Some Means Supporting Responsibility and Openness in Information and Control Systems

Michal Žemlička[1,2,3(✉)] and Jaroslav Král[4]

[1] Faculty of Science, Jan Evangelista Purkyně University, Pasteurova 1, 400 96 Ústí nad Labem, Czech Republic
michal.zemlicka@ujep.cz
[2] Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
zemlicka@sisal.mff.cuni.cz
[3] AŽD Praha, Závod Technika, Žirovnická 2/3146, 106 17 Praha 10, Czech Republic
zemlicka.michal@azd.cz
[4] Faculty of Informatics, Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic
kralq@seznam.cz

Abstract. The structure of many current systems is independent of related responsibility borders. It comes from the fact that they are designed by programmers that as a rule give more importance to efficient data moving than to reflecting existing business and control processes and to precise personal or organizational responsibility. It results in non-trivial issues when looking for reasons and responsibility for improper system behaviour and its overall quality. Another serious issue is that the requirements on the systems are changing during their development and use. Accommodation to the changes often introduces new serious issues to the systems. Proper support for responsibility and flexibility is a necessary precondition for solving the issues or for recovery from them. We cope with those issues by proposing software architecture simplifying alignment of software with responsibility domains. The solution is enabled by a proper application of specific software service types described here. The proposed solution significantly improves system and organizational flexibility and openness and brings many other interesting engineering advantages. The proposed solution is based on observations from human behaviour and roles within organizations and their cooperating groups.

Keywords: System flexibility · Organizational responsibility · Software architecture · System openness · Architectural patterns · Processes in SOA · Responsibilities of users · Middleware

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12254, pp. 895–909, 2020.
https://doi.org/10.1007/978-3-030-58817-5_63

1 Introduction

Software parts covering logically closed functionalities of information and control systems (SPICS) grow and become continuously more complex. The requirements on their functions and services and quality grow. It is known that such systems cannot have a monolithic architecture if they should be developed in time and be maintainable (compare the consequences of software engineering metrics [4,5,25] and quality characteristics – see e.g. [15]).

SPICS must have a proper architecture also for further reasons. If a SPICS is very large, then its development, maintenance, and use are difficult tasks for many experts and users (e.g. user domain IT experts, managers, IT maintainers, users, or administrators).

It follows that such SPICS must have a service-oriented architecture. SPICS should have the structure of a network of autonomous units, services, exchanging understandable messages. The messages have a form similar to the form of digitalized documents or messages similar to inter-human communication used for cooperation and communication between corresponding real-world services.

The understandability, as well as usability, are crucial aspects of SPICS – see e.g. [23]. Understandability of messages implies that the messages used in communication of services are textual and gradually have more complex structure (given by a variant of syntax). It enables many system quality aspects and simplifies balancing them. Examples are aspects discussed below.

Understanding of the notion "service" used here is different from the proposals by W3C [35], Erl [9], and others. It comes from the origin of the different versions of service orientation – compare [26]. The proposed version of service-orientation combines experience collected by the authors mainly in the domain of structured systems [37,41] and control systems [17]. It could be used to handle current issues in the development of information and control systems. The applicability spans from e-government [19] to huge control systems like traffic management systems covering large areas [42]. It is possible to design a system so that it can implement important features Responsibility and Openness (see below).

The service orientation mentioned here can work in a specific way: The messages could be redirected to other service or to manual processing (if necessary). The message addressee(s) could be defined by sender or selected according the message body and according available potential processors (software services or human beings).

Human involvement enables the use of information available to people but not available to software (e.g. it could be hard to enter it or it is not programmed yet to handle such information). In some cases, human involvement is a necessity for successful solutions of many problems to which the owners of the systems could face. We will, therefore, focus on definition, control, and solving issues related to responsibility of individual users and their groups, as well as of other people.

Our proposal is based on structuring a class of software systems according functional and responsibility borders. The presented architecture is supported by specific services supporting and enforcing its policies.

The paper is structured as follows: Sect. 2 concerns responsibility and its selected aspects, Sect. 3 Openness. Security is discussed in Sect. 4. How these observations should influence information and control systems design and use is described in Sect. 5. Finally, Sect. 6 gives conclusions.

