# Service Granularity in Industrial Automation and Control Systems

Aydin Homay
*Faculty of Computer Science*
*Technische Universität Dresden*
Dresden, Germany
https://orcid.org/0000-0002-6425-7468

Mário de Sousa
*Faculty of Enigineering*
*University of Porto*
Porto, Portugal
msousa@fe.up.pt

Alois Zoitl
*LIT | CPS Lab*
*Johannes Kepler University Linz*
Linz, Austria
alois.zoitl@jku.at

Martin Wollschlaeger
*Faculty of Computer Science*
*Technische Universität Dresden*
Dresden, Germany
martin.wollschlaeger@tu-dresden.de

***Abstract*— Applying service-oriented concepts in industrial control and automation systems requires developing new architectural concepts to make it more adaptable to this context. In the scope of service-oriented systems one of the main challenges is defining the right granularity for every service in the system. This paper presents a systematic classification of methods to decompose the services. Identifying and using the correct granularity for each service is expected to lead to higher system flexibility, scalability, and in general to improving service deployment and maintenance. To find the best granularity for a service several approaches have been proposed, some of which are based on modeling and others based on service quality analysis. The approach we propose is based on modeling.**

***Keywords—service, granularity, automation, feed-forward, feed-backward, control loop.***

## I. INTRODUCTION

The Service-Oriented Architecture (SOA) was very successful in building loosely coupled modules, commonly referred to as services in large enterprise systems. Its new iteration, the Microservices Architecture (MSA), tries to strip away the unnecessary complexities that SOA has to improve several aspects such as flexibility, interoperability, independence, and scalability [1]–[5].

Successful studies are showing that service-oriented approaches can bring more flexibility, scalability, and interoperability to industrial control and automation systems [6]. However, one of the main challenges in service-oriented systems is to find a perfect granularity for each service in the system. Service granularity refers to the size of service [7]. In general service granularity could be categorized into two categories, physical granularity which refers to the physical size of a service and semantic granularity which refers to the logical size of service [6].

The reason why service granularity is important is illustrated in Fig. 1[7]. As may be seen from the figure service granularity has a direct effect on the cost of common operations in service-oriented systems. The more coarse-grained the granularity the higher the costs are for data processing, load balancing, complexity and maintenance, and the lower for network roundtrips, redundancy, synchronization, and traceability.

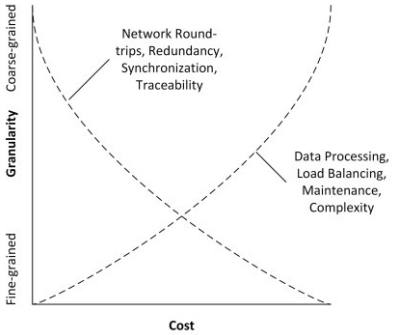


Fig. 1. Service granularity vs cost of operations in service-oriented systems.

The problem in finding service granularity is to identify a correct boundary (size) for each service in the system. In other words, each service in the system needs to have a concrete purpose, as decoupled as possible, and add value to the system.

A service has a correct or good granularity if it maximizes system modularity while minimizing the complexity. Modularity in the sense of flexibility, scalability, maintainability, and traceability, whereas complexity in terms of dependency, communication, and data processing.

The aim of this paper is to define a systematic classification of how to decompose the services. It introduces different architectural concepts for service granularity in automation and control systems. Some of these concepts are based on semantic size while others are based on physical size.

The main reason for having different service granularity classifications is that depending on automation and control system requirements and criticality, each approach will lead to a different number of services and different degrees of modularity and complexity. We will see how each approach will minimize or maximize different criteria in the system.

978-1-7281-8956-7/20/$31.00 ©2020 IEEE

The remaining part of this paper is structured as follows: Section II presents related works. Section III presents a running example. Section IV introduces three different service granularity type while section V defines some quality and quantitative metrics. Section VI summarizes different granularities in Hybrid Granularity. Finally, a conclusion of this paper will form in section VII.

## II. Related Works

Service (de)composition is a crucial aspect in the design phase, implementation phase, and run-time phase of a service. The reason why service (de)composition is so important is that too coarse-grained services may contain too many functionalities while too fine-grained services may contain too few functionalities. The result of an incorrect granularity could lead the system into a chatty communication or very tightly coupled and complex structure that requires more resources and is hard to maintain. This also directly affects overall system flexibility and scalability. Therefore, finding the right granularity is very important especially in the case of constrained environments like Industrial Control Systems (ICS).

There are different approaches to finding a correct or good granularity for a service. For example, Quality of Service (QoS) based solutions are often suitable for the run-time phase. The work carried by V. Gabrel et al. [8] introduces a complexity analysis of QoS-aware workflow-based Web Service composition. The authors argue that the composition problem comes from global QoS constraints.

