Service oriented mechanisms for smart resource allocation in private manufacturing clouds

Octavian Morariu, Theodor Borangiu, Cristina Morariu, Silviu Raileanu

IESS1.6, Bucharest, 2016

Agenda

- Research Goals
- Predictive capacity provisioning mechanism
- Queued resource allocation bus architecture
- Adaptive resource allocation in private CMfg

Research goals

- Implement scalability of CMfg based on predefined rules evaluated against real-time metrics:
  - ✓ Development of a cloud based elastic system, capable to automatically increase and decrease its capacity based on real time load without the intervention of the system administrator.
  - ✓ Set up a predictive scalability model based on repetitive usage patterns to assure better resource utilization in private CMfg RT applications (MES-mixed batch planning and product scheduling, IED-OH execution)
- Design a queued resource allocation bus architecture extended with computation of alternative schedules:
  - ✓ Adding 2 key abilities for decision support: 1) offer alternative schedules; 2) use federated resources for request fulfillment (from 2 Cloud systems)
- Set up an adaptive provisioning mechanism for optimal resource allocation
  - ✓ Improve predictability of resource utilization in private CMfg – adjust automatically resource allocation according to RT capacity requirements

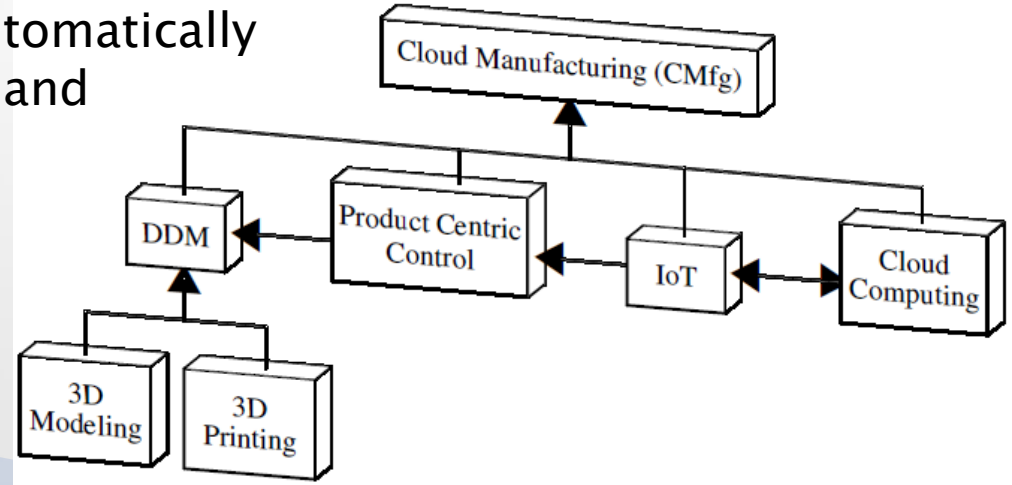
CMfg – service orientation and taxonomy

CMfg *moves* from production-oriented manufacturing processes to customer- and service-oriented manufacturing process networks [by modeling single manufacturing assets as services similar to SaaS or PaaS].

Cloud Manufacturing: service-oriented networked product development models in which service consumers can configure, select, and use customized product realization resources and services [from CAE, CARE software to reconfigurable manufacturing systems].

- all manufacturing resources and abilities for the manufacturing life cycle – *provided in different service models*
- Manufacturing resources and abilities can be *intelligently sensed & connected into a wider Internet*, automatically managed and controlled using IoT and (or) Cloud solutions.