2 Responsibility

2.1 Responsibility Identification

It holds in many countries that a human being or an institution could bear responsibility only for that matters he, she, or it can influence. With the appearance of information and control systems, there appeared also situations when the decisions were not made by human beings but by computers. It raised a problem who is then responsible for such decisions: the owner of the computer, the owner of the software, the author of the software, or even someone else?

Structure of many contemporary systems do not lower insistency of this question: the decisions are made using so complex processes divided into many parts provided by various software artefacts created by different vendors. It is then very difficult (if not impossible) to determine, who or what part of the system is responsible for a given decision. The appearance of the cloud and to it related technologies make such task even more complicated. Often it can be a problem even to determine in what country the individual decisions have been made.

Responsibility is important not only in the sense of identification of criminal liability of individual parties, but especially in the context of (sense of) control over given service, opportunity to adapt it, and especially for the opportunity to understand it properly – many of contemporary systems do not allow to understand them and to influence how or what they (the systems) process.

2.2 Responsibility and System Comprehension

Full computerization of systems is extremely difficult to achieve (if the resulting product should be really of high quality) comparing to cooperational computerization where people and computers cooperate [15]: It must be estimated ahead what all could happen and tried to solve it – and all such situations must be not only analysed but also proper solutions must be developed and exhaustively tested for them.

The developers could work arbitrarily perfect but the world is so complex that for sufficiently large or complex systems it could always appear a situation what has not been expected or that even could not be expected (e.g. significant new developments or observations are often not expected). It, therefore, makes sense to allow properly trained and domain knowing people to influence the system, if necessary—or to take control over its parts or even over the entire system itself.

On behalf of fulfilling the functions of the system, the people should have opportunity to control and to influence run of the system (they can get to information important for a proper run of the system that could not be entered into the system) or to take control over the selected part of the system (bugs, accidents, or catastrophes could arise any time). We know from our experience that an opportunity to take over functions of selected modules could be advantageous also for system development, its incremental deployment, or its use under more difficult conditions [22].

It is advantageous to use the above-mentioned observation already during system development. Individual parts of the system should be therefore equipped by interface allowing supervision and conditionally also control over the given part. Individual parts should be similarly equipped by such interface that could provide communication with other parts of the system as well as with people providing the functionality of the others.

2.3 Responsibility Delegation

Responsibility is in the real world usually bound to institutions or their clearly defined parts, respective to given human beings – either due to their function in the organization or as individuals. Systems that respect this responsibility should have borders reflecting the corresponding responsibility borders.

Responsibility could be in some cases of hierarchical nature (for something it is responsible entire organization, for something its division, for something its department, and for something exact person). In such cases (and probably not only then) it makes sense to decompose system supporting such organization into parts corresponding to the organizational breakdown of the organization.

Contemporary information and control systems often do not follow such schema. They are usually structured according to layered architecture with a limited number of layers (the classic ones have 2–3 layers, the newer ones usually up to 9—compare e.g. [6,29]).

2.4 Conditions for Responsibility Taking

Responsibility also requires that the one who should be responsible for some system should have an opportunity (either personally or using other people) to convince itself that given system works properly. In the opposite case, it would be no responsibility, it would be a hazard. Taking responsibility is easier if it is possible in an emergency or other unexpected or rare cases to take control of the entrusted task. It allows coping also with situations that have not been taken into account, whose solution would be too expensive, or that even could not be taken into account.

2.5 Responsibility Issues

Responsibility is in full extent a very general concept. Being aware of this we start with coping with some related issues and hoping for solving the others

later. Responsibility of users is a very complex problem, compare rules related to security and responsibilities needed to use clouds in SOA processes.

3 Openness

Under the notion of openness, we understand an opportunity to extend the system easily by new parts or functions or to connect the system with other systems without a necessity to modify the system significantly. Different systems can be created using different methods, can be based on different approaches, and can use different communication protocols. Direct interconnection of such systems without their previous modification is as a rule not performable or is applicable only exceptionally. Sometimes it is even a very hard task to understand how the partner system works – not to speak of how to identify parts that must be involved in the required cooperation.

Real-world services, to what the systems serve, simply not only can cooperate but often must cooperate. There are often involved more than two services in such cooperation.

It is given by the fact that interfaces of real-world services are primarily problem-oriented instead of being implementation-oriented as it is by many information or control systems. Hence on the interface, it is specified what should be done, not how – not to speak about solving technical details. Such real-world interfaces are already in use for a very long time – years, decades, or even longer. We suppose that it makes sense to let ourselves to inspire by such interfaces when designing interfaces of software systems playing a similar role. Understandability of communication improves supervision of many aspects of quality especially security and safety.