Another similar work carried by A. Homay et al. [7] introduces a cost analysis service granularity solution for identifying the service granularity via measuring run-time and some other indicators such as inter-service communication overheads.

Analyzing service complexity by modeling and verification methods are often design phase service granularity [9]–[13]. This type of method will provide an opportunity to save time and reduce overall cost in service implementation. However, unlike QoS based methods that focus on service performance, the analytical methods are mostly dealing with the high-level architecture of services.

Another similar work that is based on modeling is the work carried by H. Bloch et al. [14] which suggests using modules in Modular Process Automation to identify the service granularity. However, he does not reject the possibility of having more than one service for one module.

The work carried by S. Boukharata et al. [15] is using a multi-objective optimization method to improve service granularity through interface re-modularization. This type of service granularity analysis is more in the implementation phase where at least the service interfaces/contracts are identified and implemented.

Deriving service granularity from the size of the service development team or using software design patterns to identify (non-)functional service requirements is also a common practice for service granularity. For example, applying the *bounded context* approach from Domain-Driven Design by mapping services to business contexts [16].

In summary, service granularity could be identified by measuring QoS metrics at run-time or could be derived from model-based approaches like Modular Process Automation or Domain-Driven Design in the design time. This paper is introducing model-based approaches by defining three main types of granularities based on different aspects of the system to identify service granularity at design time.

## III. Running Example

This section presents a real example of a control system that will help in better understanding the different granularity types that will be discussed in the following sections. The example is a heat-exchange and a level control system.

In the example, a containment vessel along with valves, sensors and actuators are all together considered a single machine which needs to be controlled. Different control loops are defined. The control system needs to be implemented in an integrated approach that uses a feed-forward, feedback, and a cascade control system for a heat-exchanger with a level controller. The detailed system requirements are summarized in the following P&ID (Piping and Instrument Diagram).

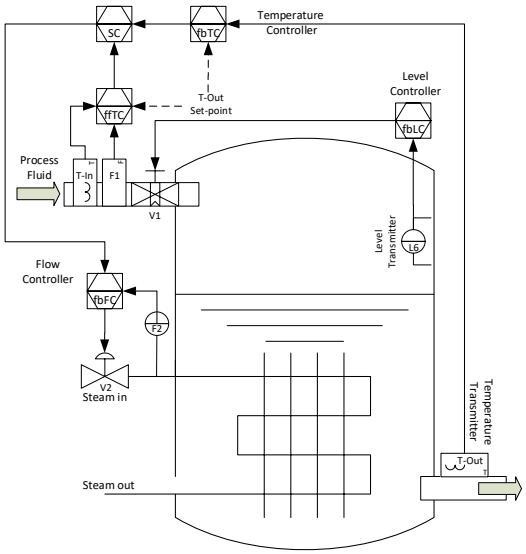


Fig. 2. T-In and T-Out are temperature transmitters, V2 and V1 are values, F1 and F2 are respectively fluid-in and steam-in transmitters, and L6 is a level transmitter.

A heat-exchanger is a thermal energy exchange system between a hot and a cold fluid [17]. The controller of the primary loop determines the set-point of the summing controller in the secondary loop. In level-control systems, fluid flow rates are controlled by varying the degree of opening of a gate-type valve. In the schema represented in Fig. 2 a controller with a summing function (SC) totals the input from both feed-forward temperature controller (ffTC) and feedback temperature controller (fbTC) and sends a unified signal to final control element feedback flow controller (fbFC). At the same time, a feedback level controller (fbLC) is controlling the level of the vessel.

In principle, a feedforward loop will handle major disturbances in the process fluid while a cascade flow control loop will handle issues related to steam pressure and valve

problems, and a feedback loop will handle everything else. Note that combining the three techniques to optimize heat exchanger temperature control is necessary to minimize process variance, maximize product quality, and ensure energy efficiency in petrochemical industries. However, having a level controller in a heat exchanger is not so common but it was added to make the case study sufficiently complex to cover various concepts that will be discussed during the paper.

## IV. Service Granularity Types

ISA88 introduces three multi-level hierarchical models for batch control. The first one is the Process Model which is the basis for defining equipment independent recipe procedures. The second one is the Procedural Control Model which contains different procedures to accomplish the task of a complete process (or part of process) based on the resources of a specific process. The last one is the Physical Model that describes equipment entities, as well as their relationship to a manufacturing enterprise [18]. Based on these three models we derived the idea of service granularity types in three layers, Process layer, Functionality layer (or Procedural layer), and Machine layer (or Physical layer).

A service could be granularized based on the processes that will be handled by the service. We call this type of granularization the Process Granularity. A service could be granularized based on the back-end systems (Machines) that will cooperate with the service. We call this type of granularization the Machine Granularity. A service could be granularized based on the number of functionalities that will be exposed to the rest of the system. We call this type of granularization the Functionality Granularity.

