case of a system failure, rapid recovery to resume operations shall be ensured.

Several architectural patterns can help achieve those goals.

MULTI-SERVER / MULTI-PROCESS ARCHITECTURE

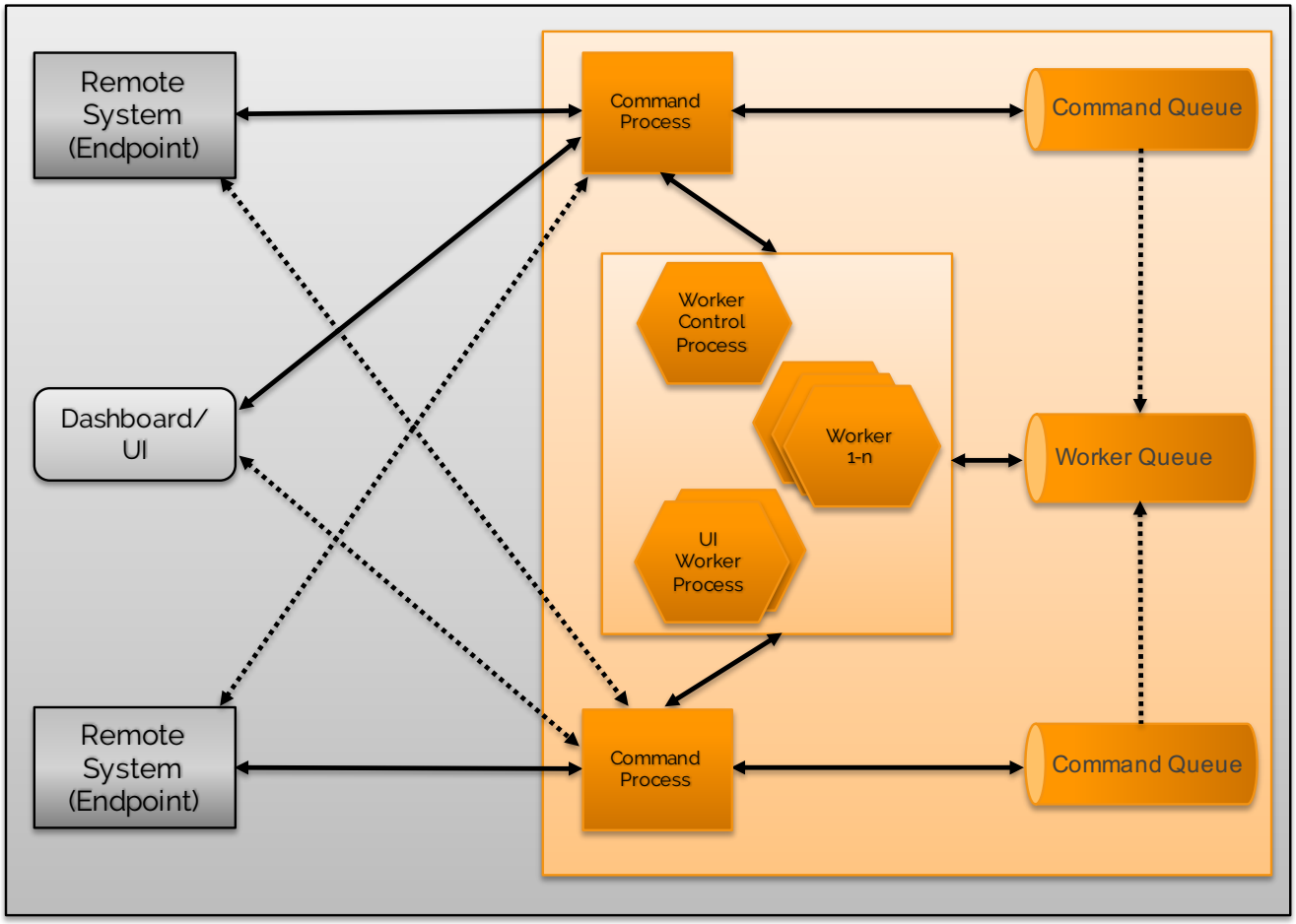
Assuming, that the automation solution of choice operates a centralized management layer as core component, the intelligence of the automation implementation is located in a central backbone and distributes execution to "dumb" endpoints. Advantages of such an architecture will be discussed in subsequent chapters. The current chapter focusses on the internal architecture of a “central engine" in more detail:

Robustness and near 100% availability is normally achieved through redundancy which comes by the trade-off of a larger infrastructure footprint and resource consumption. However, it is inherently crucial to base the central automation management layer on multiple servers and multiple processes. Not all of these processes necessarily act on a redundant basis, as would be the case with round-robin load balancing setups where multiple processes can all act on the same execution. However, in order to mitigate the risk of a complete failure of one particular function, the different processes distributed over various physical or virtual nodes need to be able to takeover operation of any running process.

This architectural approach also enables scalability which has been addressed previously. Most of all, however, this type of setup best supports the goal of 100% availability. At any given time, workload can be spread across multiple physical/virtual servers as well as split into different processes within the same (or different) servers.

TRANSACTIONS

Another aspect of a multi-process architecture that helps achieve zero-downtime non-stop operations is that it ensures restoration of an execution point at any given time. Assuming, that all transactional data is stored in a persistent queue, the transaction is automatically recovered by the remaining processes on the same or different node(s) in case of failure of a physical server, a single execution, or a process node. This prevents any interruption, loss of data, or impact to end-users.



Purposely, the figure above is the same as in the "Scalability" chapter, thereby showing how a multi-server / multi-process automation architecture can support both scalability and high availability.

ENDPOINT FAILOVER

In a multi-server/multi-process architecture, recovery can't occur if the endpoint adapters aren't capable of instantly reconnecting to a changed server setup. To avoid downtime, all decentralized components such as control and management interfaces, UI, and adapters must support automatic failover. In the case of a server or process outage these interfaces must immediately reconnect to the remaining processes.

In a reliable failover architecture, endpoints need to be in tune with the core engine setup at all times and must receive regular updates about available processes and their current load. This ensures that endpoints connect to central components based on load data, thereby actively supporting the execution of load balancing. This data can also be used to balance the re-connection load efficiently in case of failure/restart.

DISASTER RECOVERY

Even in an optimum availability setup, the potential of an IT disaster still exists. For this reason, organizations should be prepared with a comprehensive business continuity plan. In automation architectures, this does not necessarily mean rapid reboot of central components – although this could be achieved depending on how lightweight the automation solution central component architecture is constructed. More importantly, however, is the ability of rebooted server components to allow for swift re-connection of remote automation components such as endpoint adapters and/or management UIs. These concepts were discussed in depth in the scalability chapter.

KEY PERFORMANCE INDICATORS

Lightweight, robust architectures as described above have two main advantages beyond scalability and availability:

- They allow for small scale setups at a low total cost of ownership (TCO).
- They allow for large scale setups without entirely changing existing systems.

One of the first features of an automation system as described above is a low TCO for a small scale implementation.

Additional indicators to look for include:

- Number of customers served at the same time without change of the automation architecture
- Number of tasks per day the automation platform is capable of executing
- Number of endpoints connected to the central platform

In particular, the number of endpoints provides clarity about strengths of a high-grade, enterprise-ready, robust automation solution.

EFFICIENCY THROUGH OBJECT ORIENTATION

A software architecture pattern for an IT oriented automation solution? How would this work together? Let's have a closer look:

SOLID OBJECT ORIENTATION

Software guru Robert C. Martin identified "the first five principals" of object-oriented design in the early 2000's. Michael Feathers introduced the acronym SOLID to easily remember these five basics that developers and architects should follow to ensure they are creating systems that are easy to maintain and to extend over time.[3]

- **S**ingle responsibility: any given object shall have exactly one responsibility or one reason to change.
- **O**pen-close: any given object shall be closed for modification but open for extension.
- **L**iskov substitution principle: any given live instance of a given object shall be replaceable with instances of the objects subtypes without altering the correctness of the program.
- **I**nterface segregation: every interface shall have a clear and encapsulated purpose; interface consumers must be able to focus on the interface needed and not be forced to implement interfaces they do not need.
- **D**ependency inversion: create any dependency to an object's abstraction not to its actual presence

The SOLID principle and other object-oriented patterns, are discussed further in this article: Object-Oriented-Design-Principles

"SOLID" IT AUTOMATION

Many of today's enterprise workload automation solutions were developed with architectural patterns in mind, which date back well before the advance of object orientation. In order to avoid the risk of immaturity of the solution, patterns weren't innovated in some cases. At the same time, the demand for rapid change, target-dependent concretion and re-usability of implementations has been increasing. An object oriented approach can now be used as a means to support these new requirements.

