Towards the Architecture of a Decision Support Ecosystem for System Component Selection

Jakob Axelsson[1, 2], Ulrik Franke[1], Jan Carlson[2], Séverine Sentilles[2], Antonio Cicchetti[2]

[1] Swedish Institute of Computer Science (RISE SICS)
SE-164 29, Kista, Sweden

[2] Mälardalen University
SE-721 23 Västerås, Sweden

***Abstract*—When developing complex software-intensive systems, it is nowadays common practice to base the solution partly on existing software components. Selecting which components to use becomes a critical decision in development, but it is currently not well supported through methods and tools. This paper discusses how a decision support system for this problem could benefit from a software ecosystem approach, where participants share knowledge across organizations both through reuse of analysis models, and through partially disclosed past decision cases. We show how the ecosystem architecture becomes fundamental to deal with efficient knowledge sharing, while respecting constraints on integrity of intellectual property. A concrete architecture proposal is outlined, which is a web-based distributed system-of-systems. Experiences of a proof-of-concept implementation are also described.**

Keywords—Software ecosystems; software architecture; decision support systems; software components; system-of-systems.

I. Introduction

It is nowadays common practice to develop software-intensive systems based on existing software components and frameworks, in combination with additional components that are tailored for the application in question. The components can come from many different sources, including internal or outsourced development, commercial off-the-shelf, and for software also open source communities. They can also have varying characteristics and interfaces, as well as requirements on the context they can be used in. All this makes the decisions about what components to use very complex, involving not only technical, but also business and organizational perspectives, and life-cycle considerations. The decisions are critical to the success of a system development project, and yet they have to be made with very little available information. In practice, the decisions are often made ad hoc, with little use of structured methods. A number of persons are usually involved, which adds challenges related to the interplay between humans in the context of systems engineering processes [2].

Component selection would thus benefit from decision support tools that guide designers through the decision process in an efficient way. Such guidance would lead to decisions of higher quality, i.e., the decisions would have a lower risk of getting torn up later on, or of being suboptimal. Ideally, the decisions should be based on facts rather than guessing, and knowledge and evidence should therefore be used by this tool. These ideas have led to the ongoing ORION research project where a goal is to develop a support tool called COACH (Component Options Analysis in Cooperation with Humans).

One of the key ideas in COACH is to use knowledge to support decision makers, but a critical aspect is how to actually acquire the necessary evidence on which that knowledge will be built. The data could, at least partially, come from previous decision cases. However, it does not appear realistic that an individual organization will make sufficiently many similar decisions on their own to build up a knowledge base of the necessary size that can lead to good decision recommendations. Therefore, we have investigated if a software ecosystem approach could be used, in which different users and organizations can share knowledge between each other in order to make decisions more efficiently and effectively.

The contribution of this paper is to address the research question "What would be a suitable software architecture to support an ecosystem for decision support?" This has been investigated using a design science methodology [15], and the major steps have been to elicit requirements; perform architectural analysis, resulting in a description based on the ISO 42010 standard [18]; and develop a proof-of-concept prototype to validate the architectural description. To our knowledge, the idea of using an ecosystem approach to decision support is novel and lacks previous research.

The remainder of the paper is structured as follows: In the next section, key requirements are presented. In Section III, software ecosystem perspectives on the system are discussed further, and in Section IV the architecture principles are explained. Section V describes the proof-of-concept prototype, and Section VI introduces some related work. The final section summarizes the conclusions and gives directions for future extensions.

II. Key Requirements

The initial step in the research was to elicit the key architecturally significant requirements. This was done by creating scenarios from decision processes, and use them to identify stakeholders, their concerns, and important use cases. These requirements address what different users want to get out of using the decision support.

A. *Stakeholders*

Four key stakeholder groups were identified:

This work is supported by The Knowledge Foundation in Sweden (Grant No. 20140218), as part of the ORION project, www.orion-research.se.

1. *Decision makers.* COACH users who participate in a decision process using the tool.

2. *Contributors.* Developers of extensions to the tool, in the form of modules implementing special decision processes, estimation models, etc.