### A. Process Granularity

In Process Granularity (PG) the target is to identify the service boundaries (size) according to the control processes. In other words, dividing the control processes into several independent pieces will specify the granularity for each service. This idea is like identifying service granularity based on UseCases in UML language. In UML, each UseCase represents a set of actions that happens in a sequence [19]. Usually a coarse-grained UseCase will need refinement and this refinement could introduces new smaller UseCases. This is exactly what we can observe in Process Granularity as well. Based on the running example (see Fig. 2) creating an independent service for each scenario (UseCase) will lead int more fine-grained services. In contrast combining both processes into one service will make one coarse-grained service. This illustrates that our decision parameter between coarseness and fineness in Process Granularity is derived from the granularity of the process that is being controlled. Note that the Process Granularity refers to the semantic size of service. It is also important to point out that cutting processes in such a way that each process will be an independent process could help to remove inter-service communications (see Fig. 3) but atomizing each control process may reach pure fine granularity in the cost of inter-service communication overhead.

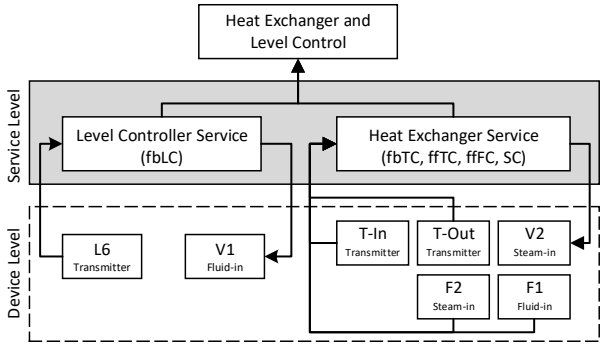


Fig. 3. Process Granularity - Two services are controlling two major processes heat exchanger and level controller. Directional arrows are showing the data-flow between devices and services.

### B. Machine Granularity

In service-oriented industrial automation systems machines are acting as backends for services, sometimes even storing the final state of a request handled by the service. In the concept of Domain Driven Design [16], machines are domains and each service will interact with a machine as a whole in coarseness (the entire machine as one domain) or with a specific part of the machine in fineness (dividing machine in several domains).

In other words, in Machine Granularity (MG) the service granularity will be mapped to the different parts of a machine in fineness, or the entire machine in coarseness. This means one service per machine will result in a coarse-grained service while in contrast several services per machine will result in more fine-grained services. According to the running example (see Fig. 2) the entire vessel including the controllers, transmitters, and valves could be considered as one machine which will result in a coarse-grained service. Such a machine could also be divided into several components like sensors, actuators, controllers, etc. and for each part one independent service is created which will result in more fine-grained services. The idea of dividing a machine into several parts in a hierarchy is not a new idea. Hierarchical architecture for modeling distributed control systems in [20] and equipment entity architecture in ISA88 [18] has already discussed this idea by presenting a hierarchal structure from a machine and its parts.

Machine Granularity works towards the same idea to introduce service granularity in a hierarchy that is structured based on different parts of a machine (compare Fig. 4 with Fig. 3).

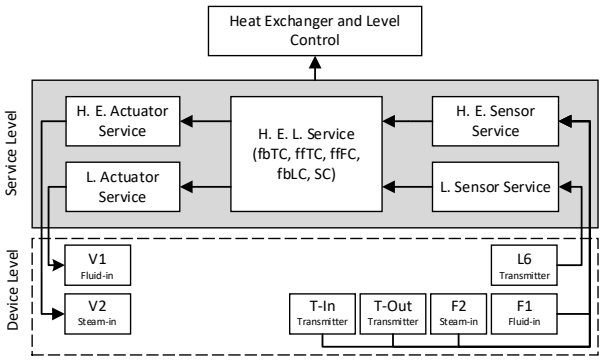


Fig. 4. Machine Granularity - Five services are controlling different parts of the machine. H. E. Actuator Service is controlling the steam-in value. L. Actuator is controlling fluid-in valve. H. E. Service and L. Service are executing the control logics, and H. E. Sensor Service and L. Sensor Service are reading values from the sensors. Directional arrows are showing the data-flow between devices and services.

Machine Granularity also refers to the semantic size. Note that this type of granularity improves the fault tolerability of control system, makes the control system more extensible and scalable, and increases the flexibility. However, it also introduces inter service communication.

The question is why such granularity is necessary? The answer to this question really depends on system functional and nonfunctional requirements. For example, if there is a requirement saying that in the case having a failure in the level controller (control logic failure) service the fluid-in valve must still be able to receive/trigger a close command. In such scenario having a separate service that controls the fluid-in valve would be a must to ensure that the failure in the level controller service will not cause hazardous situation by letting the fluid-in valve stay open. Consequently, if we need to receive the real-time value of the vessel level even in the case of control logic failure in level controller service in order to switch the control system from automatic to manual, then having an independent service that reads the value of the level transmitter would be needed.

