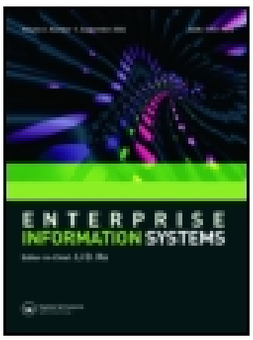

Enterprise Information Systems

ISSN: 1751-7575 (Print) 1751-7583 (Online) Journal homepage: http://www.tandfonline.com/loi/teis20

Aspect-oriented challenges in system integration with microservices, SOA and IoT

Tomas Cerny

To cite this article: Tomas Cerny (2018): Aspect-oriented challenges in system integration with microservices, SOA and IoT, Enterprise Information Systems, DOI: 10.1080/17517575.2018.1462406

To link to this article: https://doi.org/10.1080/17517575.2018.1462406

Published online: 10 Apr 2018.

ENTERPRISE INFORMATION SYSTEMS, 2018
https://doi.org/10.1080/17517575.2018.1462406

Taylor & Francis
Taylor & Francis Group

ARTICLE

Aspect-oriented challenges in system integration with microservices, SOA and IoT

Tomas Cerny

Department of Computer Science, Baylor University, Waco, TX, USA

ABSTRACT

The era of system integration through service-oriented architecture is rapidly becoming a legacy. New systems increasingly rely on the successor microservice architecture. However, distributed applications can also play a significant role in the internet-of-things. In this paper, we gather evidence demonstrating cross-cutting issues in these areas and highlight how aspect-oriented programming addresses them. Specifically, the purpose of this paper is to provide a roadmap for using aspect-oriented programming to deal effectively with crosscuts in application design, systems interaction, and integration.

ARTICLE HISTORY
Received 1 December 2017
Accepted 9 March 2018

KEYWORDS
AOP; SOA; IoT; microservice; cross-cutting concerns

1. Introduction

Aspect-Oriented Programming (AOP) was first introduced in 1997 by Kiczales et al. (1997). In 2001, an AOP extension for Java is released known as AspectJ. Within a short period, MIT predicted that it would be the paradigm for the next decade, according to Prasad et al. (2017). However, the boom never really happen despite undoubtable benefits in its the separation of cross-cutting concerns leading to significant code reductions (Kiczales et al. 1997) and the simplicity of runtime decisions enabling context-awareness and adaptability (Cerny et al. 2013).

While AOP seems to be a good fit for code-centric and monolithic applications, its advantages can also be found also for distributed systems, as demonstrated in Cerny and Donahoo (2015). Individual modules can benefit from reduced restatements internally, or AOP can be used for separation of concerns in module interaction, correlation, integration, reduction of transmission volume, etc.

This paper draws a roadmap of existing AOP challenges and research in the areas of System Integration (SI), Service-Oriented Architecture (SOA) and its successor Microservices (MSA); it also shows AOP's potential for employment in the Internet of Things (IoT). The main focus of this paper is the use of AOP for development, design, and maintenance of the above areas and particular software systems. The evidence is received not only by manual survey but also by a systematic search through multiple research indexing sites and portals, as suggested by Petersen, Vakkalanka, and Kuzniarz (2015). The outcome is analyzed to provide a survey, roadmap and a categorization of existing work.

Section 2 discusses the methodology used to review the literature on AOP utilization. Section 3 outlines the research questions that are addressed by this paper. Section 4 provides the categorization and roadmap to the research, introducing particular categories in details. Section 5 considers particular cross-cutting concerns addressed in the evidence. Section 6 introduces existing aspect weavers. Section 7 provides discussion about validity threats of this work. Conclusions are drawn in Section 8.

CONTACT Tomas Cerny tomas_cerny@baylor.edu Department of Computer Science, Baylor University, One Bear Place #97356, Waco, TX 76798, USA

2. The current literature on AOP utilization

To review the literature on the current state of AOP utilization, we accessed the primary relevant indexing sites and portals (indexers) including IEEE Xplore, SciVerse Scopus, ACM Digital Library (DL), SpringerLink and ScienceDirect. Our target was to search for papers containing AOP and related terms, including AOSD (Aspect-Oriented Software Design). To help refine the results, we added targeted terms such as *Internet of Things, IoT, System Integration, Service Integration, Microservice, Service-Oriented Architecture, SOA* and *Service Oriented Architecture*. The initial searches considering the query in Listing 1 resulted in 483 papers. These searches provided results that fitted our control result set.

```
      ('System integration' OR 'Service integration' OR 'SOA'
        OR 'Service oriented Architecture' OR 'Microservice'
        OR 'Service-oriented Architecture' OR 'IoT'
        OR 'Internet of Things')
AND ('Aspect Oriented' OR 'Aspect-Oriented' OR 'AOP'
        OR 'AOSD' OR 'Separation of concerns')
```

Listing 1: Search query applied to indexers (each indexer had a custom syntax)

The filtering process is captured in Figure 1. Limiting the search to 'full-text only' resulted in 277 papers to examine. In the next stage, we read the titles and abstracts of the papers to decide whether each paper matched the targeting area; moreover, we excluded papers with fewer than 4 pages. This resulted in 112 papers to read. In the last stage, we read the papers to categorize them, further eliminating papers that had no specific output, involved obsolete technology, outdated problems, papers that concluded with a suggestion or an opinion (Wieringa et al. 2006) but with no method proposal (Petersen, Vakkalanka, and Kuzniarz 2015), etc. The final number of considered papers was 48. Table 1 shows the summary divided per indexer.