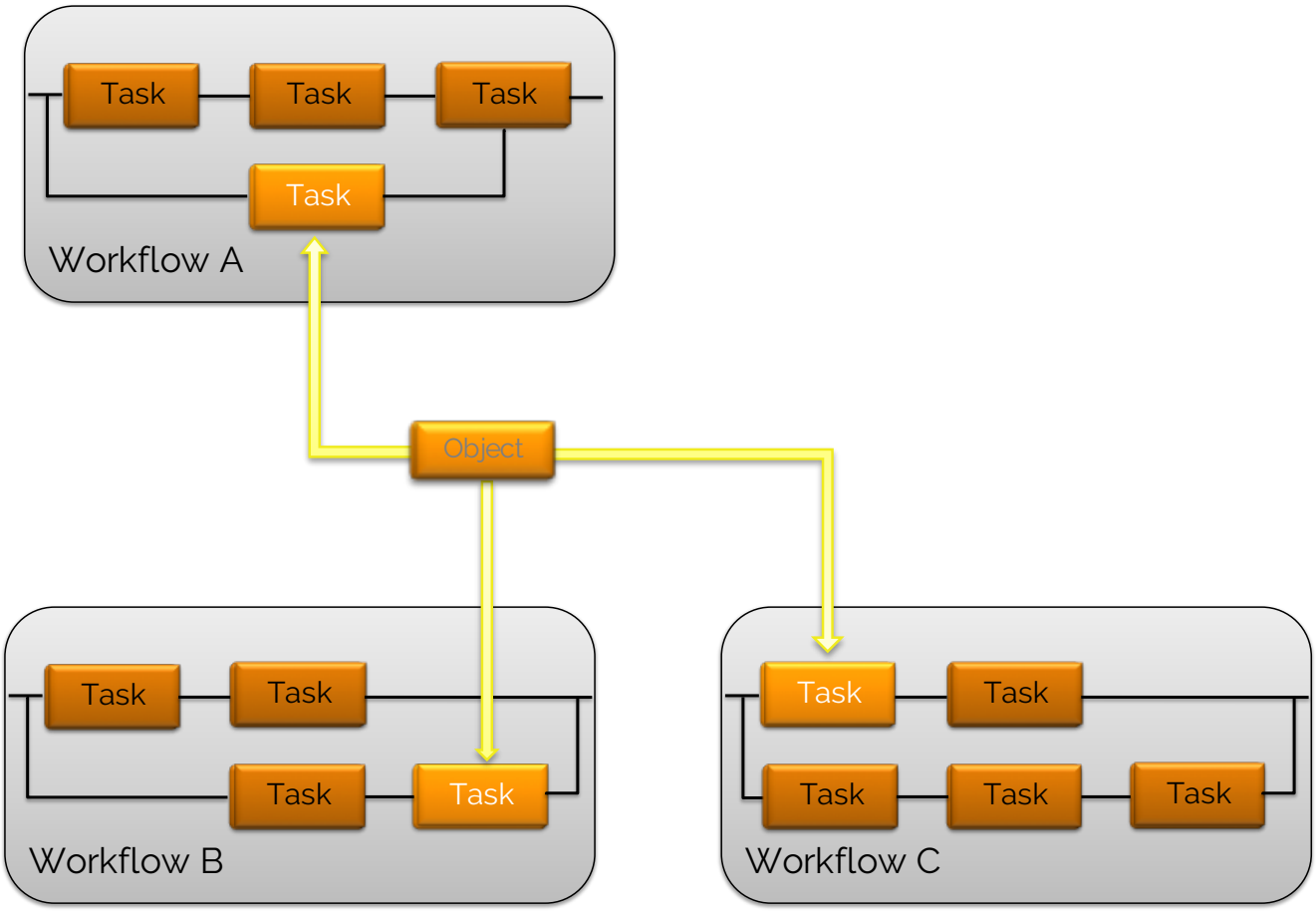
Object orientation, therefore, should be one of the key architectural patterns of an innovative enterprise automation solution. Such a solution encapsulates data and logic within automation objects and thereby represent what could be called an "automation blueprint." The

[3] Source: http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod and Wikipedia - "SOLID" Object Oriented Design

object presents a well-defined "input interface" through which a runtime-instance of the automation object can be augmented according to the requirements of the specific scenario.

Through interaction with automation objects, an object-oriented workflow can be defined, thus presenting an aggregated automation object. By employing the patterns of object-oriented architecture and design, developers ensure that the implementation of automation scenarios evolves into object interaction, re-usability, specialization of abstracted out-of-the-box use-cases, and resolves specific business problems in a dedicated and efficient way.

Enterprise-grade, innovative automation solutions define all automation instructions as different types of objects within any kind of object repository – similar to traditional object oriented programming languages. Basic definition of automation tasks is represented as “automation blueprints”. Through instantiation, aggregation and/or specialization, the static object library becomes the arbitrary, dedicated, business process oriented solution to execute and perform business automation.



The figure above shows an example of how an object can be instantiated as a dedicated task within different workflows.

IT-AAS EXAMPLE

The following IT-aaS example illustrates an object-oriented automation pattern. The chosen scenario assumes that an IT provider intends to offer an IT service to request, approve, and automatically provision a new, compliant infrastructure service such as a web application server.

- Object definition: A set of aggregated objects - automation blueprints - define how to provision a new server within a certain environment. The object definition would not - however - bind its capability to a specific hypervisor or public cloud provider.
- Object instantiation: Request logic - realized for example in a service desk application - would implement blueprint instantiation including the querying (e.g. by user input or by retrieval from a CMDB) and forwarding of the parameters to the objects' instance.

This not only automates the service provisioning but also addresses burst-out scenarios required by modern IT service delivery through integrated automation.

PATTERNS BENEFITTING FROM OBJECT REUSABILITY

The concept of object orientation allows the reuse of automation objects eliminating the need to duplicate information. It also allows the creation of automation blueprints describing the automation logic that is processed at runtime. An automation blueprint can behave differently once it gets instantiated during runtime because of features like integrated scripting, variables, pre- and post conditional logic, and logical programming elements such as conditions and loop constructs.

INHERITANCE

Automation objects' relationships and dependencies as well as input/output data are set at the time they are defined. Run-time instantiated objects can inherit their input parameters from parent objects. They can also pass parameters from one runtime instance to another as well as to their parent containers. This enables fully flexible multi-processing of business workflows without e.g. being forced to clean-up variables in static containers.

ABSTRACTION VERSUS SPECIALIZATION

Automation blueprint definitions form the basic abstraction of scenarios within an enterprise-grade automation platform. Abstract automation objects get their concrete execution information when instantiated. Object oriented platforms provide the means to augment the instantiated objects at runtime via patterns like prompt sets, database variables or condition sets – in the best case to be modeled graphically; this supports flexibility and dynamic execution.

MAINTAINABILITY

As the concept of object orientation eliminates the need for duplication of automation logic, maintaining automation workflow definitions becomes minor. Typical modifications such as

changes of technical user-ids, path/binary name etc. can be performed in a centrally defined object and is applied wherever the object (blueprint) is used.

ADAPTABILITY & EXTENSIBILITY

Today's Automation solutions normally are ready-built enterprise products (or a consumable service) offering out-of-the-box functionality for multiple automation, orchestration, and integration scenarios. On top of this, ease of installation, implementation, and use would be of importance.

However, in less than 20% of cases, the automation platform remains unchanged for the first six months. This is why it's crucial that from the beginning, the selected solution has the ability to extend and adapt in order to serve business needs. The architecture should be able to leverage technologies for hot plugging integrations and industry-standard interfaces while augmenting standard functionality through dynamic data exchange.

HOT PLUG-IN OF NEW INTEGRATIONS

Once basic automation workflows for an IT landscape are implemented, avoiding downtime is critical. While some automation solutions may offer fast and flexible production updates, the expectation on top of that is to be able to integrate new system and application adapters on the fly.

The first step to this level of integration can be achieved by rigorously complying with the SOLID object orientation principles discussed in the last chapter. Integrating adapters to new system or application endpoints, infrastructure management layers (like hypervisors), or middleware components is then a matter of adding new objects to the automation library. Existing workloads can be seamlessly delegated to these new entities, avoiding the need to stop any runtime entities or updating or upgrading any of the system's components.

Hot-plugging, however, isn't the main factor in assessing an automation solution's enterprise readiness. In addition to being able to plug new adapters into the landscape, a truly expandable automation solution must be able to build new adapters as well. The automation solution should offer a framework, which enables system architects and software developers to create their own integration solutions based on the patterns the automation solution encompasses.