3. *System administrators.* Responsible for setting up a system instance, and maintaining its operation.

4. *Process analysts.* Researchers who analyze aggregated data from many decision cases for research purposes or for further improving the tool.

B. Concerns

The stakeholders have different concerns, and the most important ones from an architecture perspective are, in the terminology of ISO 25010 [17]:

- *Usability in use*. It must be easy to use the system for decision making, for adding extensions, and for configuring it for different uses.

- *Flexibility in use*. The system must be dynamically extensible with new contributions, as well as being interoperable (e.g. for including legacy components such as existing data sources).

- *Transferability*. The system should be adaptable to the needs of different organizations with different decision making practices.

- *Maintainability*. The system will have a long life time, and it must be possible to continue its development, while still having access to old data and contributions.

- *Security*. This includes confidentiality and integrity. The decision support will be a critical application for companies, and the security level should be similar to that of other critical applications.

Some of these concerns are conflicting, constituting trade-off points. For instance, high flexibility may make it more difficult to maintain and secure the system.

The elicited requirements also contain descriptions of about 20 use cases, including creation and closure of decision cases; inviting users to a decision case; adding case facts and decision alternatives; deciding on the decision process to use; deciding on what property to evaluate alternatives against, and how to estimate those properties; generating and reviewing decision recommendations; and extending the tool with different models.

III. ECOSYSTEM CHARACTERISTICS

In a typical organization, the most difficult decisions would be the ones that are rarely made, whereas the more standard ones are well known and are often already well supported through organizational processes. It is thus in the rare decisions that decision support would be most valuable. However, this means that a single organization would typically not build up sufficient evidence about these rare decisions to be able to generate good advice to the decision makers. Instead, it is necessary to provide ways of sharing the relevant knowledge and evidence across organizations, and this is a central idea of our research.

We have found it natural to consider a software ecosystem approach to address knowledge sharing in the development of COACH. A common definition of a *software ecosystem* is "the interaction of a set of actors on top of a common technological platform that results in a number of software solutions or services" [21]. The actors in this case would include decision makers in different organizations, and the platform is the decision support system, which can be extended in different ways with both data and services.

In this section, we will highlight some characteristics of this software ecosystem, and these characteristics are important as input for deriving a suitable decision support system architecture. In the first subsection, we describe what data is shared inside the ecosystem and how that can be used for reasoning. Then, services in the form of best practice methods are introduced, and these are also shared knowledge assets that can be used to improve decision making. Finally, it is discussed how to make the ecosystem attractive to decision makers and contributors.

A. Shared data and reasoning

In order to support decision making, information from a wide range of different data sources is needed. A prime example of such a data source is previous decision cases within the own organization or outside it. Also, public sources such as scientific literature, web sites, etc. are useful. Additionally, there is often a lot of relevant information in different internal development tools in the organization. All these sources can be seen as *knowledge repositories* that can be used for reasoning in order to provide recommendations to decision makers. This broad range of knowledge repositories makes it necessary to have a uniform query mechanism to search for information across the whole range.

When it comes to the information from previous cases, the idea is to allow analogy-based reasoning to suggest recommendations to users, based on information from cases that are in some way similar. To achieve this, a way of analyzing similarity is needed and this requires a classification of decision case data. Therefore, we have created a *taxonomy* for decision cases in the area of software components. This taxonomy is called GRADE [23], and it characterizes decision cases through five perspectives, namely Goals, Roles, Assets, Decisions, and Environments. In short, Goals document the key objectives of a decision; Roles the stakeholders involved in the process; Assets the options available to the decision; Decision the methods used to make the decision; and Environment details of the context in which the decision is taken. The categories are used to tag items in the knowledge repository. GRADE has further been refined into an information model suitable for storing cases in a knowledge repository [12].

In addition, the concept of a *context model* is being explored, which corresponds to the environment perspective of GRADE. It contains information about aspects like organization, business domain, etc., which can to some extent be reused between decision cases within an organization [8].

The information in the context model constitutes the basis for determining whether the preconditions of a previous decision case, potentially from a different organization, is similar enough to justify using it for recommendations in a new case.