3. Research questions

We examine the following research questions in this paper:

- *RQ1*: What is the taxonomy of AOP used in SOA, MSA, IoT or SI? What are the topics/subjects that have been addressed in existing research and what is their distribution?
- *RQ2*: What are the specifics in particular research categories? What are the applications of AOP within these categories?

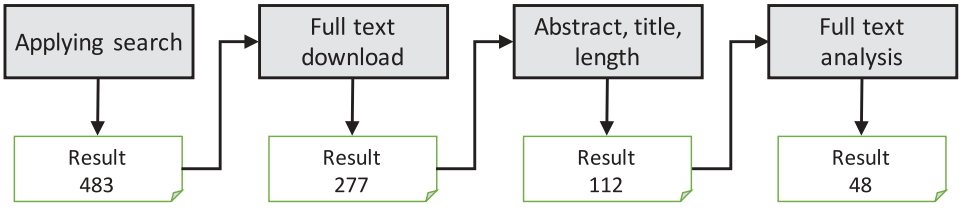

Figure 1. Number of included papers in the selection and filtering process.

Table 1. Processed resources with filtering by indexer.

Indexer	Results	Download	Abstract & Length	Full text
IEEE Xplore	47	47	36	14
SciVerse Scopus	203	8	8	7
ACM DL	42	31	23	11
SpringerLink	182	182	37	12
ScienceDirect	9	9	8	4
Sum	483	277	112	48

• *RQ3*: What are the cross-cutting concerns in the in these areas? What are the existing strategies, aspects, and concerns that are addressed by AOP?

• *RQ4*: What are the existing tools aspect weavers that are used in the evidence?

We answer *RQ1* and *RQ2* in Section 4 below. *RQ3* is answered in Section 5. *RQ4* is addressed in Section 6. Notice that, since we placed all relevant papers into the different categories (service modularization, a composition of services, etc.), a few papers show up multiple categories. To verify our manual categorization and avoid human errors, we also extracted papers' keywords automatically using RAKE algorithm (Rose et al. 2010), which we later compared with the result of full-text reading and manual keyword extraction.

4. Taxonomy of the research

After identifying papers with relevant topics, we then categorized them. To determine categories, we performed full-text reading considering the content. While various works differ with their purpose and approaches, Wieringa et al. (2006) suggest determining research type based on an evaluation framework. Following this suggestion helped us to derive the addressed problem, contribution, and results; however, the evaluation framework did not suggest the research topics categorization. Consequently, we determined the best methods for categorization.

Existing mapping studies, such as Kratzke and Quint (2017), derive research topics by keywords. For instance, Kratzke and Quint (2017) analyzes papers using either provided keywords or derives them from the content. Each paper is accompanied by a set of keywords. These keywords are unified to use the same terms and clustered to research topics. These are eventually aggregated to the main research topics. Finally, the papers are grouped to the main topics. Moreover, one paper can fit multiple topics.

Waltman and Van Eck (2012) mentions another approach where categorization can be determined by the journal or conference area and then using citations to determine clusters of research areas. The final step, labeling, involves extracting labels from titles and abstracts, normalizing them and identifying each area by a set of terms.

Alshuqayran, Ali, and Evans (2016) used an automated keyword extraction tool that provides authors with a set of possible categories. They visualized the topics to see the distribution of the underlying population, which provided them with possible candidates. Next, they grouped keywords to particular categories and performed a text search over the evaluated papers to categorize them.

This work generally followed the procedure suggested by Kratzke and Quint (2017). However, we did not use the author-provided keywords; instead, we extracted them, considering the particular research type of the given work through full-text reading. These were clustered to derive particular categories. Each derived category presented a large area with multiple sub-areas. Table 2 shows extracted categories, and Table 3 provides the references. In our classification, papers are not exclusive only to a single category. The following text introduces each category with details in a separate subsection.

4.1. *Verification through automated keyword extraction*

We compared our categorization with the automated keyword extraction approach suggested by Alshuqayran, Ali, and Evans (2016). Since it is an automated approach, we expected it to be less precise than the manual approach. To perform the automated keyword extraction, we used the RAKE algorithm (Rose et al. 2010) . In order to do such extraction, we transformed the PDF documents into text, using the XPDF tool[1] *pdftotext*, and then made manual adjustments as necessary to the text, e.g. stripping references. Next, we applied the RAKE algorithm for keyword extraction. The setting for the algorithm was as follows: a keyword phrase contains at most three words, with each word at least three letters in length and occurring at least three times in the text. We set the RAKE accuracy index higher than four, which is suggested by Rose et al. (2010).