4 Security

Security has many aspects. We focus here on restricting access only to agreed partners (mostly to their systems) and their reasonable behaviour at all system levels. Precise access control is here combined with natural borders of responsibility what also improves the chance that people will be more likely to respect other security measures. We suppose that reasonable access control could be based on the following principle: the access to the system should be by design limited to partners that are planned (e.g. if they have signed an agreement with the institution running the system).

It is, the access to the system must be controlled so that it could be efficiently blocked if necessary. It is also reasonable if the access of individual partners to the system could be controlled independently. Then if any of the partner systems starts to behave incorrectly (e.g. producing too much traffic), it is possible to block such communication without a necessity to restrict other partner systems. Communication in such structure could and should be authorized and protected by corresponding encryption methods.

Information and control systems face many security threats. Many papers are identifying at least some of them and trying to cope with them – compare e.g. [8,27,32]. Some authors tried to classify the threats. Let us mention at least Jouini et al. [16] or Gerić and Hutinsky [11].

5 Impacts

Let us look at how fulfilling the above quality aspects could reflect on the structure of the system.

5.1 Security

Cooperation with other systems must be under control. Required openness without support from the security side could be hazardous. Therefore both aspects must be solved side-by-side.

To control individual access from individual partners we can equip the system with special autonomous services – *access points* (AP). The situation is depicted in Fig. 1.

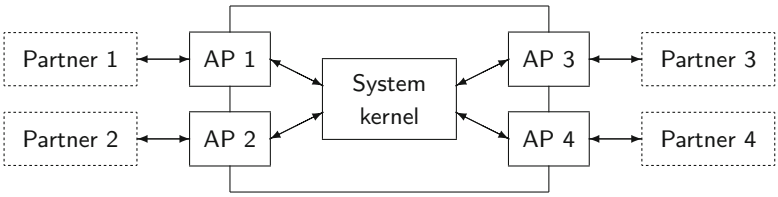


Fig. 1. System with distinguished access points

It makes sense to connect AP on both sides to different networks. The inner interface should be connected to the internal network of the company or its unit, the outer to a dedicated link between the partner and the AP or to an at least encrypted channel through a public network.

The AP's appear when necessary (e.g. after signing a cooperation contract between institutions) and disappear when it is not necessary or when the protected (sub)system gets under attack (e.g. when a system of the partner or the channel do not work as agreed). It is reasonable to have a separate AP for each contract as it allows precise control over the connection and forces the partner not to reveal the connection data to third parties.

Access points could be used also for precise control over a communication of individual parts of a system even within a single organization. It could then control explicit access to subnetworks of individual organizational subunits. Such a solution can improve the chance that attacking one part of the system does not affect the entire system.

It could happen that there will be established multiple access points for cooperation with the same partner. Such an approach could separate individual agreements between the cooperating organizations. It is also possible and could be reasonable to establish multiple AP by a single agreement. The individual AP's then can separate cooperation between the institution according to additional attributes like access rights on the partner side, communication style (e.g. stream vs. messaging), etc.

An access point could be treated as a smart plug: it allows to connect a partner and check the validity of the connection (the partner can establish the connection – e.g. using a proper key) and can establish a connection to its counterpart (to log to partner's system). Access points are expected to serve for a given period only – according to the time-span of the related agreement between partners. As the agreements could be renewed/extended, the time limitation of the access points should be also possible.

Access points are expected to separate connections to the partners from the inner network of the organization. They route the traffic from the partner only to desired parts of the organizational systems and support sending requests to the partner system if necessary.

The use of access points (and other architectural services described below) reduces the severity of many security threats. As there are multiple system access points (separate for individual partners), it is harder to block access to the system using distributed denial of service (DDoS) attacks. Moreover, using dedicated connections gives more responsibility to the partners for the systems connected to them.

The incoming communication is processed by relatively simple services. Taking control over them (or their disabling) still gives only very limited opportunity to make harm to the rest of the system as well as only very limited access to the processed data. The real processing and storing of data is in the system kernel (its services/applications) so there is a chance for system administrators to detect attacks on the outer services and take relevant measures sooner than the attack gets to the critical parts of the system.