B. Best practice methods

Apart from data, there is also a potential for sharing best practice methods in decision making. This can be seen as consolidated procedural knowledge that is best operationalized through software modules. In order to make it possible to extend this type of procedural knowledge over time, the tool has to be extensible with new modules. COACH therefore needs variability points in the decision support system architecture that allows the inclusion of software extensions. These extensions are used to encode a set of different methods [27]:

- *Decision processes*. Each case follows some process to reach a decision, but there will not be a single process that fits all needs. Therefore, different modules can be implemented that define the necessary steps to reach a decision, and describe how trade-offs are to be made between different properties when evaluating alternatives [14].
- *Context models*. As already described above, these models encode contextual factors, and they can vary between companies and business segments.
- *Property models*. A property is a characteristic against which alternatives are evaluated, and for each property different estimation methods are possible, ranging from simple expert opinions to more advanced analytical ones.

C. Ecosystem attractiveness

A key question is how to make the ecosystem actually work, or in other words, what is the value provided that will encourage people and companies to contribute to the ecosystem? As discussed in e.g. [3], a central element is the ownership of information, and this applies very much in this system. Many decision cases will involve proprietary information that should not be shared. At the same time, something has to be shared in order to both build the data in the knowledge repositories, and to be able to generate recommendations from it.

Some characteristics that will contribute to making the ecosystem attractive to contributors are:

1. *Low barrier*. The decision makers should be able to focus on their tasks, and spend minimum time in adjusting their information to fit a specific format enforced by the knowledge repository, or to add meta-information to the case. Preferably, as much as possible should be automated.
2. *Control of confidentiality*. To build confidence, the user must be able to see, and control, what data is shared with others through the knowledge repository. Ideally, the user should be given a direct feedback of the added value received from providing more information, such as an estimate of the confidence level of recommendations.
3. *Useful feedback*. There must be a clear value to decision makers, in the form of useful recommendations, or access to a broader range of appropriate analysis methods.

IV. ARCHITECTURE PRINCIPLES

Given the requirements, and the desired ecosystem characteristics, the architecture principles were elaborated through a set of *viewpoints* that included the overall software structure; the deployment structure; the control flows for the major states in the system; the data flows between the different modules; and the data storage structure.

The architecture analysis led to a number of *cornerstone decisions* that provide the basis of the more detailed architecture. Many elements are similar to previous architectures for decision support systems (see Section VI), but what is unique here is the provisions for building a software ecosystem for decision support.

The cornerstones of the COACH architecture, and their rationale, are:

- *Service oriented architecture* (SOA) with each module being implemented as a stand-alone micro-service (i.e. a self-contained program). **Rationale**: usability for decision makers and contributors (easy to configure system with new modules); flexibility (extensibility, interoperability – different modules can be deployed on different servers, allowing a mix of contributions from internal and external sources in the ecosystem); and maintainability (different pieces can be exchanged independently of each other).
- *Representational state transfer* (REST) based interface between components. **Rationale**: maintainability (minimizes dependencies between modules).
- *Web service protocols* (e.g. HTTPS) for interactions between components. **Rationale**: flexibility (allows components to be developed in any programming language, and supports wrapping of legacy components); maintainability (standardized, widely used technologies).
- *Web browser based user interfaces*. **Rationale**: usability (familiar user interface, cross platform implementations, low threshold for new users); transferability (look and feel can be customized for different organizations by changing style sheets); maintainability (no software installed on user computers, each piece of the user interface is a part of the service to which it is connected).
- *Semantic web ontologies* for describing information structure. **Rationale**: flexibility (since ontologies are extensible, it becomes possible for software extensions to describe how they add data which is specific to that module); maintainability (since the