Table 2. Categorization of existing work.

Category ▷ Subcategory	Paper Volume	Percentage
Service modularization	14	21%
Composition of Services	21	31%
▷ Composition of Services	8	12%
▷ Monitoring and reconfiguration	11	16%
▷ Quality of services	6	9%
Modeling	25	37%
▷ Business process,	15	22%
▷ Visualizations, prototyping, modeling	11	16%
Security	7	10%
Testing	1	1%

Table 3. Categorization with references.

Service modularization
(Anderson and Ahmed 2006; Hmida, Tomaz, and Monfort 2005; Zhang, Meng, and Liu 2008; Zhang, Meng, and Liu 2009; Yumei et al. 2009; Deepiga, Senthil, and Babu 2014; Baligand and Monfort 2004; Wada, Suzuki, and Oba 2008; Walraven et al. 2010; Charfi and Mezini 2004; Corradi et al. 2009; Cemus, Cerny, and Donahoo 2015; Harrison 2011; Wang and Wang 2013)
Composition of Services (three below)
▷ **Composition of Services**
(Seiler et al. 2011; Cibrán et al. 2007; Han, Jung, and Yeom 2011; Marzouk and Jmaiel 2013; Ermagan, Krüger, and Menarini 2008; Cerny and Donahoo 2016; Cerny and Donahoo 2015; Spieldenner et al. 2017)
▷ **Monitoring and reconfiguration**
(Cugola, Ghezzi, and Pinto 2012; Huang, Wu, and Wei 2009; Castor, Leite, and Rubira 2012; Balakrishnan and Sangaiah 2017; Balakrishnan and Sangaiah 2015; Giunta et al. 2015; Antunes and Vieira 2017; Baresi, Guinea, and Pasquale 2007; Dell'Amico et al. 2012; Kim and Hong 2011)
▷ **Quality of services**
(Balakrishnan and Sangaiah 2017; Balakrishnan and Sangaiah 2015; Giunta et al. 2015; Baresi, Guinea, and Pasquale 2007; Hassan, Ali, and Bahsoon 2017; Kim and Hong 2011)
Modeling (two below)
▷ **Business process**
(Wada, Suzuki, and Oba 2008; Charfi and Mezini 2004; Corradi et al. 2009; Cemus, Cerny, and Donahoo 2015; Wang and Wang 2013; Seiler et al. 2011; Marzouk and Jmaiel 2013; Cugola, Ghezzi, and Pinto 2012; Huang, Wu, and Wei 2009; Baresi, Guinea, and Pasquale 2007; Kim and Hong 2011; Souza et al. 2011; Braem et al. 2007; Baouab et al. 2009; Wang, Bandara, and Pahl 2010)
▷ **Visualizations, prototyping, modeling**
(Ermagan, Krüger, and Menarini 2008; Hassan, Ali, and Bahsoon 2017; Souza et al. 2011; Braem et al. 2007; Peinado, Ortiz, and Dodero 2015; Krüger, Mathew, and Meisinger 2006; Mosser et al. 2011; Alhaj and Petriu 2010; Alhaj 2011; Wang et al. 2014a; Wang et al. 2016)
Security
(Cerny and Donahoo 2015; Antunes and Vieira 2017; Baouab et al. 2009; Cerny, Trnka, and Donahoo 2016; Sonchaiwanich et al. 2010; Dell'Amico et al. 2012; Sendor et al. 2014)
Testing
(Antunes and Vieira 2017)

The result of the RAKE extraction provided an interesting set of suggested keywords for the considered papers. We ignored the general terms and normalized the keywords (e.g. plural vs singular, equivalent terms). For instance, 16 papers suggested the *web services* keyword, 12 papers suggested the keyword *service composition*, 7 suggested *security*, etc. While these keywords do not necessarily indicate our categorization, according to Alshuqayran, Ali, and Evans (2016), these keywords can be clustered under categories. Table 4 shows our clustering.

As can be noted in Table 5, our manual categorization and the automated approach classified the papers similarly. The automated approach could not match the categorization accuracy since various works may not provide appropriate keywords and the automated keyword extraction has its limits producing a wide-range of distinct keywords. Finally, the clustering and selection of keywords considerably impact the achieved precision. However, the results indicate the suitability of our categorization for a given set of papers.

Table 4. Rake keyword clustering.

Category	Keywords
Service modularization	structure design, development efforts, maintenance efforts, modularization, modularity, service adaptation
Composition of services	(three below)
▷ Composition of services	service composition, composite web service, orchestration, service integration
▷ Monitoring and reconfiguration	monitoring, fault tolerance, service replacement, reconfiguration, dynamic service integration, runtime service integration
▷ Quality of service	quality of service, response time
Modeling	(two below)
▷ Business processes	business rules, service business rules, business process, business logic, BPEL, process execution
▷ Visualizations, prototyping, modeling	chart, UML, modeling constructs, service-oriented modeling, system modeling, aspect-oriented modeling, formal modeling, transformation, visualization, prototype tool
Security	security
Testing	testing

Table 5. Manual and automated categorization comparison.