Such a framework allows for the creation of integration logic based on existing objects and interfaces, self-defined user interface elements specific to the solution's out-of-the-box templates. Extensions to such a framework include a package-manager enabling third party solutions to be deployed in a user-friendly way. It must also take into account any dependencies and solution versions and a framework IDE enabling developers to leverage the development environment to which they are accustomed (e.g. Eclipse and Eclipse plugins).

Herewith, plugging new integrations into an automation solution can expand the automation platform by leveraging a community-based ecosystem of 3rd party extensions.

LEVERAGE INDUSTRY STANDARDS

Hot plugging all-new automation integration packages to adapt and expand your automation platform might not always be the strategy of choice.

In a service-based IT-architecture, many applications and management layers can be integrated using APIs. This means that an automation solution needs to leverage standards to interface with external resources prior to forcing you into development effort for add-ons.

The automation layer needs to have the ability to integrate remote functionality through common shell-script extensions like PowerShell (for Microsoft-based landscapes), JMS (Java Message Service API for middleware integration), REST (based on standard data formats like XML and JSON), and maybe (with decreasing importance) SOAP.

DYNAMIC DATA VALIDATION & EXCHANGE

Part of the adaptability and extensibility requirement is for the product of choice to be able to process and integrate the results of the resources previously discussed into existing object instances (as dynamic input data to existing workflows) without having to develop customized wrappers or additional interpreters.

This can be achieved through either variable objects – their values being changeable through integration like DB queries or Web Services results – or through prompts that allow to set variable values through a user input query.

To be truly extensible and adaptable, an automation solution should not only offer manual prompts but it should be able to automatically integrate and present those prompts within external systems. The solution should be responsible for automating and orchestrating IT resources while other systems - a service catalogue or request management application - handles IT resource demand and approval.

Together, all of the above forms the framework of an automation solution that can be extended and adapted specifically to the needs of the business leveraging it.

INTEGRATED FILE TRANSFER

There is a wide range of sophistication when it comes to the IT systems that businesses operate. Some came on line around 2000, but others have been in use for much longer. Some have constructed systems in order to maintain high quality services for years to come. Still others are constantly adapting their systems to take advantage of the latest technology.

Because of this wide disparity, an automation solution must be able to handle current and future innovations in integration, orchestration and performance. It must also be backwards compatible so it can support legacy technologies.

Saying this, one of the technologies that an automation solution must support is file transfer between systems. Along with this, it must also support elaboration, interpretation, and transformation of file content to create new levels of automation integration for enterprise IT.

Experiences with multiple customers show that replacing legacy file transfer applications with state-of-the-art APIs is sometimes simply too time consuming and costly. However, it is crucial that these legacy system capabilities are provided for an automated and integrated IT landscape. Strangely enough, therefore, being able to address, process, interpret, and transfer files with the demands and challenges of an automated IT environment is still a must-have criteria for an enterprise automation solution.

SUPPORT OF MULTIPLE DIFFERENT FILE TRANSFER PROTOCOLS

FTP[4] not equals FTP: Following are the most common FTPs still in use which must be supported by the selected automation solution:

- FTP: This is the standard protocol definition for transferring files in an insecure manner. When operating behind a firewall, using FTP for transporting files is convenient and needs to be an integrated feature of your enterprise automation solution.
- FTPS: adds support for "Transport Layer Security" (TLS) and the "Secure Socket Layer" (SSL) encryption protocols to FTP. Many enterprises rely on this type of protocol for security reasons, especially when it comes to moving beyond the network.
- FTPES: This differs from FTPS only in terms of the timing of the encryption and transferring of login information. It adds an additional safety control to FTPS-based file transfers
- SFTP: has been added to the Secure Shell protocol (SSH) by the Internet Engineering Task Force (IETF)[5] in order to allow for access, transfer and management of files through any reliable (SSH) data stream.

[4] FTP: file transfer protocol - see a list of different file transfer protocols from Wikipedia

[5] IETF: Internet Engineering Taskforce - https://www.ietf.org/

In addition to supporting all of the above protocols, an automation solution can enhance FT integration in automation scenarios by offering a direct endpoint-to-endpoint file transfer - based on a proprietary protocol. Providing this protocol eases the need for a central management engine implementation solely to transport files from one system to another.

STANDARD FT PROTOCOLS

The most convenient way to allow connecting FTP capable remote systems based on the protocols listed above is through a graphical UI that allows defining the transfer much the way it is done with standard FTP clients. The actual transfer itself is normally executed by means of a dedicated adapter only initiated by the centrally managed and executed automation workflows. To comply with security requirements limiting login information to only top-level administrators, sensitive information such as username, password, or certificates are stored in separate objects. At the same time, file transfers are integrated into automation flows by specialists who do not have access to the detailed login information but can still make use of the prepared security objects.

ENDPOINT-TO-ENDPOINT FILE TRANSFER

In addition to supporting the standard FTP protocols, the automation solution's ecosystem should offer a direct secure file transfer between two endpoints within the IT landscape.

In this case the automation solution issues the establishment of a direct, encrypted connection between the affected endpoints - normally using a proprietary internal protocol. This type of mechanism eliminates the need for additional tools and increases the performance of file transfers significantly.

Other features the solution should support include:

- Multiple code translation (e.g. from ASCII to EBCDIC)
- Data compression
- Wildcard transfer
- Regular checkpoint log (in order to re-issue aborted transfers from the last checkpoint recorded)
- Checksum verification based on hashing algorithms (like e.g. MD5)
- Industry standard transfer encryption (e.g. AES-128, AES-192 or AES-256)

PROCESS INTEGRATION

Finally, the key to offering enterprise ready integrated file transfer through any protocol is to allow seamless integration into existing automation workflows while leveraging all the automation functionality without additional re-coding or re-interpretation of transferred files. This includes:

- Using file transfer results and file content in other automation objects.

- Including file transfer invocation, execution, and result processing in the scripting environment.
- Using files within pre or post conditions of action or workflow execution or augmenting pre/post conditions by making use of results from transferred files.
- Bundling file transfers to groups of endpoints executing similar - but not necessarily identical - file transfer processes.

This allows the integration of legacy file transfers into innovative business processes without losing transparency and flexibility.

AUDIT & COMPLIANCE

Audit and Compliance has assumed greater importance in recent years. Following the Global Financial Crisis of 2007-08[6] – one of the most treacherous crises of our industrial age - audit and standardization organizations as well as governmental institutions invested heavily into strengthening compliance laws, regulations, and enforcement.

This required enterprises in all industries to make significant investments to comply with these new regulations. Standards have evolved that define necessary policies and controls to be applied as well as requirements and procedures to audit, check, and enhance processes.

Typically, these policies encompass both business and IT related activities such as authentication, authorization, and access to systems. Emphasis is placed on tracking modifications to any IT systems or components through use of timestamps and other verification methods – particular focused on processes and communications that involve financial transactions.

Therefore, supporting the enforcement and reporting of these requirements, policies and regulations must be a core function of the automation solution. Following are the key factors to consider when it comes to automation and its impact on audit and compliance.

TRACEABILITY

The most important feature of an automation solution to meet compliance standards is traceability. The solution must allow for logging capabilities that tracks user activity within the system. It must provide tracking of all modifications to the system's repository and include the user's name, date and time of the change, and a copy of the data before and after the change was made. Such a feature ensures system integrity and compliance with regulatory statutes.

STATISTICS

Statistical records are a feature that ensures recording of any step performed either by an actual user or one initiated by an external interface (API). Such records should be stored in a hierarchy within the system's backend database allowing follow up checking as to who performed what action at what specific time. Additionally, the system should allow for comments on single or multiple statistical records, thereby supporting complete traceability of automation activities by documenting additional operator actions.

[6] Reference links to various sources can be found on Wikipedia: https://en.wikipedia.org/wiki/Financial_crisis_of_2007%E2%80%9308

VERSION MANAGEMENT

Some automation solutions offer the option of integrated version management. Once enabled, the solution keeps track of all changes made to tasks and blueprint definitions as well as to objects like calendars, time zones etc. Every change creates a new version of the specific object which can be accessible at any time for follow up investigation. Objects include additional information like version numbers, change dates and user identification. In some cases, the system allows for restoring an older version of the specific objects.

MONITORING

All of the above handle, process and record design-time activity of an automation system, ensuring stored data and data changes are documented to comply with audit needs. During execution, an automation system should also be able to monitor the behavior of every instantiated blueprint. Monitoring records need to track the instance itself as well as every input/output, changes performed to or by this instance (e.g. putting a certain task on hold manually).

FULL AUDIT TRAILS

All of the above features contribute to a complete audit trail that complies with the reporting requirements as defined by the various standards. Ultimately an automation system must be able to easily produce an audit trail of all system activity from the central database in order to document specific actions being investigated by the auditor. An additional level of security that also enables compliance with law and regulations is the system's ability to restrict access to this data on a user/group basis.