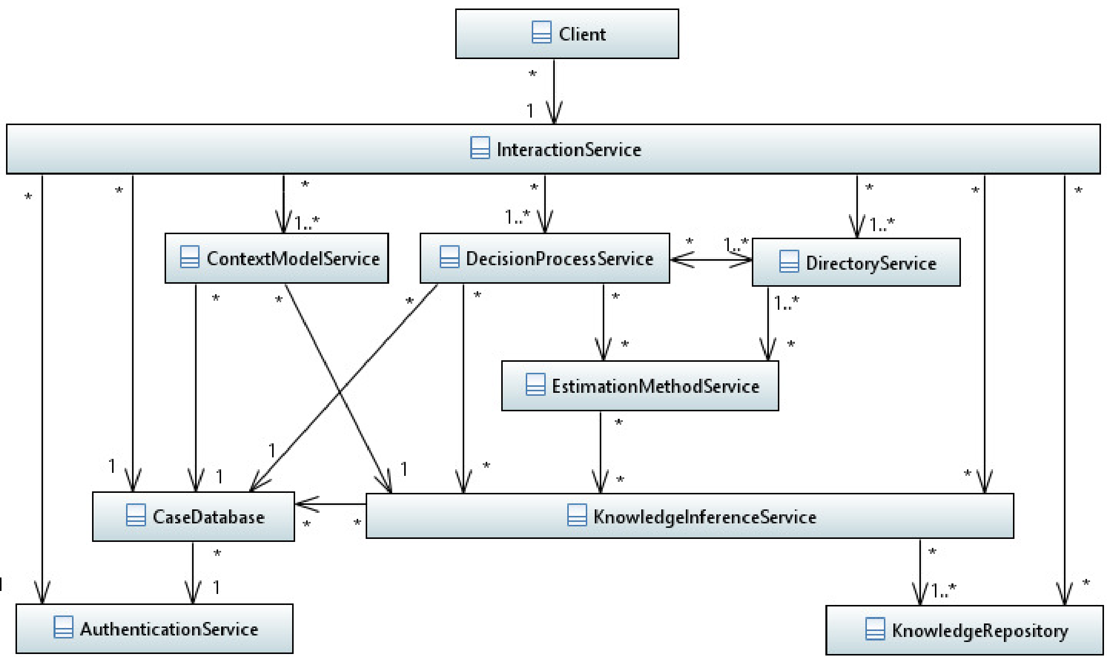


Fig. 1. Software structure viewpoint of the COACH architecture (UML class diagram notation).

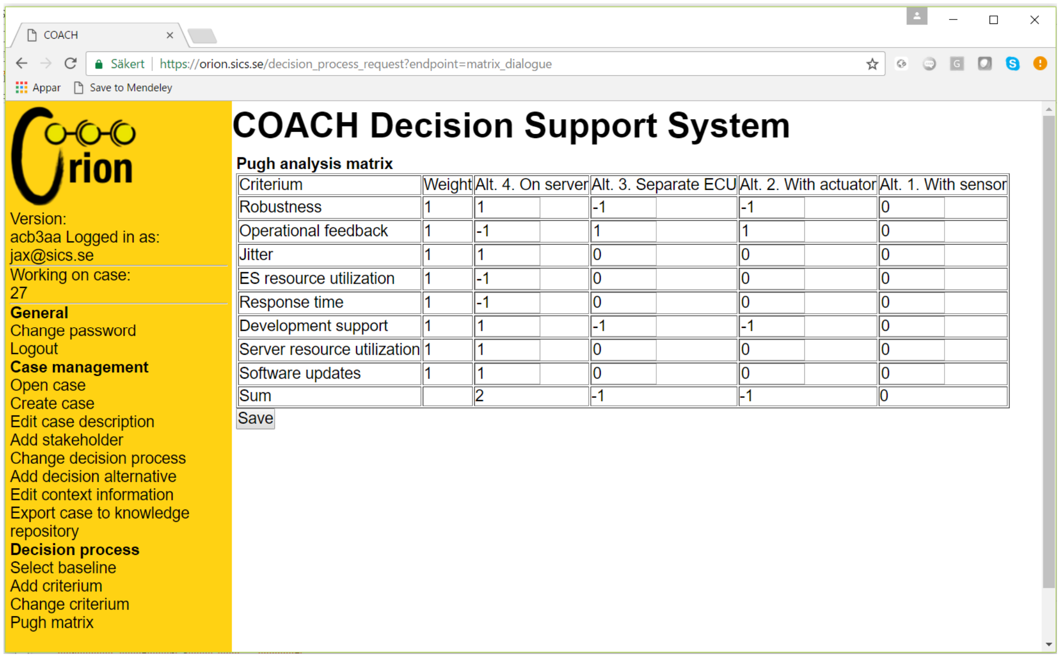
data structure described in the ontology is in itself stored as data, and not hard coded into the algorithms).