Category	Automated categorization (unique papers)	Manual categorization
Service modularization	15	14
Composition of services	22	21
▷ Composition of services	11	8
▷ Monitoring and reconfiguration	10	11
▷ Quality of service	6	6
Modeling	23	25
▷ Business processes	14	15
▷ Visualizations, prototyping, modeling	11	11
Security	7	7
Testing	1	1

4.2. *Service modularization*

Multiple resources deal with adaptability and dynamic reconfiguration of service concerns. Anderson and Ahmed (2006) suggest weaving the functionality aspects together for a particular service based on the runtime context. This can help to integrate new technologies into the services. AOP can help with dynamic system evolution. In particular, Anderson and Ahmed (2006) provide a study applying the AOP to services in telegrids.

The topic of service concerns composition through aspects is elaborated in more detail by Hmida, Tomaz, and Monfort (2005) who suggest two directions: either to encapsulate novel functionality to aspects or to consider individual advices to be web services. Encapsulation of novel functionality to aspects that weave into the services, involves the separation of concerns applied to non-functional requirements. The assembly is managed by a policy engine the aspect weaver. The second option considers individual advices implemented as web services. These are possible extensions of other services. This way the approach separates out the platform dependency and removes language dependency in the weaving process. The weaving involves the interaction of the original service with multiple other 'advice' services. Aspects are defined through XML files that are interpreted by a custom XQuery engine generating XQuery scripts for the actual service interaction. In other words, this is the mechanism to weave in an aspect to the original service. While the approach brings an interesting solution, no performance impact is provided, but since significant communication overhead can be noted.

A somewhat similar approach suggested by Zhang, Meng, and Liu (2008) and later formalized and narrowed in Zhang, Meng, and Liu (2009) is to separate out cross-cutting concerns from services. Concerns, such as logging, authorization and transaction, are stripped out of the service to prevent code-tangling and enable adaptability, and instead, they are realized individually as aspect services. The web service is delegated to a weaver that is responsible for coordinating the original

web service with aspect services. The solution is demonstrated in a case study; however, no overhead information is provided to see the impact of the separation. The same approach is researched further by 5,348,541 further using Petri Nets to model the primary service logic to find out the right states for the triggering of the new service feature, realized through aspects.

Separation of services with base functionality and cross-cutting functionalities is also explored by Deepiga, Senthil, and Babu (2014) . This work further applies separation to various service with evolution versions. The authors consider a case study with an SOA application with five distinct versions of evolution depicting changes in the processes and application of aspects. On a step by step evolution, they demonstrate cross-cuts, and they utilize AOP to support all system versions. However, the considered concerns are well known from textbooks, e.g. logging, exceptions handling, observer, persistence, security, etc. The research applies various software quality metrics on the source code demonstrating AOP benefits.

Similarly, Baligand and Monfort (2004) notes that functional and non-functional code, as well as the logic, can be separated by AOP mechanisms, in particular by aspects. The authors note a missing mechanism, in standard solutions, to dynamically bind policies to web services, which usually leads to the inability of effective dealing with service extensions and evolution. In their proposal, the policy document determines aspects that are weaved into the core services. The authors demonstrate the approach using Servlets in Java involving the Javassist library and bytecode modification.

Separation of functional and non-functional application properties in SOA is also explored by Wada, Suzuki, and Oba (2008). In their work, aspect specifies non-functional properties that cross-cut among multiple services in a business process. In order to apply the approach, they developed a new special language involving BPMN extended with aspects.The extended BPMN model is processed by an aspect weaver transforming the model into UML and subsequently to code. This strongly impacts the practice far from a conventional development process.

A different perspective is given by Walraven et al. (2010). Ther aim is to compose distributed features in a distributed environment based on web services. High-level abstraction of feature definition enables integration into composite services. However, when we compare this approach to the previous research, the core functionality is again separated from the additional features. The given examples depict the additional features as non-functional properties. Each aspect-component designing a feature carries out a specification detailing the component mapping to core services. Final composition then links the core services with the aspect-component based on the specification identifiers.

A detailed description of crosscuts in processes within web services is provided by Charfi and Mezini (2004). The authors notice that the business logic contains both the business policies and constraints that are coded as a monolithic block, which makes the business rules hard to change without affecting the core composition logic. Instead, they suggest breaking down the logic into a core part (the process) and several well-modularized business rules that exist and evolve independently. The authors provide good formalism classifying business rules into a constraint, action enabler, computation or inference. Their approach integrates cross-cutting modularity into Business Process Execution Language (BPEL) and their solution is introduced as AO4BPEL considering all classified rules. The business rule is captured as a system aspect that leads to an action, switch, response, etc. Similar goals to improve readability are also attempted by Corradi et al. (2009), however, on less detailed classification and formal description.

Cemus, Cerny, and Donahoo (2015) suggests encapsulating rules as aspects and binding them to data objects rather than to processes. The data object then indicates by annotations, which rules to trigger. Moreover, Drools are used for the enforcement and definition, which further allows rule transformation to e.g. JavaScript representation.