The main interest of Machine Granularity is to keep the machine under control more responsive under different circumstances to reduce hazard.

### C. *Functionality Granularity*

In Functionality Granularity (FG), the size of service is the size of function. This means for each major function (where a control loop is running like a PID) will be one independent service. This type of granularity will influence modularity and reusability. Fine-grained services will result in higher modularity and better reusability, while in contrast coarse-grained services will decrease modularity and diminish reusability, this will also have a negative effect on service stability.

Some of the Object-Oriented principles are useful in Functionality Granularity, like the Reuse/Release Equivalence Principle which indicates that the granule of reuse is the granule of release or the Common Closer Principle which says gather into one component (service) those functions that change for the same reason at the same time and separate into different components (services) those functions that they change for different reasons and at different times [16].

Considering the running example (see Fig. 2) one could create fine-grained services by creating an independent service for each controller (fbTC, fbLC, ffTC and fbFC) and orchestrating or choreographing them to handle the entire control logic or one coarse-grained service to hold all controller functions (control logics).

Fig. 5 shows an example of Functionality Granularity which unlike Machine Granularity has separated the fbTC and ffTC into two independent services. This allows scaling the control system more easily. For example, by shutting down the fbTC we could scale from an integrated control approach (cascade plus feedforward and feedback) to a simple feedforward and feedback control system. It also makes the control system more fault tolerable since if one service fails, another service can keep the control system up and running to some extent (graceful degradation).

The Summing Service is acting as a service orchestrator for ffTC and fbTC by adding/summing both temperature values and feeding it into Flow Service. The degree of fineness could go even deeper by creating an independent service for all controllers and even sensors/actuators. But as mentioned in the introduction this is a tradeoff between granularity and operation cost (see Fig. 1). The Functionality Granularity refers to the physical size of a service.

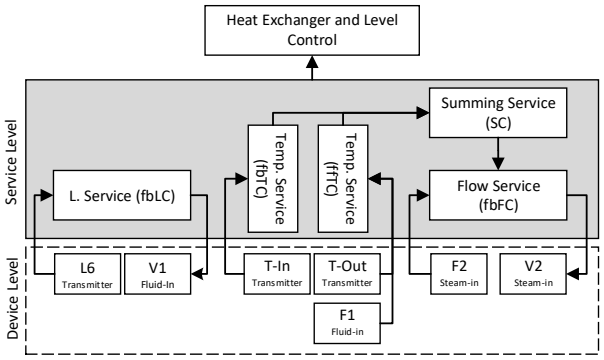


Fig. 5. Functionality Granularity - Five services are exposing independent functionalities. Temp. Service (fbTC) is representing a cascade controller for handling steam pressure issue while Temp. Service (ffTC) is representing a feed-forward controller to handle major disturbances in the process fluid. The summing service is playing an orchestrator role and the flow controller (fbFC) is controlling everything else. The L. Service (fbLC) is controlling the level of vessel.

## V. Quality and Quantitative Metrics

As we mentioned earlier service granularity could be measured based on quality and quantitative parameters.

Quality like flexibility, scalability, maintainability, monitorability, fault tolerability, and reusability. Quantitively like number of services, number of deployments, number of inter and intra service communication, and point of failures.

For example, having a service which handles the entire control or automation process (coarse-grained service) will require only one-time deployment. Note that such service even will not require orchestration or choreography. In contrast to the same process which is being handled by several services (fine-grained services) more deployments are required, and a proper orchestration or choreography need to be in place.

The same applies to monitoring and diagnostic. In the case of a machine controlled by a single service (like in Process Granularity), in the case of service failure the entire machine will be marked as a failed entity. If different parts of the machine are being controlled by different services (like in Machine Granularity or Functionality Granularity), a service failure will in principle only affect the corresponding part of the machine, and there will be more precise diagnosing. From flexibility and scalability point of view, if we have a separate service for different parts of the machine it becomes easier to adapt and evolve the control application when the machine itself is upgraded, applying the plug-and-play concept to the control application itself. Considering this explanation, the idea of the following tables are to summarize some metrics for the different granularity types.

TABLE I. QUALITY METRICS

| Granularities | Metrics |
|---|---|
| **Process Granularity** | High-level monitoring and diagnostic, low fault tolerability, costly maintenance per service, close to scalability and less flexibility, almost no reusability |
| **Machine Granularity** | More detailed monitoring and diagnostic, improved fault tolerability, lower cost in maintenance per service, more flexible and scalable, to some degree reusability |
| **Functionality Granularity** | Very detailed monitoring and diagnostic, high fault tolerability, low-cost maintenance per service, highly flexible and scalable, high reusability |

TABLE II. QUANTITATIVE METRICS

| Granularities | Metrics (values are based on running example) |
|---|---|
| **Process Granularity** | Number of services = 2, number of inter-service communication = 0, number of deployments = 2, single point of failure |
| **Machine Granularity** | Number of services = 5, number of inter-service communication = 4, number of deployments = 5, multipe points of failure |
| **Functionality Granularity** | Number of services = 5, number of inter-service communication = 3, number of deployments = 2, multiple points of failure |

## VI. HYBRID GRANULARITY

The Hybrid Granularity (HG) is referring to the combination of different types of granularities. Often only one type of granularity will not satisfy the entire control and automation system needs, so there will be cases where the combination of two or all types of granularities will be a better fit.