- *Graph database* for storing case database and knowledge repository. **Rationale**: flexibility (the extensible data structure makes the graph format more appropriate than tables).
- *Common format for all transfers of structured data.* **Rationale**: maintainability (it reduces the need for creating interface adapters).

For security concerns, a number of key decisions are also made. Mostly, they are in line with standard practices for web based IT systems of the kind described above, including for example user authentication; encryption of sensitive data; and time limited tokens controlling the rights for services to access the data, where the core components can access a broader range of services and a larger part of the data than can the extensions.

Fig. 1 shows the software structure viewpoint of the architecture (using UML notation). As a basis, there are a number of core classes that contain the basic functionality for creating microservices, decision processes, estimation methods, etc. Some of the classes in the framework will now be explained further:

- *Microservice*: The base class of all microservices. It contains the functionality for setting up a service that can act as a stand-alone web server. Subclasses are expected to create the URL endpoints, i.e. the concrete web services to which the microservice should respond. If the service provides a user interface, it is also related to a set of HTML templates and possibly CSS style sheets describing that user interface. (All the classes in Figure 1 except Client are subclasses of Microservice, but this is not shown to avoid cluttering the picture.)
- *Client:* This is a standard web browser through which the user accesses the system.
- *InteractionService*: Implements the overall workflow manager and decision case manager. It is configured with links to a number of directories, which are used for searching for other services. It contains the basic functionality for logging in; registering users; creating and closing decision cases; attaching users to a decision case; selecting the decision process; and initiating a transfer of partial case data to a shared KnowledgeRepository.
- *ContextModelService*: Implements the interactions that allow the user to describe the context in which the decision case is taken place. This information is useful when looking for analogies with other decision cases, in order to provide the user with recommendations. The context data is stored in the CaseDatabase.
- *DecisionProcessService*: Base class for decision process services, which provide the logic of a decision process. It can provide a process menu to the InteractionService, and when the user selects different process steps, further endpoints of the decision process can be invoked. Decision process specific data is stored in the CaseDatabase.


COACH
Säkert
https://orion.sics.se/decision_process_request?endpoint=matrix_dialogue
Appar
Save to Mendeley
Orion
Save

Version:
acb3aa Logged in as:
jax@sics.se
Working on case:
27
General
Change password
Logout
Case management
Open case
Create case
Edit case description
Add stakeholder
Change decision process
Add decision alternative
Edit context information
Export case to knowledge repository
Decision process
Select baseline
Add criterium
Change criterium
Pugh matrix

COACH Decision Support System

Pugh analysis matrix

Criterium	Weight	Alt. 4. On server	Alt. 3. Separate ECU	Alt. 2. With actuator	Alt. 1. With sensor
Robustness	1	1	-1	-1	0
Operational feedback	1	-1	1	1	0
Jitter	1	1	0	0	0
ES resource utilization	1	-1	0	0	0
Response time	1	-1	0	0	0
Development support	1	1	-1	-1	0
Server resource utilization	1	1	0	0	0
Software updates	1	1	0	0	0
Sum		2	-1	-1	0

Fig. 2. Proof of concept prototype.

- *EstimationMethodService*: The base class for estimation methods. Normally, they provide a dialogue for users to enter parameters, and an evaluation method that produces the result of the estimation based on the data.

- *DirectoryService*: Used for providing catalogues of other services. It can be used by the InteractionService to look up e.g. DecisionProcessServices and EstimationMethodServices, but also for a Decision-ProcessService to find estimation methods. To make a new service available to users, it is sufficient to add its URL to a directory that the user can access. The role of this service in the ecosystem is elaborated further below.

- *KnowledgeInferenceService*: Provides a uniform query mechanism to the different knowledge repositories, in order to find evidence and analogies that can be used to generate recommendations to the user.