CMfg Taxonomy: underlying concepts and technologies



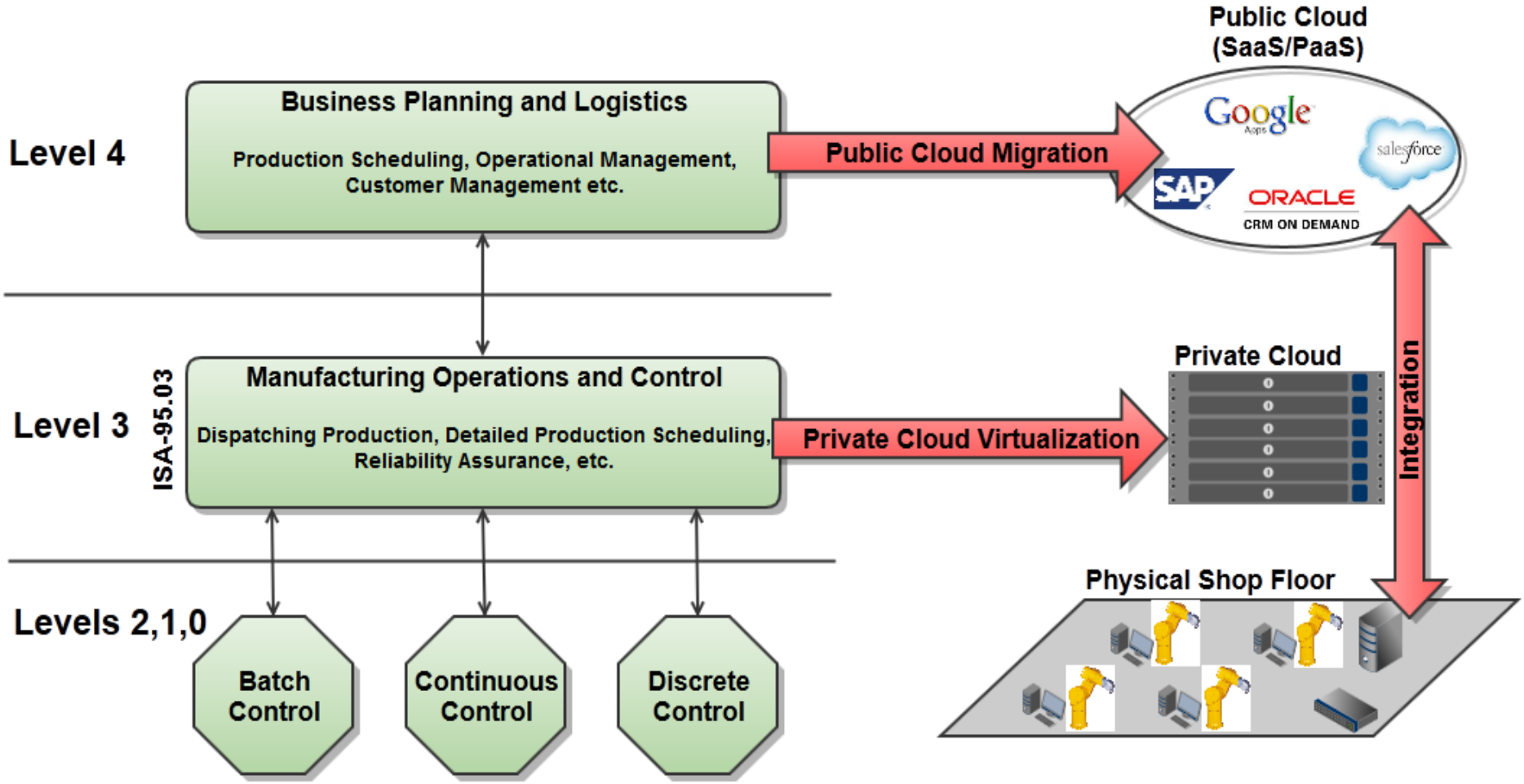
ISA-95 Layers

- **Levels 0, 1 and 2** represent the process control levels and their objective is to directly control the physical shop floor equipment in order to execute the actual production operations that result in one or more finished products;