Harrison (2011) surveys SOA and distributed system on modularization. The authors discuss various approaches and advantages from the system evolution perspective with an emphasis on all available approaches. While the authors consider AOP marginally and using it for comparison, they

note the AOP advantages when it comes to system quality, such as maintenance, integrability or extendability.

Similarly, Wang and Wang (2013) surveys SOA systems and cross-organizational applications with a need for service change management. The work divides change categories into interface (data schema and operations), behavioral (interaction and BPEL processes) and non-functional (QoS properties). Next, it looks into service adaptation noticing, process flexibility, service evolution, change analysis and management in service compositions. In the adaptation, they see opportunities for AOP usage; for instance, the business logic is treated as the main concern, and the adaptation logic is specified as crosscutting concerns. To advance process flexibility, they highlight the advantages brought by AO4BPEL, especially for logging, persistence, and security. This brings the AOP perspective to treat a particular business logic as the main concern, where crosscutting concerns aspect are data validation and security. In addition, the work suggests options to use AOP for multiple process variants that correspond to contextual changes based on original processes. In service protocols, they notice a problem of evolution in modifying the protocol definitions or managing running protocol instances when the corresponding protocol definitions change especially for long conversations; both issues are addressable through AOP.

4.3. *Composition of services*

Extension of web services that must handle large multimedia transfers and often deal with redundant calls is examined by Seiler et al. (2011). AOP extends the services to efficiently handle large transmissions and preserving service simplicity. In the work, the aspects are weaved into the service processing chain. Every time a service invocation goes through, workflow engine aspects are considered. A demonstrated case study shows improvements in data transfer between services.

Another work by Cibrán et al. (2007) considers selection policies that encapsulate changing business requirements in SOA, influencing service integration and composition. They introduce a mediation layer responsible for the integration. This layer then defines various aspects that intercept and influence the service interaction. For instance, billing, caching, service selection/management, redirection, and monitoring concerns are cleanly encapsulated in aspects of the mediation layer easily manageable throughout the system.

Han, Jung, and Yeom (2011) explores AOP application to a distributed environment and cluster computing. However, the authors point out the weak research interest in this highly-potential area. AOP enables separation of different types of communication in Message Passing Interface (MPI) processes, thus making the solution more readable and easier to maintain, for instance when it comes to initiation of MPI processes. Moreover, AOP instruments help in this discipline with consistent checkpointing, decoupling indication of critical sections, rollbacks recovery or connection among groups. In an example of critical sections, the authors set the *begin* and *exit* points of each critical section as join points that are applied to the advice about consistent checkpointing. In the evaluation, they show AOP significantly improving source code readability and the modularity. Moreover, they show minimal performance and scalability impact when compared to conventional approaches.

Previously mentioned areas (Han, Jung, and Yeom 2011; Charfi and Mezini 2004) are further elaborated in Marzouk and Jmaiel (2013), specifically, targeting self-adaptation and mobility in web service orchestration and the distributed environment involving AO4BPEL. In case of failure or to guarantee the Quality of Service (QoS), a service can relocate. They designed a prototype solution built on an engine executing BPEL processes incorporating checkpointing and rollbacks. Multiple aspects are introduced to support checkpointing, saving service results, service instance mobility or execution resumption after relocation, extending BPEL processes driven by controllers. However, more detailed reasoning on cross-cuts is not provided in the work.

Service communication through an enterprise service bus (ESB) is tackled on the analytical level by Ermagan, Krüger, and Menarini (2008). The authors propose extending message sequence charts

(diagrams) of interacting services with aspects to modify the behavior of services. An aspect has a specific syntax in the charts followed by the regular expression to match service roles, an interceptor. Furthermore, a novel 'join' message is sent for aspect composition of services. Join is a parallel composition operator that synchronizes services on common messages and integrates the first indicated sequence chart together to a second one. Another operator, a 'match' message, is similar to join but removes the instantiating message from the resulting (weaved) chart. The authors believe (however, with no detailed proof or a study), that such extension will lead to 'advanced routing mechanisms in ESB that foster the creation of applications where the separation of crosscutting concerns is maintained at all levels from modeling to implementation' (Ermagan, Krüger, and Menarini 2008).

Dealing with ever-changing data representations in distributed systems is a challenge noticed by Cerny and Donahoo (2016). This article suggests that the knowledge of data representation should be encapsulated in the component responsible for persistence. Moreover, this component should provide another component with meta-information about the data structure. This approach accommodates existing enterprise development standards (Cerny and Donahoo 2015) that promote capturing validation or even security concerns bound to data representation through annotations. Streaming other concerns together with data structure meta-information enables integrating components to integrate structure, security, and validation on the fly and thus automatically adapt to changes. However, the integrating components only weakly reference such data components as proxies since a stronger reference would lead to coupling and dependency. An example usage of a weak reference where it is applied in practice is user interface (Cerny and Donahoo 2015). The considerable advantage of the approach (Cerny and Donahoo 2015) is that user interface can be presented on different platforms using native components, while only providing a single backend. The aspect weaver is thus distributed between backend and frontend, leading into automated adjustments in all the heterogeneous user interfaces upon a backend change. For instance, when a novel data field is introduced on a backend component, e.g. with novel validation rules and constraints, no redesign of the user interface is needed as it automatically appears in the update structural stream.