- *KnowledgeRepository*: Some instances of this class stores selected data from closed decision cases, and uses it to derive generic knowledge and provide recommendations. Others provide access to further information sources, such as existing legacy tools or public sources.

- *CaseDatabase*: Provides the interface to the database for storing case information. This information includes all the data that users have entered into the system as part of using it to reach a decision, and is based on the aforementioned GRADE taxonomy. It wraps an API around a graph DBMS, and this API is used by the InteractionService, ContextModelService, DecisionProcessService, and KnowledgeInference-Service.

- *AuthenticationService*: Responsible for managing user accounts, checking passwords, and generating tokens to be used when InteractionService delegates authority to other services. The CaseDatabase also uses the service to validate that provided credentials are sufficient for accessing different endpoints of its API.

The architecture is strictly layered, so service invocations go downwards in the figure, and results are sent back upwards. It roughly follows the Model-View-Controller pattern, where the View is the web interface shown in the browser client; the Controller is the InteractionService, DecisionProcessService, EstimationModelService, and ContextModelService; and the model is basically the rest.

The directory services play an important role in the ecosystem, since they will be the integration point for contributed services and be the “app stores” of the system. As discussed in [4], this infrastructure of which the directory is a part can be used in many ways for quality assurance, such as:

- *Pre-release testing*: When a service is registered with the directory, a standard test suite can be ran to check that common requirements are fulfilled, before making it available to users.

- *Dynamic configuration management*: Since the directories keep track of what services exist, they can also automatically check that they are interoperable with each other, by testing each newly registered service together with existing services.

- *Online diagnosis and testing*: The directories can continuously check the quality of service of each registered service, and automatically prompt the service owners for actions when an issue is detected.
- *Sharing of operational knowledge*: The performance of different extensions can be measured continuously, both by the system and by user ratings, and this knowledge can be taken into account when making recommendations.

V. PROOF-OF-CONCEPT PROTOTYPE

In order to validate the architecture design, a prototype proof-of-concept implementation has been created, as illustrated in Fig. 2, which shows a Pugh analysis of the case reported in [5]. It is available as open source[1] which is continuously updated, and also as a publicly available demonstrator[2]. The prototype has been implemented mainly in the Python programming language, and uses the Neo4j graph database for storing case data. In some respects, the current implementation is a bit simplified (e.g., the data representation in the case database needs to be elaborated for realistic decision scenarios, and the interaction with the knowledge repository is only rudimentary). The implementation is continuously refined in different ways, both by improving the framework with better user interfaces, improved security, and new features, but also by developing new extensions.

The major conclusion from the proof-of-concept is a validation that the microservice-based concept works in practice, and constitutes a good basis for a software ecosystem. Even though the architecture itself is extremely distributed, it manages to deliver a coherent interaction experience to the users.

VI. RELATED WORK

The research presented in this paper builds on previous work in several areas. Within decision support systems (DSS), a number of architectures have been presented which are web based and have other similarities to our approach. The progress in the area of web DSS was reviewed in [6], concluding that the literature focuses much on implementation and applications, and not so much on architectural issues and design guidelines based on empirical evidence. In [29], an integrated solution is presented to deal with the distributed nature of web DSS, proposing a layered architecture that can integrate data from different disciplines using a component based approach. They use four layers: presentation, knowledge, information, and data. Another framework is presented in [22], which is like ours based on system-of-systems thinking. They emphasize the ability to deploy through cloud computing, and acknowledge the need to support a mixture of public and private clouds. Although the approach to use microservices appears to be unique in our work, [13] is suggesting something similar, but call the services agents. A multi-agent framework is also suggested by [16]. In [28], the emphasis is on how to support flexibility using a service-oriented approach to DSS. A key aspect of the COACH framework is to have access to data, and one approach for this is data mining, as discussed in [24]. Their approach is to use a web DSS in combination with service-oriented solutions, and they apply it in an e-business and e-customs context. The purpose of COACH is to support group decision making, and this was one reason for choosing a highly distributed system with thin clients. Similar work exists in group DSS, such as [1] which focuses on how to reach consensus using a moderator in a Delphi like process. This would constitute a decision process in our framework.