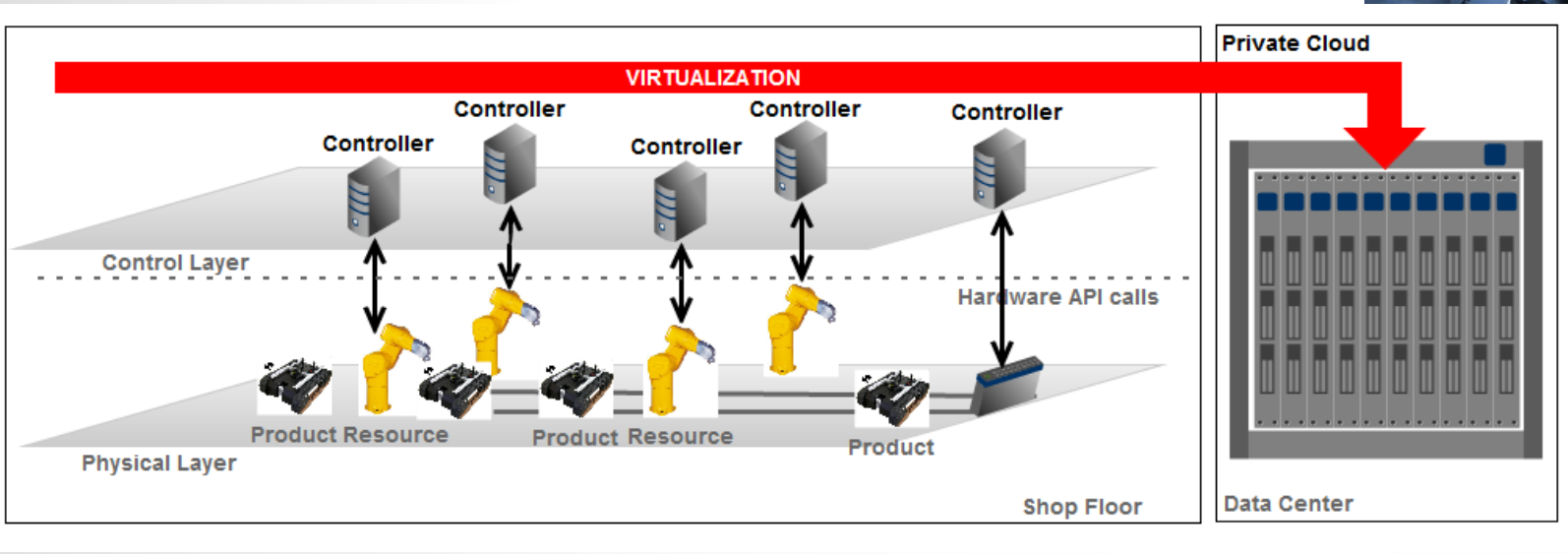
- **Level 3** is the MES (Manufacturing Execution System) level and consists of several activities that have to be executed in order to prepare, monitor and complete the production process executed at level 0, 1 and 2;

- **Level 4** is the ERP (Enterprise Resource Planning) level that executes the financial and logistic activities.

ISA-95 and virtualization (vMES)



CMfg: MES and Shop floor Virtualization



[IBM CloudBurst platform, University Politehnica of Bucharest, 2016]

- This separation between hardware and software provides *high flexibility and agility* to the manufacturing solution.

- The basic concept of MES and shop floor virtualization: *migration of all workloads* traditionally executed on physical machines located on the shop floor to the data centre, specifically to the private cloud infrastructure as virtual workloads.
- Run all the control software in a virtualized environment and keep only the physical devices on the shop floor.

Predictive capacity provisioning mechanism

Standard implementing of scalability in private clouds – based on predefined thresholds; can be defined in low level metrics: CPU / memory / disk usage (IBM Tivoli Monitor uses a Linux based OS agent to collect real time metrics and monitor predefined thresholds).

When a threshold is reached a new workload instance can be provisioned based on the predefined workflow.

The workflow is executed by IBM Tivoli Provisioning Manager.

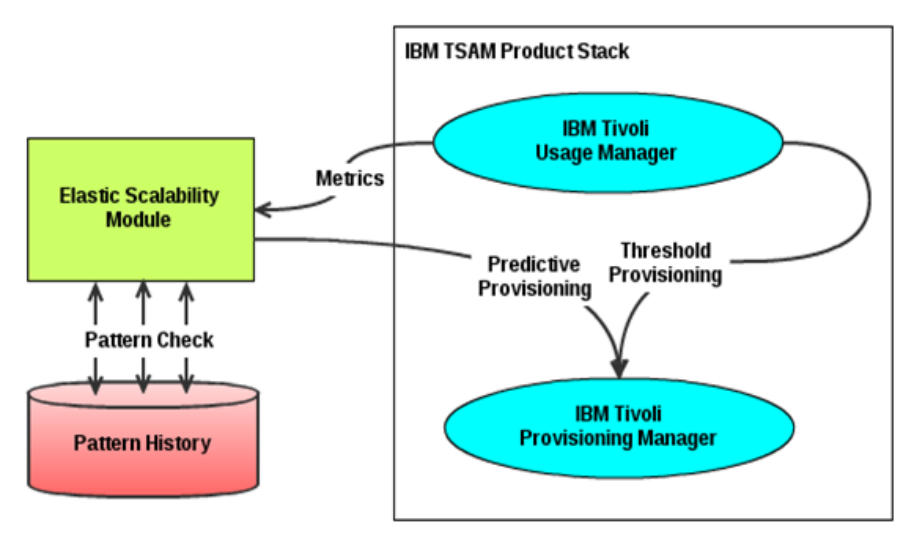
Disadvantage: the provisioning time → set thresholds at a lower level than dictated by capacity requirements, to allow time for the new instance to become active before the capacity is exceeded by user load.