For example, Fig. 7 shows that by combining Functionality Granularity with Machine Granularity we could achieve a higher reliability than applying only Functionality Granularity, and fewer inter service communication when to compared to using only Machine Granularity.

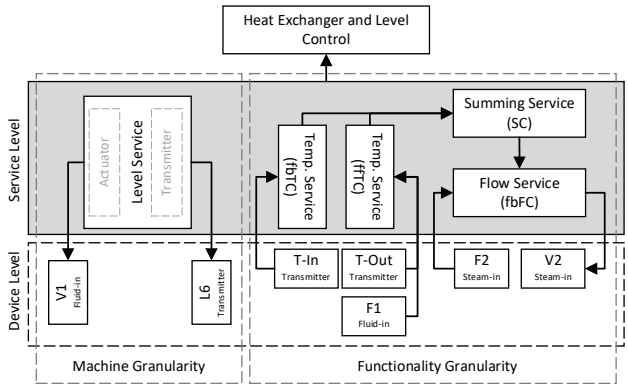


Fig. 7. Seven services are exposing independent functionalities. Temp. Service (fbTC) is representing a cascade controller for handling steam pressure issue while Temp. Service (ffTC) is representing a feed-forward controller to handle major disturbances in the process fluid. The summing service is playing an orchestrator role and the flow controller (fbFC) is controlling everything else. The L. Sensor Service is reading the value of level transmitter and the L. Actuator Service is controlling fluid-in valve after receiving control value from L. Service (fbLC).

Depending on several aspects of a control system and automation needs like criticality, flexibility, security, and reliability, different combination of granularities could be merged to shape the control and automation system in more efficient way.

## VII. ADVANTAGES AND DISADVANTAGES OF GRANULARITY TYPES IN NEW RUNNING EXAMPLE

Following is a new running example that will help to understand each granularity advantages and disadvantages in deeper. It will also bring an opportunity to see how each granularity types will look like in another example. The new example represents a scenario that attracts more attention to fault tolerability.

Fault in automation systems will often result in a hazardous situation and could cause undesired reactions and shutdown. This could be very dangerous for the environment, plant under control, machine, and human. Therefore, Fault Diagnosis and Fault-Tolerant Control are crucial for modern complex control systems [21].

The design of the following block diagram (Fig. 6) is based on fault tolerability and scalability on two major parts of the system. The level controller and the Ph controller. The level controller will ensure the level of vessel will not raise over the level of set-point and the Ph controller will keep the Ph of liquid inside the vessel under the specified set-point. Therefore, it represents five control loops one for controlling temperature, two controllers for level of vessel, and two for Ph of liquid inside the vessel. Note that in total there are three processes that needs to be controlled automatically. These processes are level of vessel, Ph, and Temperature.

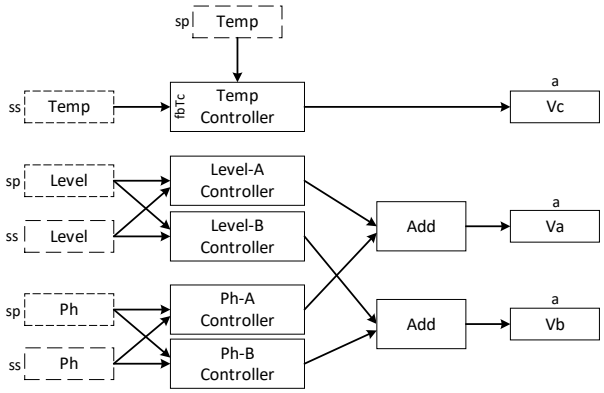


Fig. 6. The above block diagram represents five PID controllers with two adding (summing) functions. The adding functions will total outputs from each level controller with each Ph controller and send a unique signal to each valve (Va for fluid-in and Vb for Ph). Each controller receives a set-point (sp) and a sensor value (ss).

The output of each level controller (Level-A and Level-B) will be totaled with the output of each Ph controller (Ph-A and Ph-B). This is due to the fact that more fluid will increase the level of vessel and consequently will reduce the amount of Ph and accordingly more inlet from the Ph valve will increase the Ph ration and the vessel level.