Compared to the above work on DSS, our approach is similar in many respects, but differ on two major points. The first is the use of ontologies for decision making as an extensible data representation, and the other is that we take an ecosystem approach to decision making where data is shared and reused.

Regarding ontologies for decision making, several suggestions have been made, in the context of engineering decisions [25], and for supporting complex group decision processes that involve many subtasks [10]. A generic decision support ontology is described in [26], containing over 200 classes, and another general ontology is presented in [19]. All these suggestions are fairly similar in their general structure, and would also be a suitable basis for the needs in our domain.

When it comes to ecosystem architectures, [7] identifies a number of challenges that have also been identified in our work, including interface stability; workflow and user integration; security; and extensions with new functionality. Examples of ecosystem architectures for distributed systems-of-systems are given in the domains of telemedicine [11] and smart cities [20]. The latter also investigates decision support in order to reuse architectural decisions between different applications, which is similar to the approach of COACH.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we have outlined the architecture of the COACH decision support system for selecting components. The approach is a modular architecture that forms the basis of a software ecosystem, where users can share both abstracted knowledge from previous decision cases, and also best practices encapsulated in services in the microservice based architecture. In this way, the decision makers can profit both from a wider range of analysis possibilities, and from recommendations based on historic data from a wide set of decision makers.

The current architecture forms a promising starting point for the tool and its ecosystem. In the near future, the work will continue in several directions. First, the COACH framework will continue to evolve towards more features and improved quality. This will be carried out by the team members, but hopefully also by others who contribute to the open source or by providing services. Secondly, certain aspects of the framework require more fundamental research, and this is in particular true for the interplay between the knowledge repository and the rest of the framework. Finally, the concept will be used for experiments and case studies, where users will apply the tool to real decision cases, thereby contributing both to validation of the system, but also with data that can be used

[1]https://github.com/orion-research/coach.

[2]https://orion.sics.se.

by analysts to improve knowledge of how decisions are actually made.

References

[1] S. Alonso, E. Herrera-Viedma, F. Chiclana, and F. Herrera, “A web based consensus support system for group decision making problems and incomplete preferences,” *Inf. Sci.*, vol. 180, no. 23, pp. 4477–4495, 2010.

[2] J. Axelsson, “Towards an Improved Understanding of Humans as the Components that Implement Systems Engineering,” in *INCOSE International Symposium*, 2002, vol. 12, no. 1, pp. 1137–1142.

[3] J. Axelsson, E. Papatheocharous, and J. Andersson, “Characteristics of software ecosystems for Federated Embedded Systems: A case study,” *Inf. Softw. Technol.*, Apr. 2014.

[4] J. Axelsson and M. Skoglund, “Quality assurance in software ecosystems: A systematic literature mapping and research agenda,” *J. Syst. Softw.*, vol. 114, pp. 69–81, Dec. 2015.

[5] J. Axelsson, “Architectural Allocation Alternatives and Associated Concerns in Cyber-Physical Systems,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops - ECSAW '15*, 2015, pp. 1–6.

[6] H. K. Bhargava, D. J. Power, and D. Sun, “Progress in Web-based decision support technologies,” *Decis. Support Syst.*, vol. 43, no. 4, pp. 1083–1095, 2007.

[7] J. Bosch, “Architecture challenges for software ecosystems,” in *Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA '10*, 2010, p. 93.

[8] J. Carlson, E. Papatheocharous, and K. Petersen, “A Context Model for Architectural Decision Support,” in *2016 1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*, 2016, pp. 9–15.

[9] M. Cataldo and J. D. Herbsleb, “Architecting in software ecosystems,” in *Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA '10*, 2010, p. 65.

[10] J. Chai and J. N. K. Liu, “An Ontology-driven Framework for Supporting Complex Decision Process,” in *World Automation Congress (WAC)*, 2010.

[11] H. B. Christensen, K. M. Hansen, M. Kyng, and K. Manikas, “Analysis and design of software ecosystem architectures – Towards the 4S telemedicine ecosystem,” *Inf. Softw. Technol.*, vol. 56, no. 11, pp. 1476–1492, 2014.