New proposal:

- ✓ A predictive mechanism augmenting the generic threshold- based implementation, by recognizing repetitive (daily and weekly) pattern in application usage.
- ✓ The model predicts the future required capacity and acts *before the threshold is reached*, allowing setting higher levels for thresholds and so avoid false positive triggers.
- ✓ Implementation for IBM CloudBurst 2.1 on System x.
- ✓ The Elastic Scalability Module (ESM): encapsulates the predictive algorithm, augmenting the CloudBurst 2.1 threshold mechanism.

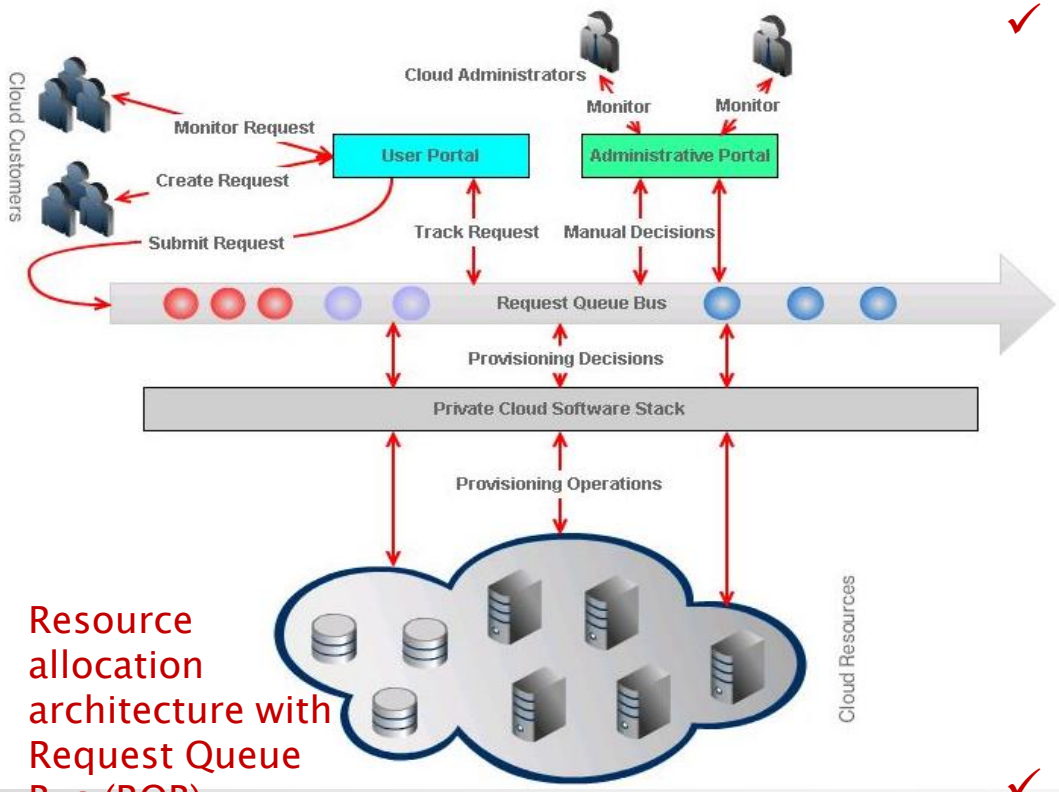
Predictive capacity provisioning mechanism

- The ESM collects RT usage metrics provided by IBM Tivoli Usage Manager (TUM):
- Two operational phases:
  - ✓ The learning phase: ESM stores the metrics provided by IBM TUM per day for each day of the week in a relational database together with the threshold trigger events generated by TUM. This is done several times until a pattern is established.
  - ✓ The driving phase: ESM sends predictive provisioning instructions to the IBM Tivoli Provisioning Manager (TPM), eventually replacing the trigger based behaviour of TUM).



- ✓ Overall goal of the ESM algorithm: keep the capacity as close as possible to the average user load using historical usage patterns.
- ✓ [Current provisioned capacity – the estimated capacity] / the instance capacity = the number of instances that need to be provisioned or de-provisioned

Queued resource allocation bus architecture



Resource allocation architecture with Request Queue Bus (RQB)

- ✓ Tivoli Service Automation Manager (TSAM) software stack *manages the resources in the private cloud*. As part of the Tivoli stack the Tivoli Service Request Manager (TSRM) allows customers to create tickets that trigger an internal approval workflow. TSRM implements *initial ticket validation* that prevents customers to create requests if the system does not have the required capacity for provisioning in the in the required interval.
- ✓ Adding two key abilities for decision support:

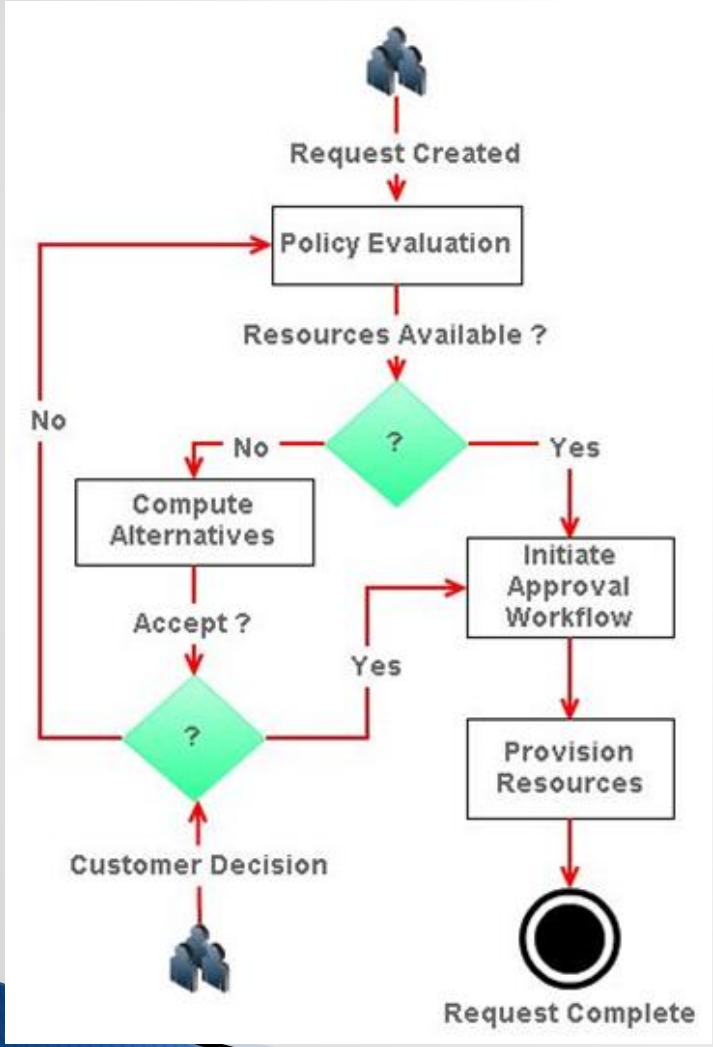
- ❑ the ability to *offer alternative schedules for customers in case the resources are not available at the required time*, instead of a simple request deny of the request either;
- ❑ the ability to *use federated resources for request fulfillment*, in the scenario where two or more private clouds are interconnected

Queued resource allocation bus architecture

Request Queue Bus (RQB). Functionalities:

- ✓ Requests are submitted by the customers through a user portal
- ✓ The requests are queued for processing rather than assigned directly for an immediate decision.
- ✓ The user portal allows creation of requests based on the services catalogue offered by the underlying private cloud software stack, without performing a prior validation in regards to the available capacity in the time interval required.
- ✓ The request object data structure contains references to *the service catalogue* items selected together with the associated amounts.
- ✓ The request data structure contains *the initial time interval* for which the respective resources should be provisioned.
- ✓ The user portal assembles the request object and submits it in the resource allocation queue.
- ✓ The user portal also allows customers to track requests created and provide further responses on the options for request fulfillment provided by the system.

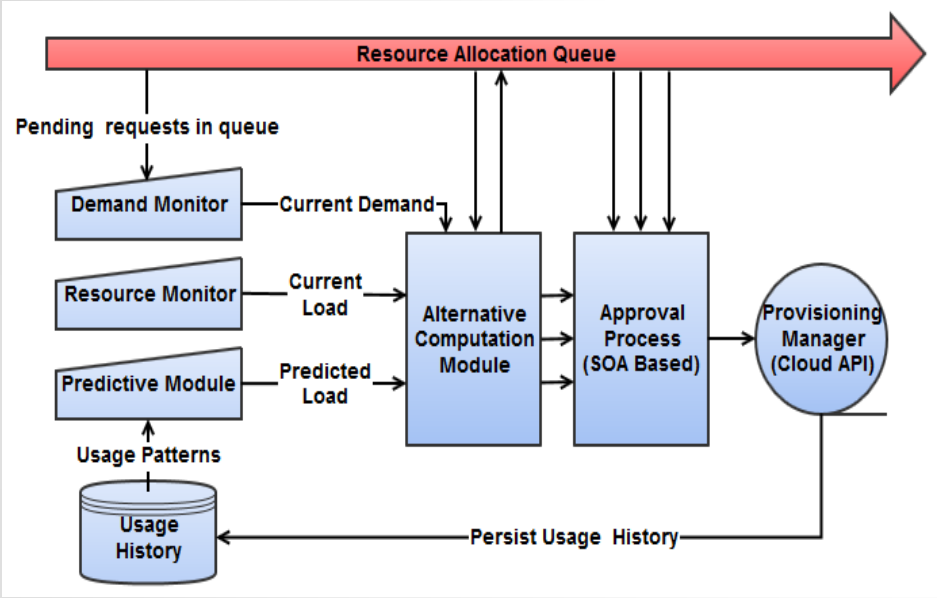
Queued resource allocation bus architecture