COMPLIANCE THROUGH STANDARDIZATION

Finally, to ease compliance adherence, the automation solution must follow common industry standards. While proprietary approaches within a system's architecture are applicable and necessary (e.g. scripting language - see chapter "Dynamic Processing Control"), the automation solution itself must strictly follow encryption methods, communication protocols, and authentication technologies that are widely considered as common industry best practice. Any other approach in these areas would significantly complicate the efforts of IT Operations to prove compliance with audit and regulatory standards. In certain cases, it could even increase the audit cycle to less than a year depending on the financial and IT control standard being followed.

INDEPENDENT TENANTS

Another decision one needs to make during the selection process is whether the automation platform needs to support multi tenant and/or multi client capability. How you choose can have a significant financial impact.

MULTI TENANCY VERSUS MULTI-CLIENT

Multi tenancy is closely related to the Cloud. Although not strictly a cloud pattern, multi tenancy has become one of the most discussed topics in application and service architectures due to the rise of Cloud Computing. That is, because multi tenancy eventually enables one of the essential cloud characteristics, namely: virtually endless scaling and resource pooling.

MULTI-TENANCY

Multi tenancy partitions an application into virtual units. Each virtual unit serves one customer while being executed within a shared environment together with the other tenants' units. It doesn't interact or conflict with other virtual units nor can a single virtual unit remove all resources from the shared environment in case of malfunction (resource pooling, resource sharing).

MULTI-CLIENT

In contrast, a multi-client system is able to split an application into logical environments by separating functionality, management, object storage, and permission layers. This enables setting up a server that allows logons by different users with each user having their separate working environment while sharing common resources - file system, CPU, memory. However, in this environment, there remains the possibility of users impacting each other's work.

IMPORTANCE OF MULTI TENANCY AND MULTI CLIENT

These concepts are critical because of the need to provide separated working environments, object stores, automation flows for different customers or business lines. Therefore, one should be looking for an automation solution which supports this capability out-of-the-box. In certain circumstances you may not require strict customer segregation or the ability to offer pooling and sharing of resources out of one single environment. This clear differentiation might become a cost-influencing factor in certain cases.

INDEPENDENT UNITS WITHIN ONE SYSTEM

Whether your automation solution needs to be multi-tenant or not depends on the business case and usage scenario. Normally in enterprise environments having major systems running on-premises, multi-tenancy is not a major requirement in an automation solution. Experience shows that when automation systems are shared between multiple organizational units or are

automating multiple customers' IT landscapes in an outsourcing scenario, multi-tenancy isn't required since management of all units and customers is controlled through the central administration and architecture.

Multi-client capabilities, though, are indeed a necessity in an enterprise ready automation solution, as users of multiple different organizations want to work within the automation environment.

Multi-client capabilities would include the ability to:

- Split a single automation solution instance into up to 1,000++ different logical units (clients)
- Add clients on demand without downtime or without changing underlying infrastructure
- Segregate object permission by client and enable user assignment to clients
- Segregate automation objects and enable assignment to specific clients
- Allow for automation execution delegation of centrally implemented automation workflows by simply assigning them to the specific clients (assuming the specific permissions have been set)
- Re-use automation artefacts between clients (including clear and easy to use permission management)
- Share use of resources across clients (but not necessarily for secure and scalable resource pooling across clients; see differentiation above)

SEGREGATION OF DUTIES

Having multiple clients within one automation solution instance enables servicing of multiple external as well as internal customers. This allows for quick adaptation to changing business needs. Each client can define separate automation templates, security regulations, and access to surrounding infrastructure. Having a simple transport/delegation mechanism between clients at hand, allows to implement a multi-staging concept for the automation solution.

SECURITY

An obvious key point to consider when choosing an automation solution is security. We've discussed Audit & Compliance separately from security since audit trails and compliance need the architectural support by the solution but are both less technical in itself compared to security.

Considering security issues for an automation solution means focusing on the following areas:

- Confidentiality: How does the solution manage authorized access?
- Integrity: How does the solution ensure that stored objects and data are consistently traceable at any point in time?
- Availability: How does the solution guarantee availability as defined, communicated, and agreed upon?
- Authenticity: How does the solution ensure the authenticity of identities used for communication of partners (components, objects, users)
- Liability: How does the solution support responsibility and accountability of the organization and its managers?

None of these areas rely on one particular architectural structure. Rather they have to be assessed by reviewing the particular solution's overall architecture and how it relates to security.

USER SECURITY

AUTHENTICATION

Any reputable automation solution will offer industry standard authentication mechanisms such as password encryption, strong password policy, and login protection upon fail. Integrating with common identity directories such as LDAP or AD provides a higher level of security for authenticating user's access. This allows for the "bind request" to be forwarded to the specific directory and thereby leveraging the directory's technologies not only to protect passwords and users but also to provide audit trail data for login attempts. Going a step further, an authentication system provided through an external, integrated LDAP might offer stronger authentication - such as MFA - out-of-the-box without the need to augment the solution to gain greater security.

In addition, the solution should provide a customized interface (e.g. provided through an "exit - callback" mechanism) for customers to integrate any authentication mechanism that is not yet supported by the product out-of-the-box.

PERSONNEL DATA BASE

Most organizations use one core personnel database within their master data management (MDM) process. For example, new employees are onboarded through an HR-triggered process which, in addition to organizational policies, ensures creation of access permissions to systems that employees use every day. As part of an automation system's architecture, such an approach

involves the need to offer automatically available interfaces and synchronization methods for users – either as objects or links. The automation workflow itself, which supports the HR onboarding process, would subsequently leverage these interfaces to create necessary authentication and authorization artefacts.

AUTHORIZATION & ACCESS

Enterprise-grade automation solutions should offer a variety of access control for managed objects. In addition to the core capabilities already discussed, IT operations should expect the solution's support for securing various layers and objects within it. This involves:

- Function level authorization: The ability to grant/revoke permission for certain functions of the solution.
- Object level authorization: The ability to create access control lists (ACLs) at the single object level if necessary.
- ACL aggregation: The ability to group object level ACLs together through intelligent filter criteria in order to reduce effort for security maintenance.
- User grouping: The ability to aggregate users into groups for easy management.

In addition, a secure solution should protect user and group management from unauthorized manipulation through use of permission sets within the authorization system.

API

Automation solutions that do not include APIs are rarely enterprise ready. While compatible APIs (e.g. based on java libraries) would inherently be able to leverage previously discussed security features, Web Service APIs need to offer additional authentication technologies along commonly accepted standards. Within REST, we mainly see three different authentication methods:

1. Basic authentication is the lowest security option as it involves simply exchanging a base64 encoded username/password. This not only requires additional security measures for storing, transporting, and processing login information, but it also fails to support authenticating against the API. It also opens external access for any authorized users through passwords only.
2. OAuth 1.0a provides the highest level of security since sensitive data is never transmitted. However, implementation of authentication validation can be complex requiring significant effort to set up specific hash algorithms to be applied with a series of strict steps.
3. OAuth 2.0 is a simpler implementation, but still considered a sufficiently secure industry standard for API authentication. It eliminates use of signatures and handles all encryption through transport level security (TLS) which simplifies integration.

Basic authentication might be acceptable for an automation solution's APIs being operated solely within the boundaries of the organization. This is becoming less common as more IT operations evolve into service oriented, orchestrated delivery of business processes operating in a hybrid environment. Operating in such a landscape requires using interfaces for external integration, in which case your automation solution must provide a minimum of OAuth 2.0 security.

OBJECT LEVEL SECURITY

The levels of authorization previously mentioned set the stage for defining a detailed authorization matrix within the automation solution's object management layer. An object represents an execution endpoint within a highly critical target system of automated IT operation. Accessing the object representing the endpoint grants permission for the automation solution to directly impact the target system's behavior. Therefore, an automation system must provide sufficiently detailed ACL configuration methods to control access to:

- Endpoint adapters/agents
- Execution artefacts such as processes and workflows
- Other objects like statistics, reports, and catalogues
- Logical tenants/clients

The list could be extended even further. However, the more detailed the authorization system, the greater the need for feasible aggregation and grouping mechanisms to ease complexity. At the same time, the higher the number of possibilities for controlling and managing authorization, the better the automation solution's managability.

SEPARATION OF CONCERN

Finally, to allow for a role model implementation that supports a typical IT organizational structure, execution must be separated from design and implementation. Object usage must not automatically imply permission for object definition. This allows another automation specialist to access the system to construct workflows with this and other objects without revealing the underlying credentials.