The overlap between level and Ph processes will require to total output of level controller with output of Ph controller. This is the reason the output from the Level-A controller will be totaled with the output from the Ph-A controller and similarly

the output from the Level-B controller with the output from the Ph-B controller. This will allow to keep fluid-in valve (valve a) and Ph-in valve (valve b) under control during level and Ph increase time. This dependency between processes called **process dependency** which will play a major role in process granularization.

The reason for using extra controllers (two for level and tow for Ph) is to guarantee under any circumstances and failure in one of the level or Ph controllers there will be another instance up and running to keep the system under control.

The other side reason is that having extra control loops will allow scaling control system responsiveness under overloaded occasions. For example, to have a better understanding imagine a limited embedded device where there is no enough process or memory resources to execute two control loops at the same processing device. In such limited computational areas having two independent control instances on auxiliary devices will help to increase computation power and consequently system responsiveness by redirecting requests from overloaded services to idle services.

Following will show how each granularity types will shape control services and will satisfy the intention of the design which is based on fault tolerability and scalability.

### *A. Advantage and Disadvantage of Process Granularity*

Based on Process Granularity (see section IV.A) the Level-A and Level-B controllers will be considered as one process, similarly the Ph-A and Ph-B controllers. Note that since there is a tight dependency (process dependency) between level and Ph controllers is not possible to decouple these two processes from each other in Process Granularity. Therefore, these two processes need to be considered as one coarse-grained process. This will reduce the number of services and the number of inter-service communications. It will also make easier to cut the system in independent pieces. It will also make easier the service deployment process. But this type of granularity will lower the fault tolerability in level and Ph controllers which is the main intention of this design. In the case of any failure in any service all control regarding to that service will be lost and consequently this may lead in a hazardous situation.

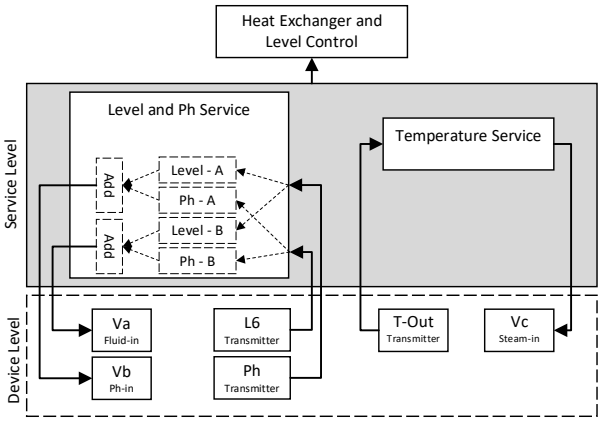


Fig. 9. Process Granularity with two independent services. To control Temprature, Leve, and Ph processes.

The other drawback of this type of granularity is that there is a little chance to have reusability. See the presented example in (Fig. 3). Reusing temperature controller will require to bring the entire Heat Exchanger Service which contains other functionalities like fluid control and summing.

### *B. Advantage and Disadvantage of Machine Granularity*

According to Machine Granularity (see section IV.B) all control loops will be centralized in one service. Note that in Machine Granularity the entire vessel including the valves, and sensors and actuators are considered one machine which then is dividing into different pieces [20]. The following figure which is based on ISA 88 standard demonstrates this idea.

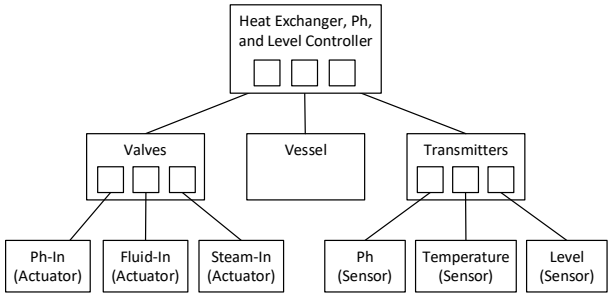


Fig. 8. Machine Hierarchy for Heat Exchanger, Level and Ph Controller.

Maybe the biggest advantage of Machine Granularity is the independent services for different parts of a machine specially the sensors and actuators parts. Because having independent services for different parts of machine will increase overall system responsiveness and will allow to keep the machine to some degree under control during a hazard. However, this type of granularity will not add much value in terms of fault tolerability in level and Ph controllers which is the main intention of the new running example.

From the other side Functionality Granularity, or even combination of Functionality Granularity with Machine Granularity (Hybrid Granularity) will boost system fault tolerability and service (control system) availability ratio significantly by service redundancy and responsiveness.