So, systems described here could resist at least partially to most known security threats caused by external subjects. As individual agendas are usually handled by different parts of the system that could be also protected by access points, there is a chance that even insiders could have only limited impact.

5.2 Flexibility

The responsibility is usually taken for assurance of some service or task to be provided. Sometimes once, sometimes for a given number of iterations or for some period. The environment and other conditions could change during time.

Fulfilling the responsibility could be conditioned by a proper extent of flexibility of the task to be able to cope with the changes (and still fulfil for what the person or organization is responsible). It could happen that some subtask can be re-delegated to other partner or that partners' systems could change, etc. To simplify accommodation to the above-mentioned changes, it is possible to

perform the cooperation with partners using some high-level stable interface—preferably the one that is understandable for professionals in a given domain, resp. topic.

Such interface need not be the same as the interface of the system supporting or providing the requested functionality. It is possible to translate an existing system interface to the interface made available for cooperation with partners. Moreover, it is possible and reasonable support partners with different needs using different interfaces. The interface translation could be provided by special autonomous services called *front-end gates* (FEG; introduced in [20]). Front-end gates could be developed using techniques developed for compilers (parsers, transducers, compiler back-ends [1,2,12,39]) or using XML transformation (XSLT [34], XPath [36]). The situation then can look like in Fig. 2.

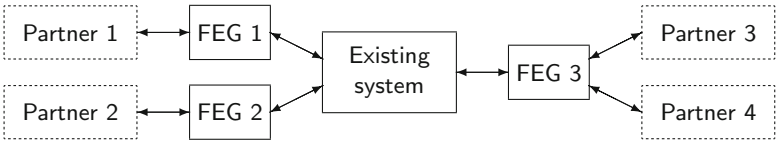


Fig. 2. Supporting flexibility by front-end gates (interface translators)

In practice, it makes sense to combine both flexibility and security provided by front-end gates and access points to support both tailored interfaces and tailored access—compare Fig. 3.

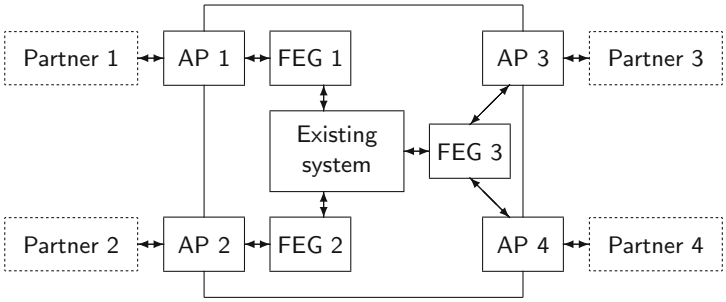


Fig. 3. Customized communication channels using front-end gates and access points

The access point(s) then ensure that only given partner is communicating given channel whereas front-end gate(s) support the partner-specific communication protocol. When multiple partners are using a very similar user interface provided by the same front-end gate, it is possible to let the access points restrict the supported interface (e.g. to inhibit the use of improper values). On the other hand, it could be possible to let one access point provide access to a single partner to multiple front-end gates.

5.3 System Decomposition and Composition

As stated above, the structure of the system should be able to follow a breakdown of the responsibility and organizational breakdown. The flexibility provided by front-end gates as well as partial security support provided by access points could be used at any level of decomposition.

To keep the system structure clean it is reasonable to treat parts at any level as black boxes. It is, all their outputs and inputs must go through a selected entry point.

A closed group of cooperating services having concentrated inputs and outputs to and from the services outside the group into single service we call *composite service* (CS) and the distinguished service *head of composite service* (HCS) or *head* for short. Composite services could look like the one on Fig. 4.

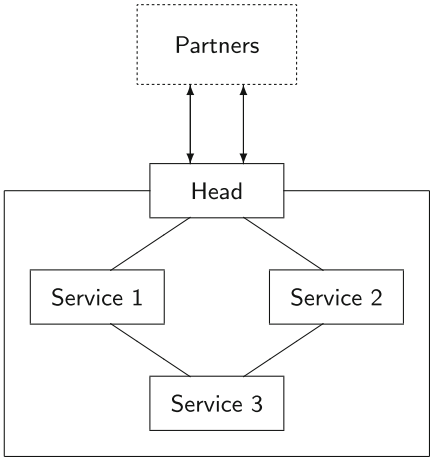


Fig. 4. Composite service – a simplified view; taken from [43]

HCS (its outer interface) behaves to related front-end gates like a primary gate in the case of simple services. The inner interface of an HCS behaves to services in the composite group as one of the partners – it is, it can be protected (as well as their front-end gates) by its access point(s). From the network point of view, the role of HCS is very similar to the role of a router in communication networks.