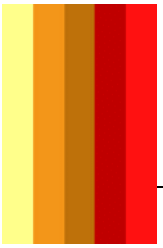
COMMUNICATION SECURITY

Securing the communication between systems, objects, and endpoints is the final security issue to be considered when assessing an automation solution. This includes

- Encryption
- Remote endpoint authentication - the ability to allow configuration of target endpoints authentication when interacting with the core automation management engine

For communication between components, encryption must be able to leverage standard algorithms. The solution should also allow configuration of the desired encryption method. At minimum, it should support AES-256.

Endpoint authentication provides a view of security from the opposite side of automation. To this point, we've discussed how the solution should support security implementation. When a solution is rolled out, however, endpoints need to automatically and securely interact with the automation core. Ideally the automation solution should generate a certification key deployable as a package to endpoint installations. Ideally this would happen via a separate, secure connection. This



configuration enables a unique fingerprint for each endpoint and avoids intrusion of untrusted endpoints into the automation infrastructure.

HOMOGENEOUS END-TO-END INTEGRATION

Whether looking to extend existing IT service capabilities through innovative service orchestration and delivery, or trying to increase the level of automation within the environment, one would always want to examine the following core features of a prospective automation solution:

- Automation
- Orchestration
- Provisioning
- Service definition and catalogue
- Onboarding and subscription
- Monitoring and metering
- Reporting and billing

Though not all of those might be utilized at once, the automation solution will definitely play a major role in aggregating them to support the business processes of an entire enterprise.

Either way, homogeneity represents a key element, when it comes to determining the right solution, with the right approach and the right capabilities.

ONE UX FOR ALL INTEGRATIONS

First, the automation platform one choses must have a unified user experience (UX) for all targeted applications. This doesn't mean that for every component in the system the exact same user interface needs to be presented. It's more important that there is a unified pattern for all the components. This should start with the central management elements of the solution and extend to both internal and external resources such as an Automation Framework IDE for 3rd party solutions discussed previously.

In addition, the core automation components also must match the same UX. Introducing an automation system with standard user interfaces and integration concepts ensures rapid implementation, since SME's can focus on automating system processes rather than being bogged down with training on the automation solution itself.

ONE PLATFORM FOR ALL AUTOMATION

The more products that make up the automation solution, the greater the effort required to integrate them all into an IT landscape. Software and system architecture throughout history has never proposed one single technology, standard or design guideline for all functional capabilities, non-functional requirements, or interface definitions. Therefore, a bundled system comprised of multiple products will in 95% of cases come with a variety of inter-component interfaces that will need to be configured separately from the centralized parameterization of the overall solution.

Project implementation experience shows that the slope of learning associated with the solution is directly proportional to the number of different components contained in the automation suite. (see figure below):

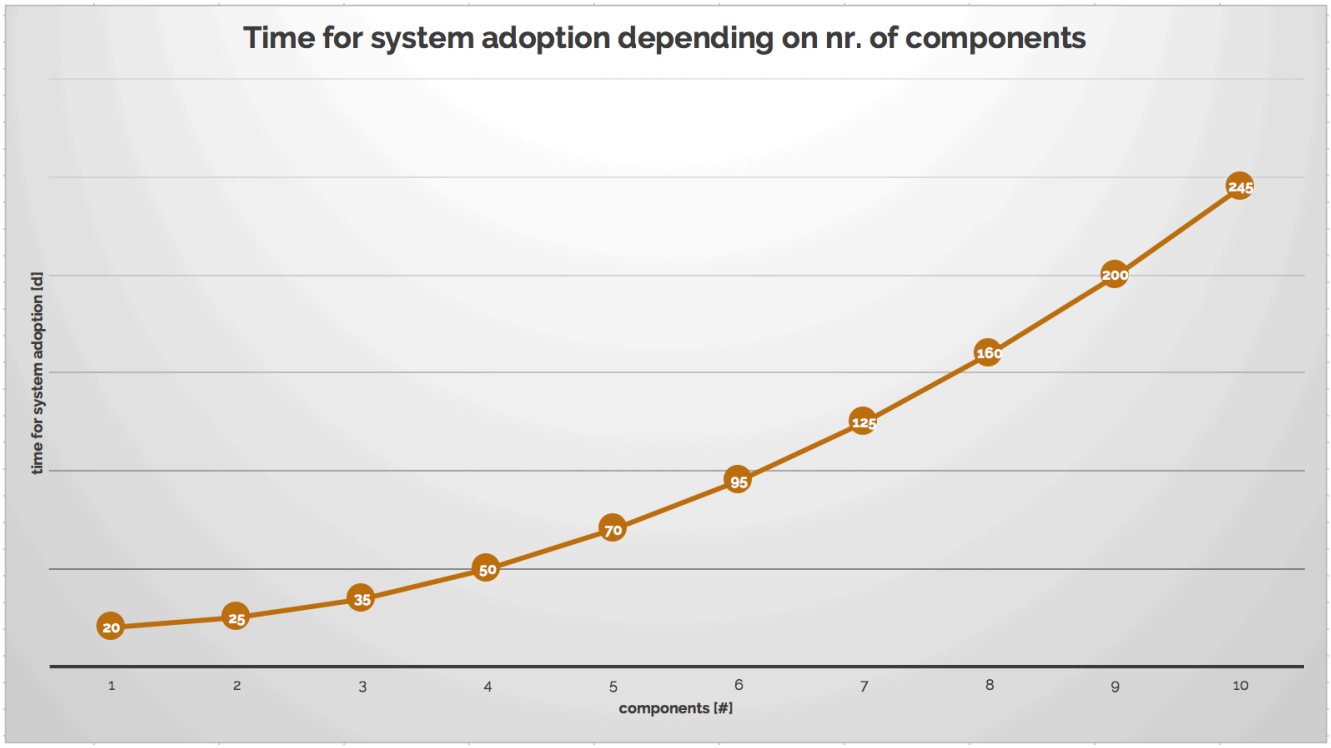


Figure: Learning curve vs. the number of components contained in an automation solution

FUNCTIONAL INTEGRATION WITH TARGET SYSTEMS

Finally, the solution's core functionality should be able to integrate with target systems using industry standards such as common OS scripting languages, REST, JMS or quasi-standards with target applications like RPC and EJB. At minimum, the solution should include 90% of these industry standards out-of-the-box.

In addition, an enterprise-grade automation solution should provide:

- Multiple action/workflow templates (either bundled with the core solution or available for purchase)
- Ease of integration implementation with target systems' core functionality at a very detailed level - such as administrative scripting control from within the automation core through scripting integration (to be discussed in the following chapter)

DYNAMIC PROCESSING CONTROL

The final compelling element to consider in an enterprise-grade automation solution is its ability to dynamically control - in real-time - how automation changes the behavior of your IT landscape. This includes:

- On-demand business process changes requested by stakeholders, mandated by regulatory requirements, or directly affected by events outside of the enterprise's core business model
- Risk of service level (SLA) penalties caused by an application failure or the system's inability to handle a change of load demand
- The ability of the system to support a rapid introduction of a new product line or service to meet changing business needs

When assessing such capabilities, the following architecture patterns within the proposed automation solution are of importance:

DYNAMIC JUST-IN-TIME EXECUTION

As described earlier, object orientation forms the basis for aggregating artefacts built into the automation solution (such as: actions, tasks, workflows, file processing, reports). This capability shall be provided in a way that automation operations remain sufficiently granular and at the same time allow the solution to act as a large automation ecosystem. More importantly, the solution must retain the ability to dynamically re-aggregate executions on-demand as required.

If the automation platform handles each artefact as an object, then object interaction, object instantiation parameters, or object execution scheduling can be redefined in a matter of minutes. All that's left is to define the object model of the actual automation implementation for the specific IT landscape - a onetime task.

The best automation solutions include a library of IT process automation actions that can be aggregated throughout automation workflows. These “IT process automation” actions are ideally delivered as part of the solution as a whole or specifically targeted to address particular automation challenges within enterprise IT landscapes.

Examples are:

- If IT SLA measures reveal a particular IT housekeeping task at risk due to an increase in processing time, dynamic adaptation of the specific workflows would involve assigning a different scheduler object or calendar to the task or re-aggregating the workflow to execute the process in smaller chunks. This assumes end-to-end object orientation and proper object model definition.
- If a particular monthly batch data processing workflow is exceeding a particular transfer size boundary, the workflow can remain completely unchanged while chunk size definition is altered by changing the input parameters. These input parameters would themselves be

derived from IT configuration management so dynamic automation adaptation would still remain zero-interactive.

PRE/POST PROCESSING OF AUTOMATION TASKS

Not only does dynamic execution require maximum object orientation patterns within the implementation and operation of the automation solution, but it must also provide the ability to:

- Change the behavior of an object by preprocessing actions.
- Process the output of an object for further use in subsequent automation tasks/workflows.