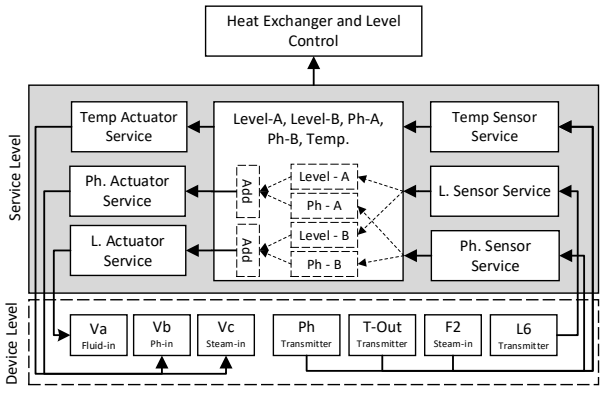


Fig. 10. Machine Granularity with seven independent services that are exposing independent functionalities. Note that all services are running on one device.

### C. Advantage and Disadvantage of Functionality Granularity

Considering Functionality Granularity (see section IV.C), one could create independent services with same functionality for Level-A and Level-B and Ph-A and Ph-B controllers. Note that under normal situations one of each service from level and Ph controllers could be put in an idle mode to save more energy and reduce communication overhead. But under hazardous situation each of the idle services could fulfill the failed service duties. Of course, to aim this degree of availability and fault tolerability some other infrastructures like signal/message routing will also be required. Following Fig. 12 illustrates Functionality Granularity and Fig. 11 represents Hybrid Granularity for the same scenario.

Reusability is another advantage of Functionality Granularity. This means during the implementation phase there is no need to implement each Level or Ph controllers two times. There will be one implementation and then one or more than one instances of the same implementation.

The degree of reusability could even go further by implementing only one controller for both Level and Ph if and only if they have a same controlling algorithm with different set-points or run-time parameters.

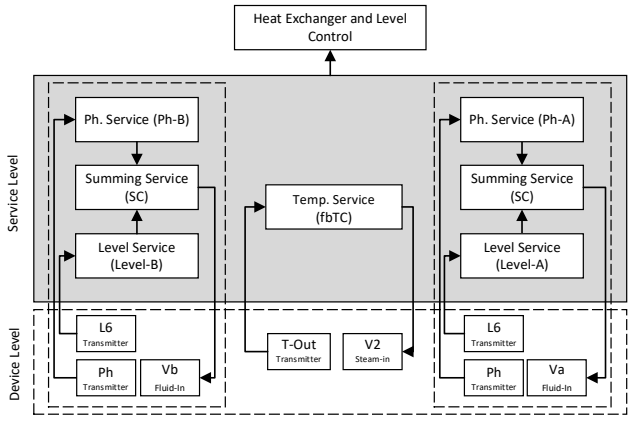


Fig. 12. Functionality Granularity with seven services are exposing independent functionalities. Temp. Service (fbTC) is representing a cascade controller for handling steam pressure. Note that dashed line represents the idle service boundary.

### D. Advantage and Disadvantage of Hybrid Granularity

The last figure is representing Hybrid Granularity (see section VI). Hybrid Granularity will boost fault tolerability and will improve system responsiveness and reusability. The idea is to use Process Granularity, Machine Granularity, and Functionality Granularity.

In the Hybrid Granularity represented in Fig. 11 there are independent services for reading level and Ph sensor values and as well as independent services for controlling Ph-in valve and Fluid-in valve. This is due to Machine Granularity.

Respectively for Level-A, Level-B, Ph-A, and Ph-B exist independent services as well. This represents Functionality Granularity and increases system fault tolerability.

The last service represents Process Granularity for temperature control. However, in this particular example between Functionality Granularity and Process Granularity for temperature service there is no significant difference. But, if the temperature control was similar to the on in the first running example (see Fig. 2) then the difference would be significant.

High fault tolerability and responsiveness are the main advantages of Hybrid Granularity. Additionally, this type of granularity is representing a very flexible and scalable system. For such system it is easy to adapt for environmental changes or scale in terms of number of services by putting some redundant services in an idle mode (see Level-B and Ph-B services in Fig. 11).

The drawback of this granularity is in the number of services and inter-service communications, and it could become expensive in terms of deployment process.

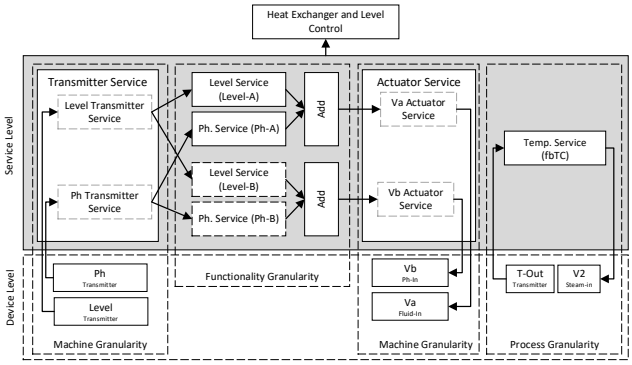


Fig. 11. Hybrid Granularity with nine services that are exposing independent functionalities. Note that services with a dashed line are representing services in idle mode.