Spieldenner et al. (2017) introduced AOP employment in virtual worlds and games. It is also suitable for many and possibly smart environments, such as IoT, Smart Cities, and Smart Factories. Such environments produce a lot of data in real-time, which require considerable development and maintenance efforts. It is a challenge to keep such systems up-to-date with changing requirements set by the connected sensors and services. The proposed aspect-oriented architecture enables programmers to create highly-maintainable, large-scale virtual environments while avoiding crosscutting concerns, such as code scattering among modules. It consists of a data model, a plugin mechanism, and a synchronization layer that is independent of the other modules. On top of the AOP architecture, which supports re-usability and adaption to new requirements at run-time, a wide range of virtual environment applications can be created. The architecture is built on top of an Entity-Component-Attribute (ECA) data model approach that is based on non-relational databases. In ECA an entity with a set of specified attributes is grouped together with similar components based on their purpose and associated functionality. This encapsulates application-domain specific concepts in a separate data container that specifies certain concepts of objects. The application's logic is encapsulated in plugins. The core server component maintains the ECA data model and schedules plugins. Plugins define component blueprints to the core and implement application logic. These plugins extend both the expressiveness of the data (e.g. adding new components) and the features of the application. The authors also describe their implementation of the weaver, based on Reflection API, component registry and a mapping function, that maps interfaces with components. Plugins then represent new components that match aspects. One specific plugin is responsible for the synchronization and contains the communication layer. The highlighted features are better maintainability, extendability, and adaptability to new requirements (changing data and services).

4.4. *Monitoring and reconfiguration*

Self-adapting orchestration through a concern-separating approach is addressed by Cugola, Ghezzi, and Pinto (2012). Their focus is on replacing imperative approaches with a declarative approach when it comes to orchestration. Their approach is to define a new declarative service orchestration language that separates orchestration aspects, such as goals, initial state, abstract actions (primitive operations) and concrete actions for abstractions. While the approach is not formalized through AOP terminology, it targets the same goals and introduces an interpreter similar to the weaver. The presented approach is directly comparable to AO4BPEL mentioned in Charfi and Mezini (2004) or Marzouk and Jmaiel (2013); however, it provides a less formal way for aspect integration and focuses on the particular area and simplified development.

Service monitoring and recovery support for composite Web services by using an AOP framework on top of BPEL is the aim of Huang, Wu, and Wei (2009). The works build on stateful aspects that track process history. To monitor the process event, the traces of the processes are transformed into abstract syntax trees (AST). The AST is auditable by a visitor pattern. If a violation is found at the process execution, an aspect manager weaves in determined recovery logic advice available in the aspect registry. The recovery then attempts based on the logic to retry, call an alternate or skip.

In Castor, Leite, and Rubira (2012) the authors target fault-tolerance that deals with coordinated exception handling. They define handlers with recovery rules dictating the particular exception propagation across applications. Moreover, in a similar manner, recovery rules are defined indicating the exceptional behavior. In this research, AOP is used to intercept and extend methods that can potentially produce an exception.

For IoT, Balakrishnan and Sangaiah (2017) notice challenges related to the unknown topology and data estimation problems in case of sensor unavailability. The services selection is typically based on QoS attributes, and in order to perform appropriate service selection, there is a need for service discovery. The weaving mechanisms bring service monitoring with decision making and software reconfiguration to meet the QoS attributes. The aspects use an abstract language to define a dynamic reconfiguration logic and functional substitution in case sensors are not available, and to cope with missing data for which they use functional substitution to generate approximated data instead. However, besides a note on Spring AOP weaving, no more details are given. IoT and service discovery is also addressed by the authors' previous work Balakrishnan and Sangaiah (2015), which provides a general overview of service discovery.

In Giunta et al. (2015), which we examine in more detail in the next subsection, the authors look at the subject of capturing incoming requests and handle them. This situation brings an opportunity for runtime monitoring of various system metrics, such as CPU workload and network usage. Moreover, the measured state influences locality of service processing.

Service monitoring is considered by Baresi, Guinea, and Pasquale (2007). Self-healing BPEL compositions (through weaving of supervision rules) are introduced, using a custom DSL for defining monitoring and recovery activities. An AOP-extended version of the ActiveBPEL orchestration engine is proposed on top of the JBoss Rule Engine to ensure self-healing capabilities. Such monitoring helps to identify issues with QoS and address adaptation to new inputs. Self-healing capabilities are materialized by weaving supervision rules into the BPEL process. Each rule comprises a location in the process, monitoring parameters that tailor the degree of supervision and monitoring expressions that define the constraints that must hold at the runtime and the recovery strategy (advice) that defines what the system should do to try to keep things on track. The case study demonstrates the approach's practical use as well as addressing the concern separation. The supervision rules are translated into Drools rules and hooked into the system.