The composite service often conforms to an organizational subunit or another responsibility unit. It should, therefore, fulfil similar requirements as the system as a whole: it should be open (having flexible interfaces) and respect responsibility border (by supporting distinguished access control). The composite service could and should be then equipped by front-end gates as well as by access points. The same holds also for services forming the composite service. The resulting situation can look like in Fig. 5.

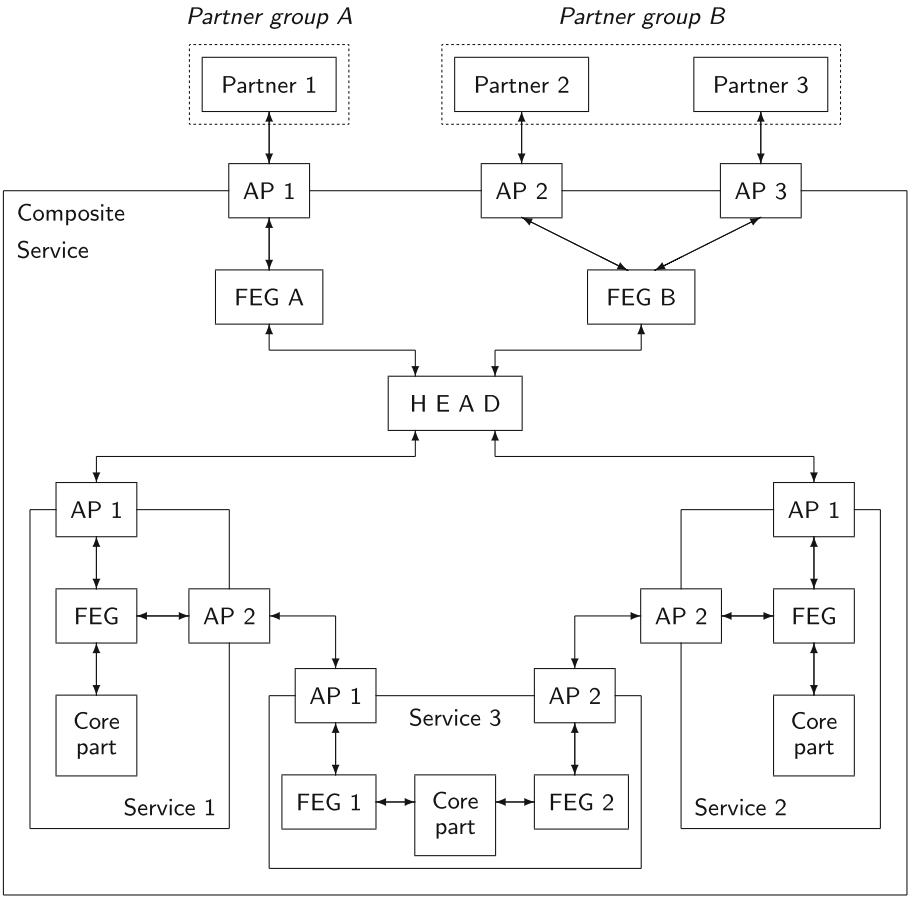


Fig. 5. More detailed view on the composite service from Fig. 4

Composite services can be composed of simple services as well as of other composite services. We can create or model this way hierarchical structure following organizational or responsibility structure of a supported organization. Let us note that even the single services are systems used to support the activity and could be still complex applications.

All services in the system could have (and often should have) user interfaces allowing to supervise their work or to take manual control or replacement over them. It is possible to do some work manually and the rest of the work could be let on the supporting applications. For details see [22].

5.4 Processes

In the real world, activities are performed in business or control processes. They control the cooperation of several actors to produce the desired output.

Business processes could be hard-coded in the services (they could know whom to ask to perform some subtask or whom to give the work to continue). It is possible in the case of very stable and already optimized business processes known ahead. The business processes must be described in detail as there is no chance to modify them or to make exceptional processing.