[12] A. Cicchetti, M. Borg, S. Sentilles, K. Wnuk, J. Carlson, and E. Papatheocharous, “Towards Software Assets Origin Selection Supported by a Knowledge Repository,” in *2016 1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*, 2016, pp. 22–29.

[13] C.-S. J. Dong and A. Srinivasan, “Agent-enabled service-oriented decision support systems,” *Decis. Support Syst.*, vol. 55, no. 1, pp. 364–373, 2013.

[14] U. Franke, “Towards Preference Elicitation for Trade-Offs between Non-Functional Properties,” in *2016 IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC)*, 2016, pp. 1–10.

[15] Hevner, Alan R., S. T. March, J. Park, and S. Ram, “Design science in information systems research,” *MIS Q.*, vol. 28, no. 1, pp. 75–105, 2004.

[16] B. Imène and T. Noria, “A Multi-agent Framework for a Web-based Decision Support System Applied to Manufacturing System,” in *Conférence Internationale sur l'Informatique et ses Applications*, 2009.

[17] ISO, “ISO/IEC 25010 Systems and software engineering - Systems and software Quality Requirements and Evaluation - System and software quality models,” 2011.

[18] ISO, “ISO/IEC/IEEE 42010 Systems and software engineering — Architecture description,” 2011.

[19] E. Kornyshova and R. Deneckère, “Decision-Making Ontology for Information System Engineering,” in *Conceptual Modeling – ER 2010*, Springer Berlin Heidelberg, 2010, pp. 104–117.

[20] I. Lytra, G. Engelbrecht, D. Schall, and U. Zdun, “Reusable Architectural Decision Models for Quality-Driven Decision Support: A Case Study from a Smart Cities Software Ecosystem,” in *2015 IEEE/ACM 3rd International Workshop on Software Engineering for Systems-of-Systems*, 2015, pp. 37–43.

[21] K. Manikas and K. M. Hansen, “Software ecosystems – A systematic literature review,” *J. Syst. Softw.*, vol. 86, no. 5, pp. 1294–1306, 2013.

[22] M. Nouh, M. Hadhrawi, A. Sanchez, and A. Alfaris, “Towards Cloud-Based Decision Support Platform for Group Decision Making,” in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 50–55.

[23] E. Papatheocharous, K. Petersen, A. Cicchetti, S. Sentilles, S. M. A. Shah, and T. Gorschek, “Decision support for choosing architectural assets in the development of software-intensive systems,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops - ECSAW '15*, 2015, pp. 1–7.

[24] L. Razmerita and K. Kirchner, “Data Mining for Web-Based Support Systems: A Case Study in e-Custom Systems,” in *Data Mining for Web-Based Support Systems: A Case Study in e-Custom Systems*, 2010, pp. 387–402.

[25] J. Rockwell, I. R. Grosse, S. Krishnamurty, and J. C. Wileden, “A Decision Support Ontology for collaborative decision making in engineering design,” in *2009 International Symposium on Collaborative Technologies and Systems*, 2009, pp. 1–9.

[26] M. Rospocher and L. Serafini, “Ontology-centric decision support,” *Proceedings of the 2012 International Conference on Semantic Technologies Meet Recommender Systems & Big Data - Volume 919*. CEUR-WS.org, pp. 61–72, 2012.

[27] C. Wohlin, K. Wnuk, D. Smite, U. Franke, D. Badampudi, and A. Cicchetti, “Supporting Strategic Decision-Making for Selection of Software Assets,” in Proceedings of the 7th International Conference on Software Business (ICSOB 2016), 2016, pp. 1–15.

[28] D. Yu and S. Zheng, “Towards Adaptive Decision Support Systems: A Service-oriented Approach,” *Adv. Inf. Sci. Serv. Sci.*, vol. 3, no. 7, pp. 26–34, 2011.

[29] S. Zhang and S. Goddard, “A software architecture and framework for Web-based distributed Decision Support Systems,” *Decis. Support Syst.*, vol. 43, no. 4, pp. 1133–1150, 2007.