The previously mentioned work by Cibrán et al. (2007) introduced a mediation layer responsible for service integration. The layer also deals with service management, redirection, and monitoring. The particular concerns are encapsulated through aspects of the mediation layer.

A later subsection elaborates Kim and Hong (2011) that detects faults through aspect processes monitoring the service invocation on top of WS-BPEL processes and suggests their replacements.

4.5. *Quality of service*

The QoS perspective is the main focus of Kim and Hong (2011). It introduces a QoS-related layer on top of the web server. Requests are assessed and partitioned into those that can be handled using local resources and those that need further cloud-based resources. AOP enables the new QoS-related code to be separated from web server modules, while runtime behavior is modified with the measures needed to handle QoS concerns. The extension layer is evaluated from the performance perspective showing a positive impact on the overall throughput, in some cases reducing the processing time to approximately 66% under large load. As mentioned earlier, the introduced aspects capture incoming requests and handle them, which brings an option to monitor various system metrics. As the result, the server can be augmented by weaving it with an appropriate subset of aspects.

Modeling of the microservice granularity is introduced by Hassan, Ali, and Bahsoon (2017). This modeling approach brings the option of determining various QoS trade-offs. Each microservice is modeled and an optimum 'ambient' is determined. Each ambient is composed of at least fours aspects: a granularity adaptation (relevant to QoS and scaling), a mobility (interaction in the hierarchy with parent), a coordination (external/internal call and redirect management) and a distribution (hierarchical position for the mobility). Aspects communicate through weaving in their proposal. For instance, they are weaved together for a particular accomplishment, such as establishing a connection with other services while preserving QoS conditions.

In previously mentioned work on IoT, by Balakrishnan and Sangaiah (2017) and Balakrishnan and Sangaiah (2015), services selection is based on QoS attributes. The weaving mechanism involves service monitoring to obtain needed attributes to determine the status and compare it with the QoS requirement, resulting in optimum software reconfiguration.

Similarly, Baresi, Guinea, and Pasquale (2007) address self-healing compositions that involve monitoring harvest system attributes to determine the QoS. The current status of the attributes determines the adaptation.

As will be discussed below, Kim and Hong (2011) also fits this section. QoS is enforced by monitoring considered attributes through service invocation resulting in conditional service replacements.

4.6. *Business process*

In order to improve SOA service composition reliability, Kim and Hong (2011) uses AOP to detect service faults. They again consider BPEL for service composition and extend it with dynamic service replacement. The key contribution is that there is no need to explicitly represent all possible recovery mechanisms in a BPEL process. Next, unexpected service failures are handled by dynamic replacement of a failed service. Aspects weave in dynamically, through a BPEL engine extension, during the process execution and trigger when conditions are met. The BPEL metamodel is extended to integrate aspects, with thorough details in the paper. Aspect processes are introduced and proposed to detect service faults and to replace a service. A case study shows reasonable overhead for the approach.

Business-Process Modeling (BPM) extended with AOP principles can be applied for service identification in early stages of system analyzing (Souza et al. 2011). An analytical approach is proposed, bringing the ability to identify services with cross-cutting interests. This is achieved by extending function allocation diagrams. These capture service activity and first focus on the primary service goals and next consider the extending aspects that become available for various services.

The AOP-based distributed service composition approach introduced by Wang, Bandara, and Pahl (2010) enables multiple process clients hot-plugging their business compliance models as crosscuts (business rules, fault handling policy, and execution monitor) to utilize BPEL business processes. The BPEL instrumentation mechanism is used to transform the original process to an aspectual business process. In the process multiple join points are defined, such as an invocation of service, processing a message or handling faults. Both before and after advice are provided. The instrumentation transforms the original process to an AOP-enabled BPEL process before deploying it in any execution engine. The instrumented process is able to communicate with the weaving component (weaver) on the client side, which enables dynamic aspect integration. The implementation involves annotations and introspection (Java Reflection). XML files define aspects. The solution centralizes the above concerns, enforcing them across the enterprise.

In the remainder of this section, we mention the previously-discussed work relevant to this area. Wada, Suzuki, and Oba (2008) suggests using BPMN for the separation of functional and non-functional properties. BPMN is supplemented with aspects and processed by an aspect weaver. Using BPEL extensions to address cross-cuts is described by Charfi and Mezini (2004) and Corradi et al. (2009). They divide the process by business rules. Charfi and Mezini (2004) introduces an extension to AO4BPEL; it is applied similarly by others. AO4BPEL is also used by Wang and Wang (2013) for process flexibility. For instance, it is used for logging, persistence, and security. AO4BEPL is used also to address service relocation and self-adaptation by byMarzouk and Jmaiel (2013). As an alternative to the previous approach, Cugola, Ghezzi, and Pinto (2012) proposes a proprietary language DSOL (declarative service orchestration language). ActiveBPEL is also used by Seiler et al. (2011) for orchestrating aspect web services with core services. Monitoring and recovery support for composite web services based on BPEL is addressed by by Huang, Wu, and Wei (2009), who proposes a ONCE-BPEL platform. Similarly, Baresi, Guinea, and Pasquale (2007) addresses self-healing using BPEL. Baouab et al. (2009) addresses security through Braem et al. (2007) designed an analytical visualization tool for integrating aspects that generate BPEL as the product (we describe this work later). Finally, Cemus, Cerny, and Donahoo (2015) utilized business rules captured in Drools to weave through the system indicated in domain objects, which would enable rule transformation to distinct formats. We should emphasize that among the most mature tools for BPEL with AOP is AO4BEPL.