## VIII. Conclusion and Future Work

This paper introduced four types of service granularity to provide a better problem formulation for finding a right granularity for each service in industrial control and automation. We discussed how each type of granularity could restructure the control system in architecture and implementation level and compared each type based on qualitive and quantitative characteristics.

We showed that how different granularity types could effect on service-oriented architectures in terms of orchestration vs chorography etc.

However, granularity is not the only challenge that exist in applying service-oriented solution in industrial control and automation. Having fine-grained services may result in other requirements like inter-service locking mechanism that prevents services from overriding each other's commands, a service recovery mechanism that recovers a service from a failure without any interruption in the control process, and a more robust service communication infrastructure that allows adding/removing a new/existing service from the chain of services without causing any failure. Building stateless service is also another challenge which exist in service-oriented industrial control systems.

## References

[1] N. Dragoni et al., “Microservices: yesterday, today, and tomorrow,” pp. 1–17, 2016.

[2] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, “Microservices in Practice, Part 1: Reality Check and Service Design,” IEEE Softw., vol. 34, no. 1, pp. 91–98, 2017.

[3] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, “Microservices in Practice, Part 2: Service Integration and Sustainability,” IEEE Softw., vol. 34, no. 2, pp. 97–104, 2017.

[4] Sam Newman, Building Microservices. O’Reilly Media, 2015.

[5] R. V. Shahir Daya Nguyen Van Duy, Kameswara Eati, Carlos M Ferreira, Dejan Glozic, Vasfi Gucer, Manav Gupta, Sunil Joshi, Valerie Lampkin, Marcelo martins, Shishir Narain, “Microservices from Theory to Practice Creating Applications in IBM Bluemix Using the Microservices Approach,” Ibm, p. 170, 2015.

[6] A. Homay, A. Zoitl, M. de Sousa, and M. Wollschlaeger, “A Survey : Microservices Architecture in Advanced Manufacturing Systems,” in INDIN, 2019.

[7] A. Homay, A. Zoitl, M. De Sousa, M. Wollschlaeger, and C. Chrysoulas, “Granularity cost analysis for function block as a service,” IEEE Int. Conf. Ind. Informatics, vol. 2019-July, pp. 1199–1204, 2019.

[8] V. Gabrel, M. Manouvrier, and C. Murat, “Web services composition: Complexity and models,” Discret. Appl. Math., vol. 196, pp. 100–114, 2015.

[9] G. N. Rai, G. R. Gangadharan, and V. Padmanabhan, “Algebraic modeling and verification of Web service composition,” Procedia Comput. Sci., vol. 52, no. 1, pp. 675–679, 2015.

[10] R. Hewett, P. K. Sanayothin, and B. Nguyen, “Scalable optimized composition of web services with complexity analysis,” 2009 IEEE Int. Conf. Web Serv. ICWS 2009, pp. 389–396, 2009.

[11] A. Khoshkbarforoushha, R. Tabein, P. Jamshidi, and F. Shams, “Towards a Metrics Suite for Measuring Composite Service Granularity Level Appropriateness,” 2010.

[12] W. Xiao-jun, “Metrics for Evaluating Coupling and Service Granularity in Service Oriented Architecture,” 2009 Int. Conf. Inf. Eng. Comput. Sci., pp. 1–4, 2009.

[13] H. Jain, H. Zhao, and N. R. Chinta, “A Spanning Tree Based Approach to Identifying Web Services,” Int. J. Web Serv. Res., vol. 1, no. 1, pp. 1–20, 2004.

[14] H. Bloch et al., “A microservice-based architecture approach for the automation of modular process plants,” IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA, pp. 1–8, 2018.

[15] S. Boukharata, A. Ouni, H. Wang, M. Kessentini, and S. Bouktif, “Improving web service interfaces modularity using multi-objective optimization,” Autom. Softw. Eng., vol. 26, no. 2, pp. 275–312, 2019.

[16] E. Evans, “Domain-Driven Design: Tackling Complexity in the Heart of Software: Amazon.de: Eric J. Evans: Fremdsprachige Bücher,” vol. 7873, no. 415, p. 529, 2003.

[17] P. C. Fundamentals, “Instrumentation & Control.”

[18] F. O. R. Use et al., “Batch Control Part 1 : Models and Terminology,” 2009.

[19] OMG, “OMG Unified Modeling Language Specification vertion 1.3,” no. March, 1999.

[20] C. Sünder, A. Zoitl, M. Rainbauer, and B. Favre-Bulle, “Hierarchical control modelling architecture for modular distributed automation systems,” 2006 IEEE Int. Conf. Ind. Informatics, INDIN’06, vol. 00, pp. 12–17, 2006.

[21] V. Yang, Hao, Jiang, Bin, Cocquempot, “Fault Tolerant Control Design for Hybrid Systems | Hao Yang | Springer,” 2010.