When the processes are not so stable, they should be driven more flexibly: by scripting, by a roadmap defined in an additional document, or by special services controlling the process run – *business process managers* (BPM; see [21,24]). Scripting adds an opportunity to change the business process a bit faster than when they are hardcoded. But scripts are programs and as such, they should not be modified when the business or control process is already running.

The attached document (forwarding bill) driven business process is quite popular in many institutions. The flexibility of so driven business process depends on the rights that the users have to the forwarding bill—who can modify it or how flexible the instructions stored in it could be interpreted (in some cases the requested signature from the boss could be replaced by some vice-bosses probably after consultation with the boss—if the boss is physically not available). This kind of handling of some business processes is used for many decades and many people can use them intuitively. Some practical experience of their application is described e.g. in [18].

There are multiple business process notations [3,7,14,31,38,40]. They were developed for various conditions. It is therefore reasonable if the services providing business process management could handle various business/control process notations. Such services should provide an interface to the responsible person – process owners. It is advantageous if the user interface is customizable—there is a bigger chance that it will be more intuitive for their users (process owners).

In some cases, the business process manager could play the role of head of composite service. In the case when a business process must be run across multiple organizational units, it is reasonable that the business processes could be partitioned according to organizational structure: individual sub-processes will be run in the organizational subunits and the overall part will be managed at the lowest common level. It could be often reasonable if the sub-process' managers report the process status to the upper business process manager.

5.5 System User Interfaces (Portals)

Complex systems usually support not only local interfaces to individual parts but also interface(s) to the system as a whole. The system is typically used by different user groups. It is therefore reasonable to equip the system with corresponding user interfaces fulfilling and matching the needs and rights of the user groups. Most systems distinguish at least between more trusted users (and support them by an intranet interface) and general users (supported by an extranet interface).

The services supporting such an interface (and playing to some extent also the role of access points) are called portals. As they can be based on web technology, their name could suggest usable technology.

5.6 Remarks

Services creating system structure like front-end gates, access points, or heads of composite services are important from the architectural point of view. They are therefore called *architectural services*. They are, as a rule, newly developed or customized for a given system. Moreover, as other services in the system, they can be equipped by user interface allowing responsible people to supervise them or to make some decisions manually. The supported activities could include also message redirection, transformation or even execution. It corresponds to forwarding improperly addressed requests or to transforming improperly formulated requests in human organizations. Such an opportunity is advantageous for debugging or in rare or emergency cases.

The combination of front-end gates and access points can someone remind on Façade [10]. The proposed pattern focuses on services instead of on objects, it is more restrictive in control over accessing the original interface and it could support human supervision if necessary.

It makes sense to structure the underlying networking so that the architecture services could fully play its role. It is, if an architectural service should be a unique connection to some part of the system, it should be so also from the networking point of view. It is especially important at the security borders originated from responsibility borders (like at borders between organizations) or where subsystems with different security or safety should cooperate.

Architectural services can be seen as special microservices [13,33] being glue for application services that could be large and complex services or again microservices.

6 Conclusions

Responsibility and its borders are significant for the structure of an information system. It is often neglected (yes, even now there are created standards ignoring and blocking it).

When we reflect the responsibility borders in the way described above, we can improve many interesting features like openness, flexibility, maintainability, or distributed development. It could be related to the fact that it corresponds to the recommendation of information hiding [30]. Moreover, the learning curve could be flattened as there is used language and formats understandable and intuitive to the users.

The proposed service-oriented architecture significantly differs from most service-oriented architectures described in the literature—compare [6,9,29] and others. The hierarchical system structure reflects the usual organization structure where managers control only a limited number of subordinates (people or organizational subunits). It simplifies understanding and utilizes already gained knowledge and skills of managers of various levels.

Such system structure to some sense reflects also observations of Miller [28] or information hiding recommendation of Parnas [30].

The impacts of user responsibility to system architecture must be further studied – especially in the frame of dependence between the development team and resulting system structure. It seems that clients' organizational structure should be reflected by the structure of the system and it should be reflected in the structure of the developer's team. It is a hypothesis that should be proven yet.

We believe that the problem of responsibility in processes can be solved gradually starting from current practices, standards, and patterns. Some practices must be used in parallel or combined with new solutions. It is not easy to do it as it has features of a new philosophy.