The request lifecycle

- ✓ Cycle start: the request is created by the customer (contains the requested resources along with the initial timeframe).
- ✓ If the resources are available, an approval workflow is initiated.
- ✓ If approved, the resources are provisioned or a resource reservation is created.
- ✓ If the resources are not available in the timeframe requested, the alternatives are computed and presented to the customer for acceptance:
  - ➢ If one is accepted, the approval workflow is initiated and normal provisioning is started.
  - ➢ If not, the request remains in the request queue and policies are re-applied.

Queued resource allocation bus: computing alternative schedules



Interactive Scheduler Architecture

Alternative schedules are computed based on two factors:

- ✓ *timeframe requested* and
- ✓ possibility to fulfill the request with *burst resources* (resources provided by interconnected private clouds)

Algorithm - computing alternatives:

- ✓ *shift timeframe* with preconfigured intervals in both directions on timeline;
- ✓ for each shift, the *resource availability is computed*; the closest alternatives relative to the timeframe, are added to the *set of options* presented to the customer
- ✓ evaluate the feasibility of each option against a set of constrains obtained in real time from:
  - Demand Monitor (summed current demand for: CPU, memory, storage)
  - Resource Monitor (RT system load: current available local capacity & burst capacity)
  - Predictive Module (predictive load information based on usage patterns stored in a persistent storage)

Adaptive Resource Allocation in private CMfg

Goal: *implement an adaptive behaviour* to automatically adjust the resource allocation for cloud applications according to real time capacity requirements.

Effect: improves *predictability* of resource utilization.

Principle: active *monitoring of cloud applications* (MES, web, J2EE, database) and *recording multiple factors* (CPU, memory, I/O, networking) provide relevant information = complex triggers for the cloud adaptive behaviour and smart resource allocations.

Solution: service oriented mechanism:

- Assures *adaptability of a CMfg private cloud system to workload fluctuations*, capable of intelligent resource allocation in terms of amount and co-locations based on virtualization optimization.
- *Real time monitoring information* is gathered with a multi-agent monitoring system capable of multi-layer and multi-factor monitoring.
- The *smart resource allocation* is achieved with a distributed genetic algorithm that considers the workload characteristics in conjunction with physical optimum allocation and the current load.

Workload and types of triggers for adaptive allocation

Workload Profiles

OS	CPU Profile	IO Profile
Windows	High	Low
Windows	Low	High
Linux	High	Low
Linux	Low	High

Example: a MES workload runs parallel processing applications which:

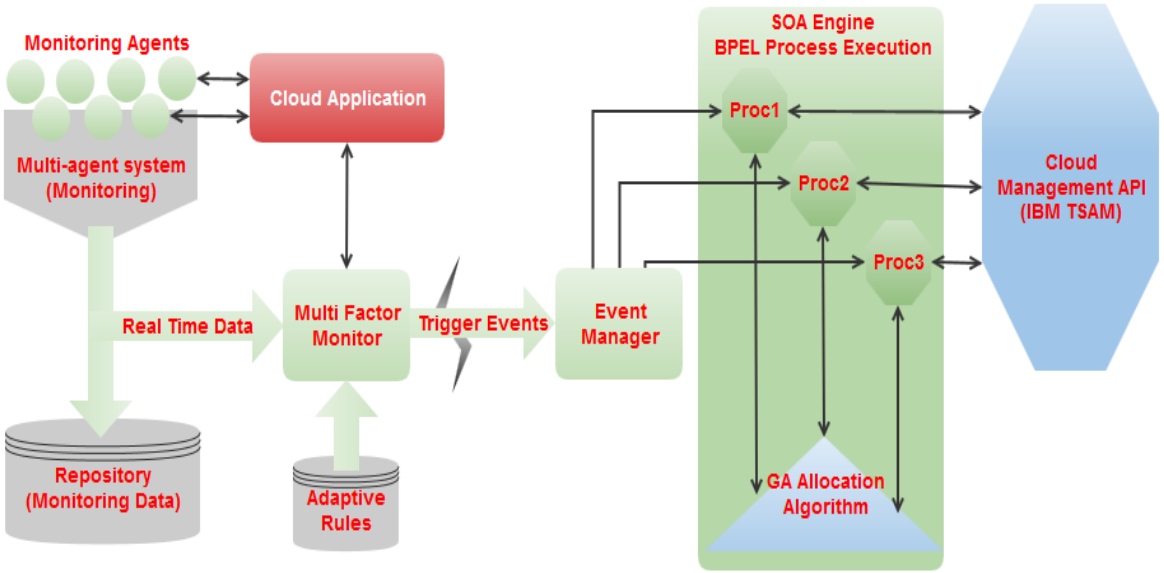
- have a CPU intensive profile (*mixed batch planning and product scheduling*) or
- run a high end relational database system, with an IO intensive profile (inventory, Supply Chain Mngmt).