Adding pre or post execution logic instead of implementing additional objects for the same logic makes it an object property – an inherent part of the object itself instead of treating it as separate object within the model – which rarely occurs with pre or post-processing. These tasks thus become part of the concrete instance of an abstract automation object.

Examples for applying this pattern in automation are:

- System alive or connectivity check
- Data validation
- Parameter augmentation through additional input
- Data source query
- Dynamic report augmentation

Automation solutions can offer this capability either through graphical modeling of pre and post conditions or in the case of more complex requirements, through script language elements.

EASY-TO-USE EXTENSIBLE SCRIPTING

The scripting language offered by the automation solution is the key to offering a system capable of implementing enterprise-grade automation scenarios. While scripting within automation and orchestration tends to evolve into supporting mainly standard scripting languages such as Python, Perl, JavaScript or VBscript, a solution that offers both standard and proprietary scripting is still optimum.

An automation system's proprietary scripting language addresses the system's own object model most efficiently while at the same time - through extension capabilities - enabling seamless inclusion of target system specific operations. The combination of both is the best way to ensure a flexible, dynamic, and high performing end-to-end automation solution.

FROM AUTOMATION TO ORCHESTRATION

In some cases, choosing an appropriate IT automation solution comes down to a matter of business dependency, migration efficiency, or simply vendor relationship and trust. Many times, these criteria alone might provide enough information to make a solid buying decision.

However, to make the best decision for an organization, examining prospective automation solutions more closely could be an option. A key consideration are the solutions' architectural patterns, which should be assessed through well-planned proof of concept testing or a detailed evaluation. Some key questions to raise are listed below, for convenience and possible inclusion in the evaluation questionnaire:

- Does the solution have sufficient scalability to react to on-demand load changes and at the same time allow for the growth of IT automation and orchestration as your business grows?
- Does the solution provide object orientation which allows for representation of real-world IT challenges through well-structured re-usable automation objects and templates?
- Can the solution quickly and easily integrate and adapt to business process changes?
- Does the solution guarantee 24/7, close-to-100% availability?
- Is the solution able to interface with traditional legacy system files and applications through integrated file transfer capability?
- Can the solution provide a full audit trail from the automation backbone and does it support compliance standards and regulation out-of-box?
- Does the solution offer multi-clients support and have the ability to segregate organizational and customer IT segments?
- Does the solution include the latest security features supporting confidentiality, integrity and authenticity?
- Can the solution handle the integration needs in a homogeneous way from a central automation layer that enables IT admins to come up to speed quickly with minimal training?
- Does the solution dynamically control the execution of processes at all times?

Obviously, not all automation solutions in the market would be able to give a "yes" to each of these questions; and as always the goal is to find the platform that does bundle all of the described criteria in the best possible compromise.

A supportive element to this decision could be a solution's capability not only to automate mundane operations tasks in an IT landscape but to bundle these tasks into larger system orchestration scenarios.

What's the essentials of such scenarios? What's to look at? And what about the delineation between system and service orchestration? This is subject to the following chapters.

WHAT IS ORCHESTRATION?

Prior to diving into the architectural patterns for a robust orchestration solution for the enterprise, a few definitions need to be clarified, and as this paper talked a lot about Automation in the first place, we shall begin with outlining precisely the delineation between Automation and Orchestration – to begin with:

- In any IT architecture framework, the automation layer creates the components necessary to provide atomic entities for service orchestration. Automation uses technologies to control, manage and run atomic tasks within machine instances, operating systems and applications on the one hand and provides automation capabilities for the larger ecosystem in order to automate processes (e.g. onboarding a new employee)
- Orchestration – in turn – uses processes, workflows and integration to construct the representation of a service from atomic components. A service could e.g. consist of operations within various different systems such as the creation of a machine instance in a virtual infrastructure, the installation of a webservice in another instance and the alteration of a permission matrix in an IAM system. "Orchestration" would provide means to aggregate these atomic actions into a service bundle which can then be provisioned to a customer or user for consumption. The Orchestration layer makes use of single automated service components.

While the above definitions mainly address the system context in IT architectures, it is valid to say that there is another slightly different context – the service context – which demands for another definition of the term "orchestration":

- Service orchestration is the coordination and arrangement of multiple services exposed as a single aggregate service. It is used to automate business processes through loose coupling of different services and applications, thereby creating composite services. Service orchestration combines service interactions to create business process models consumable as services.

In order to underpin the danger of confusion when discussing orchestration, here are a few references for orchestration definitions:

- "Orchestration describes the automated arrangement, coordination, and management of complex computer systems, middleware and services." *(wikipedia: https://en.wikipedia.org/wiki/Orchestration_(computing))*
- "Complex Behavior Interaction (Logic/Business Process Level): a complex interaction among different systems. In the context of service-oriented architecture, this is often described as choreography and orchestration" *(Carnegie Mellon University Research Showcase 12-2013: "Understanding Patterns for System-of- Systems Integration", Rick Kazman, Klaus Nielsen, Klaus Schmid)*
- "Service orchestration in an ESB allows service requesters to call service providers without the need to know where the service provider is or even the data scheme required in the service" (*InfoTech Research Group Inc. "Select and Implement an ESB Solution", August 2015*)

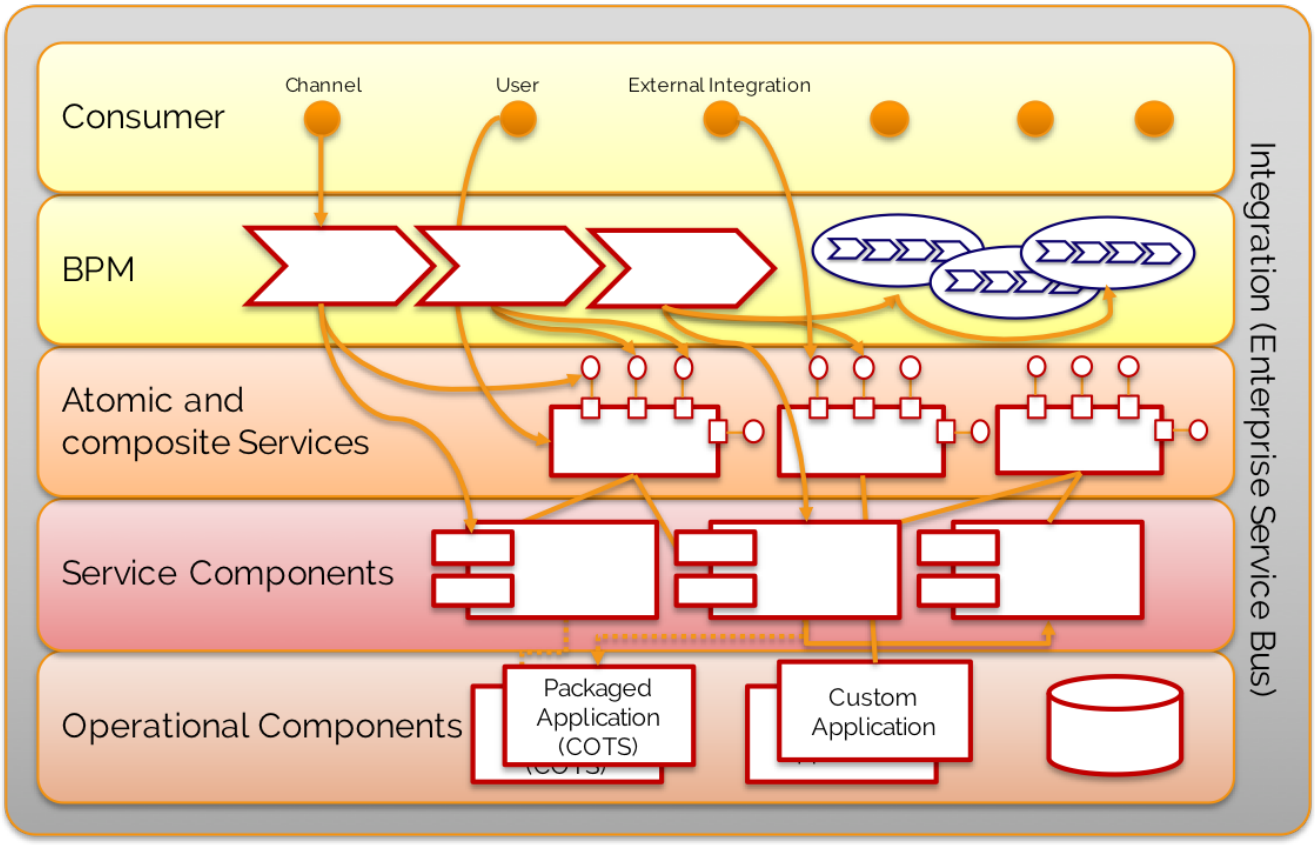
- "Orchestration automates simple or complex multi-system tasks on remote servers that are normally done manually" (*ServiceNow Product Documentation:* *http://wiki.servicenow.com/index.php?title=Orchestration#gsc.tab=0*)
- "The main difference, then, between a workflow "automation" and an "orchestration" is that workflows are processed and completed as processes within a single domain for automation purposes, whereas orchestration includes a workflow and provides a directed action towards larger goals and objectives" (*"Cloud Computing: Concepts, Technology & Architecture", Thomas Erl, Prentice Hall, October 2014*)
- "Orchestration is the automated coordination and management of computer resources and services. Orchestration provides for deployment and execution of interdependent workflows completely on external resources" (*CloudPatterns.ORG:* *http://cloudpatterns.org/mechanisms/orchestration_engine*)

Even though these definitions seemingly leave a lot of room for interpretation of what system and service orchestration really covers, clarity can be gained by looking at a few architectural principles as well as requirements for different orchestration goals, which the last few chapters of this paper will be focusing at.