It appears to be reasonable to reflect the responsibility of user subunits or individuals into system structure and reflect it to the structure and responsibilities of the development team. The flexible interfaces provided by the combination of front-end gates handling protocol flexibility and access point handling access flexibility appear to be a good background for it.

Handling responsibility of end-users in their current roles is an important issue. There are strong security issues. The system design must solve the responsibilities of end-users (represented by their roles). It must be taken into account that the responsibilities of individual users vary in time. The responsibilities could be assigned and unassigned. There must be collected experience with end-user roles during the system lifetime (and reflected in system maintenance). There should be solved complex security issues enabled e.g. by human involvement. A reasonable solution is a topic of further research.

References

1. Aho, A.V., Ullman, J.D.: The Theory of Parsing, Translation and Compiling, Vol. I.: Parsing. Prentice-Hall, Englewood Cliffs (1972)
2. Aho, A.V., Ullman, J.D.: The Theory of Parsing, Translation and Compiling, Vol. II.: Compiling. Prentice-Hall, Englewood Cliffs (1973)
3. Andrews, T., et al.: Specification: Business process execution language for web services version 1.1 (2003). http://www-106.ibm.com/developerworks/library/ws-bpel/. Last Accessed Jan 2005
4. Boehm, B.W.: Software Engineering Economics. Prentice Hall, Upper Saddle River (1981)
5. Boehm, B.W., et al.: Software Cost Estimation With COCOMO II. Prentice Hall, Upper Saddle River (2000)
6. Brown, P., Estefan, J.A., Laskey, K., McCabe, F.G., Thornton, D.: Reference architecture foundation for service oriented architecture version 1.0, December 2012. http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.pdf
7. Business Process Management Initiative: Business process modelling notation (2004). http://www.bpmn.org/
8. Cîrnu, C.E., Rotună, C.I., Vevera, A.V., Boncea, R.: Measures to mitigate cybersecurity risks and vulnerabilities in service-oriented architecture. Stud. Inform. Control **27**(3), 359–368 (2018)
9. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, Upper Saddle River (2005)

10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns Elements of Reusable Object-Orieneted Software. Addison-Wesley, Boston (1993)
11. Gerić, S., Hutinski, Ž.: Information system security threats classifications. J. Inf. Organ. Sci. **31**(1), 51–61 (2007)
12. Grune, D., van Reeuwijk, K., Bal, H.E., Jacobs, C.J.H., Langendoen, K.: Modern Compiler Design, Second edn. Springer, New York (2012). https://doi.org/10.1007/978-1-4614-4699-6
13. Hasselbring, W., Steinacker, G.: Microservice architectures for scalability, agility and reliability in e-commerce. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 243–246 (2017). https://doi.org/10.1109/ICSAW.2017.11
14. IDS Scheer: Aris process platform. http://www.ids-scheer.com/international/english/products/31207
15. International Organization for Standardization, International Electrotechnical Commission: ISO/IEC 25010:2011 systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - system and software quality models (2011). https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en
16. Jouini, M., Rabai, L.B.A., Aissa, A.B.: Classification of security threats in information systems. In: The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014). Procedia Computer Science, vol. 32, pp. 489–496 (2014). https://doi.org/10.1016/j.procs.2014.05.452, http://www.sciencedirect.com/science/article/pii/S1877050914006528
17. Král, J., Demner, J.: Towards reliable real time software. In: Proceedings of IFIP Conference Construction of Quality Software, pp. 1–12. North Holland (1979)
18. Král, J., Novák, P., Žemlička, M.: A system based on intelligent documents. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10409, pp. 176–187. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62407-5_12
19. Král, J., Žemlička, M.: Electronic government and software confederations. In: Tjoa, A.M., Wagner, R.R. (eds.) Twelfth International Workshop on Database and Experts System Application, pp. 383–387. IEEE Computer Society, Los Alamitos (2001). https://doi.org/10.1109/DEXA.2001.953091
20. Král, J., Žemlička, M.: Component types in software confederations. In: Hamza, M.H. (ed.) Applied Informatics, pp. 125–130. ACTA Press, Anaheim (2002)
21. Král, J., Žemlička, M.: Implementation of business processes in service-oriented systems. In: 2005 IEEE International Conference on Services Computing (SCC 2005), vol. 2, pp. 115–122. IEEE Computer Society (2005). https://doi.org/10.1109/SCC.2005.58
22. Král, J., Žemlička, M.: Software architecture for evolving environment. In: Kontogiannis, K., Zou, Y., Penta, M.D. (eds.) 13th IEEE International Workshop on Software Technology and Engineering Practice, pp. 49–58. IEEE Computer Society, Los Alamitos (2006). https://doi.org/10.1109/STEP.2005.25
23. Král, J., Žemlička, M.: Usability issues in service-oriented architecture. In: Cardoso, J., Cordeiro, J., Filipe, J. (eds.) ICEIS 2007: Proceedings of the Ninth International Conference on Enterprise Information Systems, vol. DISI, pp. 482–485. EST Setúbal, Setúbal (2007). https://doi.org/10.5220/0002387704820485
24. Král, J., Žemlička, M.: Implementation of business processes in service-oriented systems. Int. J. Bus. Process Integr. Manage. **3**(3), 208–219 (2008). https://doi.org/10.1504/IJBPIM.2008.023220