Event Triggers invoking adaptive resource allocation

Metric	Definition	Type
CPU (OS Layer)	Threshold	Simple
Memory (OS Layer)	Threshold	Simple
Disk IO (OS Layer)	Threshold	Simple
Network (OS Layer)	Threshold	Simple
Combined (OS Layer)	Rule	Complex
CPU (App Layer)	Rule	Simple
Memory (App Layer)	Rule	Simple
Disk IO (App Layer)	Rule	Simple
Network (App Layer)	Rule	Simple
Application Defined	Rule	Simple
Combined (App Layer)	Rule	Complex

- ✓ Sources of events triggering adaptive resource allocation: 1) the monitoring layer and 2) the application itself.
- ✓ A MAS gathers RT data at multiple layers in the cloud application stack: hypervisor, OS, application layers. The data generated by the monitoring solution sends multi-factor data (CPU, Memory, IO, Network) in a repository in a continuum stream.
- ✓ A central monitoring agent triggers events based on a set of rules: single metric / complex multi-factor conditions.

Adaptive provisioning mechanism for optimal resource allocation



Adaptive provisioning mechanism (APM) architecture:

1) **MAS monitoring solution**, gathers real time metrics from both OS layer and application layer; 2) **a multi-factor monitor** triggers reconfiguration events based on administrator defined rules; 3) a **business process manages** the life cycle of the cloud project ; 4) a **genetic algorithm** for optimal resource allocation in the private cloud

APM function:

Scale up and down the resources required by the application, as: "*when to scale*?", "*what to scale*?" and "*how to scale*?"

Solution implemented as a BPEL process, runs on top SOA engine

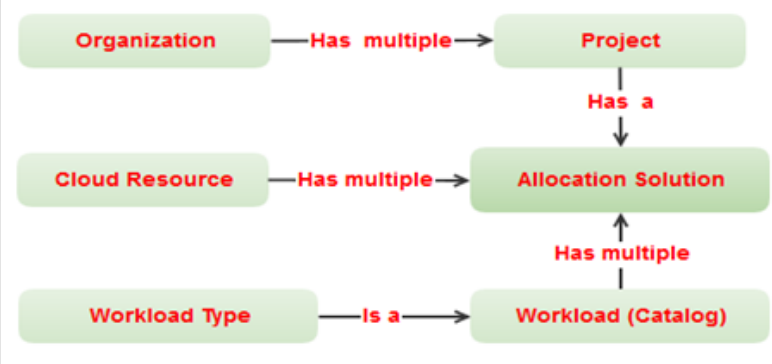
Event Manager:

- ✓ *Matches the events* triggered with the corresponding BPEL process and
- ✓ *Sequences the events* that might be triggered for the same process in short amounts of time. **Useful** in:

  a) reducing the impact of resource allocation overhead & configuration on decision making process,