SYSTEM VERSUS SERVICE ORCHESTRATION

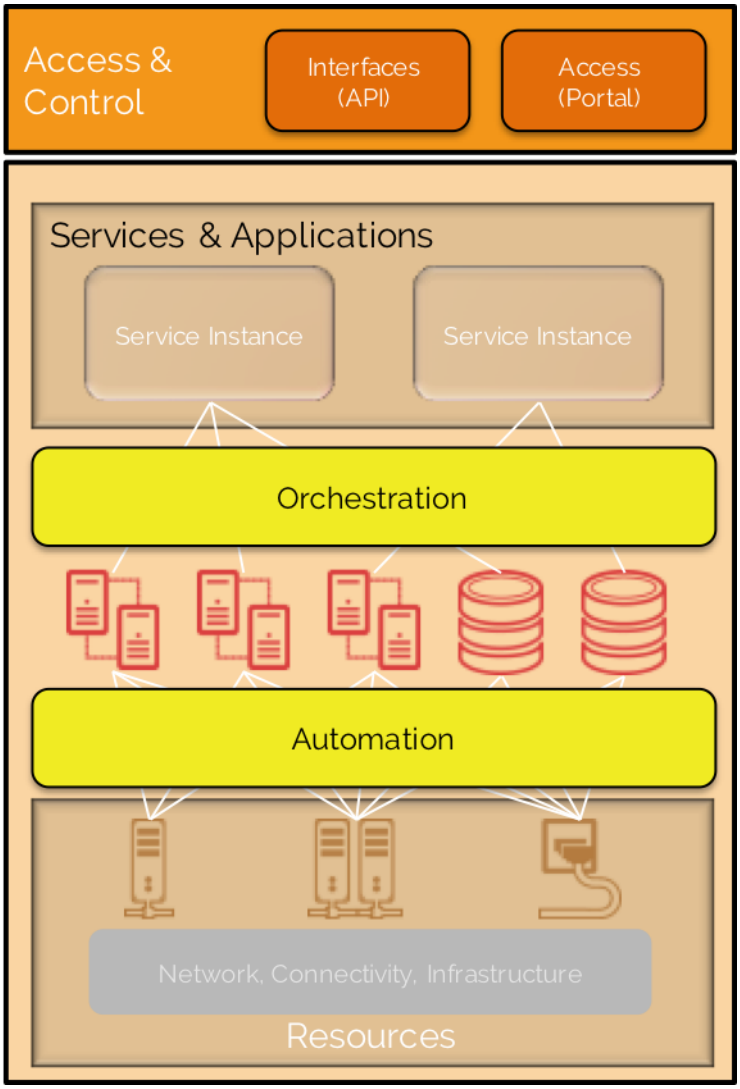
One of the most well-known blueprints for service orchestration is the representation as seen from the perspective of service oriented architecture (SOA).

The following figure in principle describes this viewpoint:



Operational components – such as "commercial off the shelf" (COTS) or custom applications, possibly with a high level of automated functionality (see previous chapters) – are orchestrated to simple application services (service components) which in turn are aggregated to atomic or composite IT services which subsequently support the execution of business processes. The latter are presented to consumers of various kind without disclosing any of the underlying services or applications directly. In well established service orchestration, functionality is often defined top-down by modelling business processes and defining its requirements first and then leveraging or composing necessary services to fulfil the process' needs.

A different approach is derived from typical definitions in cloud frameworks; the following figure shows this approach:



Here, the emphasis lies on the automation layer building the core aggregation. The orchestration layer on top creates system and application services needed by the framework to execute its functional and operational processes.

The latter approach could be seen as a subset of the former, which will become more clear when talking about the essential differences between system and service orchestration.

DIFFERENCES BETWEEN SYSTEM AND SERVICE ORCHESTRATION

System Orchestration

- could in essence be comprised of an advanced automation engine
- leverages atomic automation blocks
- eases the task of automating (complex) application and service operation
- oftenly directly supports OS scripting
- supports application interfaces (API) through a set of plugins
- may offer REST-based API in itself for integration and SOA

Service Orchestration

- uses SOA patterns
- is mostly message oriented (focuses on the exchange of messages between services)
- supports message topics and queues
- leverages a message broker and (enterprise) service bus
- can leverage and provide API
- composes low level services to higher level business process oriented services

Vendor examples of the former are vRealize Orchestrator, HP Operations Orchestration, Automic ONE Automation, BMC Atrium, System Center Orchestrator, ServiceNow (unsurprisingly some of these products have an essential say in the field of automation as well).

Service orchestration examples would be vendors or products like TIBCO, MuleSoft, WSO2, Microsoft BizTalk, OpenText Cordys or Oracle Fusion.

SYSTEM ORCHESTRATION KEY FEATURES

System orchestrators are mainly demanded to support a huge variety of underlying applications and IT services in a highly flexible and scalable way:

- OS neutral installation (depending on specific infrastructure operations requirements)
- Clustering or node setup possible for scalability and availability reasons
- Ease of use; low entry threshold for orchestration/automation developers
- Support quality; support ecosystem (community, online support access, etc.)
- Database dependency to minimum extent; major databases to be supported equally
- Built-in business continuity support (backup/restore without major effort)
- Northbound integratability: REST API
- Southbound integratability and extensibility: either built-in, by leveraging APIs or by means of a plugin ecosystem
- Plugin SDK for vendor external plugin development support
- Scripting possible but not necessarily needed
- Ease of orchestrating vendor-external services (as vendor neutral as possible, depending on landscape to be orchestrated/integrated)

- Self-orchestration possible
- Cloud orchestration: seamless integration with major public cloud vendors

MAIN REQUIREMENTS FOR A SERVICE ORCHESTRATOR

In contrary to the above, service orchestration solutions would focus mainly on message handling and integration, as its main purpose is to aggregate lower level application services into higher level composite services to support business process execution. Typical demands to such a product would therefore involve:

- Support of major web service protocol standards (SOAP, REST)
- Supports "old-style" enterprise integration technologies (RMI, CORBA, (S)FTP, EDI, …) for integration of legacy applications
- Provides a central service registry
- Supports resilient message handling (mediation, completion, message persistence, …)
- Includes flexible and easy to integrate data mapping based on modelling and XSLT
- Supports message routing and distribution through topics, queues, etc.
- Integrated API management solution
- Integrated business process modelling (BPM) solution
- Integrated business application monitoring (BAM) solution
- Extensibility through low-threshold commonly accepted software development technologies