25. Král, J., Žemlička, M.: Inaccessible area and effort consumption dynamics. In: Dosch, W., Lee, R., Tuma, P., Coupaye, T. (eds.) Proceedings of 6th International Conference on Software Engineering Research, Management and Applications (SERA 2008), pp. 229–234. IEEE CS Press, Los Alamitos (2008). https://doi.org/10.1109/SERA.2008.27
26. Quintela Varajão, J.E., Cruz-Cunha, M.M., Putnik, G.D., Trigo, A. (eds.): CENTERIS 2010. CCIS, vol. 110. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16419-4
27. Li, Z., Liu, K.: An event based detection of internal threat to information system. In: Kim, J.H., Geem, Z.W., Jung, D., Yoo, D.G., Yadav, A. (eds.) ICHSA 2019. AISC, vol. 1063, pp. 44–53. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-31967-0_5
28. Miller, G.A.: The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychol. Rev. **63**, 81–97 (1956). https://doi.org/10.1037/h0043158
29. Open Group: Open Group standard SOA reference architecture, November 2011. https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=12490
30. Parnas, D.L.: Designing software for ease of extension and contraction. IEEE Trans. Softw. Eng. **5**(2), 128–138 (1979). https://doi.org/10.1109/TSE.1979.234169
31. Pokorný, J.: Workflow management systems: a survey of possibilities. In: Coelho, J.D., Jelassi, T., König, W., Krcmar, H., O'Callaghan, R., Sääksjärvi, M. (eds.) Proceedings of the Fourth EuropeanConference on Information Systems. ECIS1996, Lisbon, Portugal, pp. 253–264 (1996)
32. Safianu, O., Twum, F., Hayfron-Acquah, J.B.: Information system security threats and vulnerabilities: evaluating the human factor in data protection. Int. J. Comput. Appl. **143**(5), 8–14 (2016). https://doi.org/10.5120/ijca2016910160
33. Taibi, D., Lenarduzzi, V.: On the definition of microservice bad smells. IEEE Softw. **35**(3), 56–62 (2018)
34. W3 Consortium: XSL transformations (XSLT) version 2.0, January 2007. http://www.w3.org/TR/xslt20
35. W3 Consortium: Web of services (2015). https://www.w3.org/standards/webofservices/
36. W3 Consortium: XML path language (XPath) version 3.1, March 2017. https://www.w3.org/TR/xpath/
37. Weinberg, V.: Structured analysis. Prentice-Hall software series, Prentice-Hall, Englewood Cliffs (1979)
38. WFMC: Workflow management coalition home page (1993). http://www.wfmc.org/
39. Wilhelm, R., Maurer, D.: Compiler Design. International Computer Science, Addison-Wesley, Wokingham (1995)
40. Workflow Management Coalition: Workflow specification (2004). http://www.wfmc.org/standards/docs/Wf-XML-11.pdf
41. Yourdon, E.: Modern Structured Analysis, 2nd edn. Prentice-Hall, Upper Saddle River (1988)
42. Zanella, G.L., Pinasco, M.: Deliverable d8.1: Requirements for the integration layer (Sep 2016). http://www.in2rail.eu/download.aspx?id=aef8ad3a-8581-4f56-be5c-77d6329dd7f4
43. Žemlička, M., Král, J.: Software architecture enabling effective control of selected quality aspects. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11623, pp. 536–550. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24308-1_43