  b) eliminating duplicate events

Genetic Algorithm for smart resource allocation



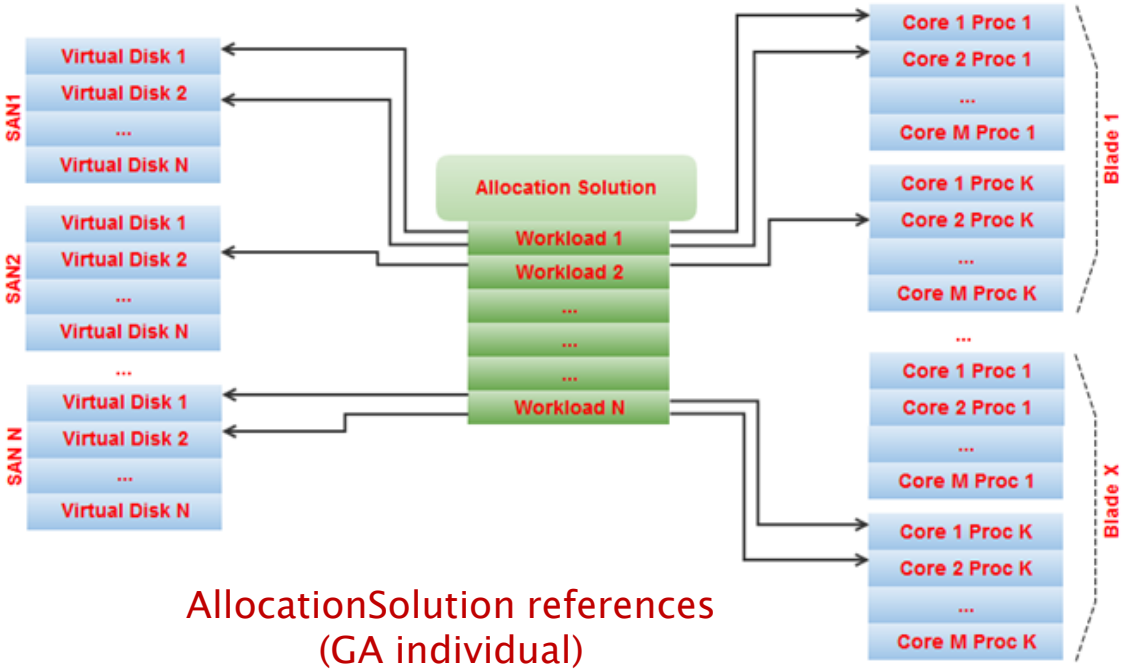
Data Objects class diagram

- GA starts with a randomly generated population of solutions
- By applying operations as *selection, crossover and mutation* on individuals, the *GA creates new generations* evaluating the fitness of each individual of the population in the process.
- When the fitness level in the population reaches a satisfying value, a *set of solutions* are obtained.

Design of the GA prototype:

- ✓ The AllocationSolution class holds the allocation information of the cloud resource to the workload in the scope of a project. The allocation solution class instance has a reference to the Project for which the allocation is computed.
- ✓ The CloudResource class represents the top of a hierarchy of cloud resources like: blade server, memory unit, virtual core, network, disk storage and so on.
- ✓ The Workload class has a reference to the cloud service catalogue and represents a virtual machine template. Each workload is described by a WorkloadType.
- ✓ The Organization class holds a reference to all the projects the organization has.

Genetic Algorithm for smart resource allocation



AllocationSolution references (GA individual)

The AllocationSolution instances are the *actual individuals in the solutions population* of the GA.

Condition5 (Soft) IO Intensive: Checks that for each SAN there are no more than 2 IO intensive WkT scheduled

Condition6 (Soft) Uniform Distribution: computes a factor characterizing the distribution of workloads. Goal: obtain a uniform distribution of workloads across all blades.

The fitness function :
Computes the fitness value for each population's individual, by evaluating a set of conditions against the schedule instance:

Condition1 (Hard) Total Load: iterates all the time slots and computes a sum of all workloads that are scheduled for each blade

Condition2 (Hard) Possible Allocation: Checks if the resource allocation scheme is valid

Condition3 (Soft) Memory Over-commitment: Checks that for each time slot only a single set of WorkloadType is scheduled

Condition4 (Soft) CPU Intensive: Iterates physical CPUs & checks that no more than 2 CPU intensive WorkloadTypes (WkT) are scheduled at the same time

Conclusions. Closed loop QoS monitoring in private CMfg

A multilayer QoS monitoring architecture was implemented on top of IBM CloudBurst 2.1 private manufacturing cloud solution designed and integrated with IBM TSAM product stack

A set of metrics are collected in this implementation at various layers. From Tivoli Stack: CPU, RAM Usage, Disk I/O rate, Network I/O; from VMware hypervisor: workload uptime, CPU quota usage, memory quota usage, virtual disk usage.

At OS layer, the metrics collected are: CPU usage per process and per kernel/user, memory usage per process, disk I/O per device, network traffic/interface.

Database statistics are collected with a specialized ANT job that computes average usage reports in terms of queries/second, top queries & database locks.

The extension built for the IBM CloudBurst 2.1 system is capable of preventive and adaptive resource provisioning on an ISA-95 MES level.

Future work: integrate field device data in CMfg through Industrial IoT.

Thank you!