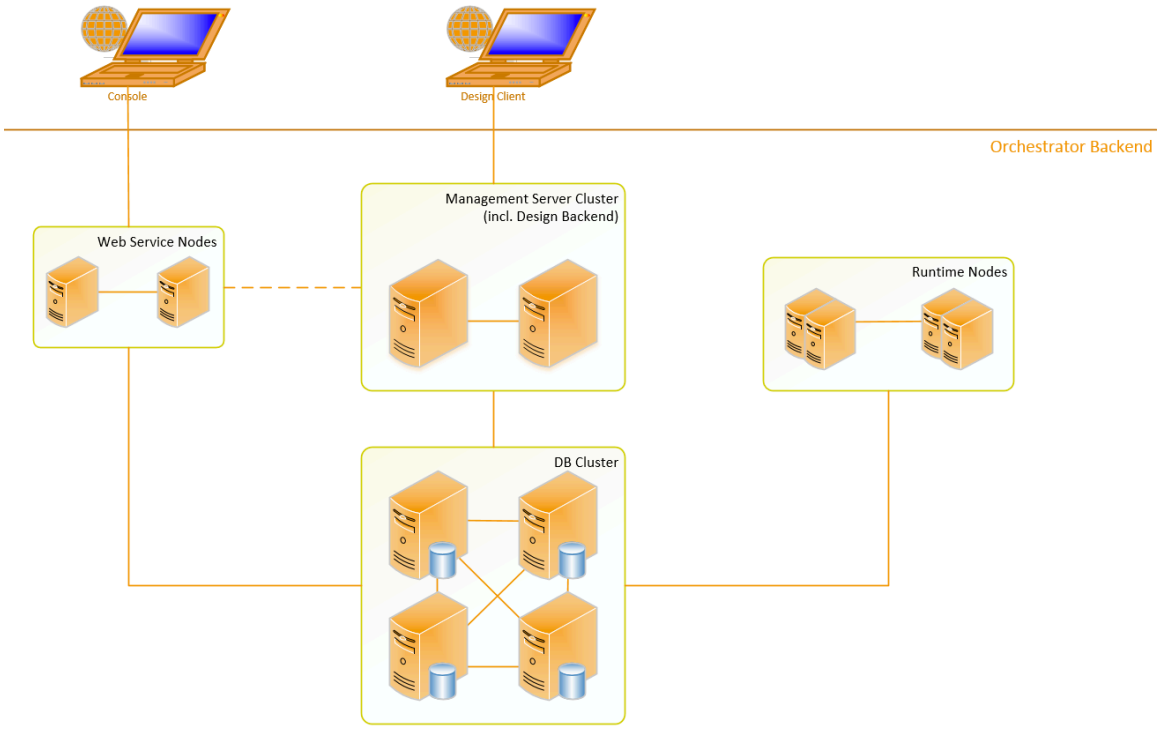
As a rule of thumb to delineate the two effectively from each other, one can say that it is – to a certain extent – possible to create service orchestration by means of a system orchestrator but it is (mostly) impossible to do system orchestration with only a service orchestrator at hand.

For this reason, we will continue with a focus on system orchestration as a way to leverage basic IT automation for the benefit of higher level IT services, and will address vanilla architectures for typical system orchestrator deployments.

SYSTEM ORCHESTRATOR ARCHITECTURE

This final chapter addresses architecture components for typical system orchestrators; comparing these with the blueprints for high-grade innovative automation solutions mentioned previously in this paper will reveal the close resemblance between automation and system orchestration patterns in a very obvious way.

The first figure below is system deployment architecture diagram describing the main physical components for a system orchestrator:

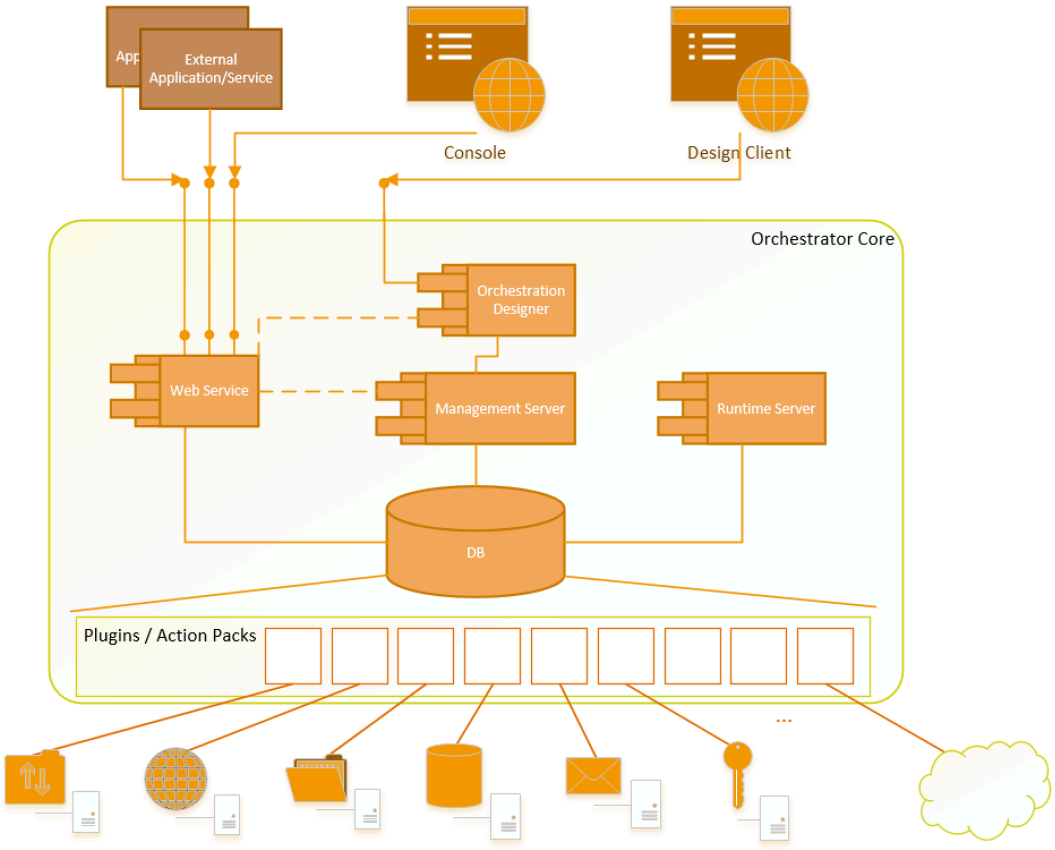


Note, that:

1. the database normally needs to be setup in a clustered mode for high availability. Most orchestrator solutions do rely fully on the database (at least at design time).
2. the Management Server's deployment architecture is depending on availability requirements for management and control.
3. the Runtime server nodes should be highly distributed (ideally geographically dispersed). The better this is supported by the product architecture the more reliable orchestration will support IT operations.
4. the Web service deployment is depending on availability and web service API needs (product and requirement dependent)

LOGICAL ARCHITECTURE

The logical architecture builds on the previous description of the deployment architecture and outlines the different building blocks of the orchestration solution. The logical architecture diagram is depicted in the following figure:



Notes to figure "logical architecture":

1. The Orchestrator DB holds runtime and design time orchestration flows, action packs, activities, plugins, logs, …
2. Management Server controls access to orchestration artefacts
3. Runtime Server provides execution environment for orchestration flows
4. Orchestration designer (backend) provides environment for creation of orchestration flows using artefacts from the database (depending on specific product architecture the designer and management components could be integrated)
5. Web Service exposes the Orchestrator's functionality to external consumers (ideally via REST)

6. Action packs or plugins are introduced through installation at the design time (normally integrated into the DB)
7. The Orchestrator's admin console is ideally implemented as web service, hence accessible via browser
8. The Design Client UI could either be web-based or a dedicated client application to be installed locally and using a specific protocol for backend communication

Of course, these building blocks can vary from product to product. However, what remains crucial to successful orchestration operations (more or less in the same way as with automation) is to have lightweight, scalable runtime components capable of supporting a small scale, low footprint deployment equally efficient to a large scale, multi sight, highly distributed orchestration solution.

CONCLUSION

Automation and Orchestration are core capabilities in any IT landscape.

Traditionally, there'd be classical on-premise IT, comprised of multiple enterprise applications (partly) based on old-style architecture patterns like file exchange, asynchronous time-boxed export/import scenarios and historic file formats.

At the same time, the era of the Cloud hype has come to an end in a way that Cloud is ubiquitous; it is as present as the Internet as such has been for years, and the descendants of Cloud – mobile, social, IoT – are forming the nexus for the new era of Digital Business.

For enterprises, this means an ever-increasing pace of innovation and a constant advance of business models and business processes. As this paper has outlined, automation and orchestration solution form the core for IT landscapes to efficiently support businesses in their striving for constant innovation.

Let's once again repeat the key findings of this paper:

- Traditional "old style" integration capabilities – such as: file transfer, object orientation or audit readiness - remain key criteria even for a cloud-ready automation platform.
- In an era where cloud has become a commodity, just like the internet as such, service centered IT landscapes demand for a maximum of scalability and adaptability as well as multi-tenancy in order to be able to create a service-oriented ecosystem for the advancement of the businesses using it.
- Security, maximum availability, and centralized management and control are fundamental necessities for transforming an IT environment into an integrated service center supporting business expansion, transformation, and growth.
- Service orchestration might be the ultimate goal to achieve for an IT landscape, but system orchestration is a first step towards creating an abstraction layer between basic IT systems and business-oriented IT-services.

Therefore, for IT leaders, choosing the right automation and orchestration solution to support the business efficiently might be *the* majorly crucial decision to either become a differentiator and true innovation leader or (just) remain the head of a solid – yet: commodity – enterprise IT.

The CIO of the future is a Chief Innovation (rather than "Information") Officer – and Automation and Orchestration both build the core basis for innovation. What to look at in getting to the right make-or-buy decision was the main requirement towards this paper.