4.7. *Visualizations, prototyping, modeling*

On a high-level of abstraction, Peinado, Ortiz, and Dodero (2015) addresses context-awareness, and adaptability of web services. The main issue with supporting context-awareness is the volume of source code. The authors note the insufficiency of model-based approaches and promote AOP-based extensions. The aspect modules encapsulate the context-aware code. Plain functionality code then indicates injection of aspects; however, join points and pointcuts are not specified, which deteriorates the potential. Therefore the approach enables solely static weaving for the transformation. However, there is additional formalized work on context-awareness support at runtime, enabling consideration of a variety of join points for advice extension, by Cerny et al. (2013).

SOA prototyping addressed with AOP is elaborated by Krüger, Mathew, and Meisinger (2006). The authors use AOP to transform interaction diagrams into different architectures. A large number of aspect types is introduced (system role, interaction, local operation or service, as well as component configuration) all captured in specially-defined language. The language description is woven together with message sequence charts and translated to a particular architecture in AspectJ language. In particular, a build file with configuration per architecture is defined, linking together service repository, classes, and aspects. Similarly, in Ermagan, Krüger, and Menarini (2008), message sequence charts are used to analyze routing mechanisms and communication through ESB.

Another analytic approach is suggested by Mosser et al. (2011) in connecting requirements and design models. The authors research pulling scenario-based requirements into business process design models leading to SOA-based implementations. Moreover, they introduce Aspect-oriented use case maps with a specialized syntax to capture pointcuts, etc. Concerns introduced in one model promote to another through an automated mapping. The product is an ADORE process skeleton with no inconsistencies.

An interesting perspective on information modeling in socio-technical systems is presented by Wang et al. (2014a). . Their approach integrates object-orientation, systems-dynamics, and AOP. The socio-technical enterprise system is complex due to information-exchange constraints between the social and technical system. In order to manage such systems, an information model must be developed. For instance, issues arise with the integration of policies and regulations in the social system domain that impacts the development and operation management in the technical system domain. Their approach proposes five modeling constructs involving a domain, an aspect, a context-of, an agent or an agent group, and a stage. It builds on top of the object-oriented modeling while introducing modeling blocks for socio-technical systems. In the Object-Aspect-Relationship-Domain model (OARD), each relationship is an object and has a lifetime that can be associated with an aspect. This can change the dependencies and associations between objects. Next, it enables model structural changes. The behavior of such a system is determined by the behaviors of its subsystems (e.g. objects, aspects, agents in the OARD model) and the structure of the system. System domain modeling is an important instrument to accomplish the integration of heterogeneous systems. For this specific reason, Wang et al. (2016) introduced a domain modeling framework for service systems (Wang et al. 2014b). The core concepts include function, context, behavior, principle, state and structure, and system decomposition. The service system's unified definition is proposed in Wang et al. (2014b), discussing various alternative definitions found in the literature.

Alhaj and Petriu (2010) proposes a model-driven approach for the evaluation of the run-time performance characteristics of SOA, although the work is in an early development stage. From UML diagrams, in particular from message sequence charts, they suggest deriving Layered Queuing Performance (LQP) models. The SOA platform-independent model acts as the primary model and includes activity diagrams, component diagrams, and message sequences charts. This model is extended with the aspect model, specifying message sequences charts weaving to the primary model either in requests or responses upon services invocation. There are consequent model transitions into a core scenario model and to an LQP-extended queuing network model, which is able to represent nested services and perform the run-time performance analysis. The approach is demonstrated in a case study showing the performance impact of different service combinations. Subsequently, Alhaj (2011) proposes UML profile introducing stereotypes for join-points.

Braem et al. (2007) examines a high-level visualization creation environment used for the analysis stage. In particular, they look into web services and service composition. The modularization supports crosscutting concerns. The visualization allows connecting aspects to components. Service can be encapsulated in both aspects and components. The code generation tool produces a BPEL process which includes weaved-in concerns. The work also introduces a concern-specific language capable of concern definition, visualizable through the introduced tool.

Earlier-mentioned work by Hassan, Ali, and Bahsoon (2017) models microservice granularity as an 'ambient' to see the QoS trade-offs. The proposal considers a hierarchical structure of ambients of at least fours aspects: a granularity adaptation, mobility, coordination and a distribution.

Finally, in the previously-mentioned work by Souza et al. (2011), the authors propose analysis with the aim of identification of services. They suggest using an event-driven process chain detailed in the activity models called function allocation diagrams and extending them via aspect-oriented business process modeling to capture crosscutting concerns encapsulated in modules. With their approach, we can analyze reuse of crosscuts and track their usage across services.

4.8. *Security*

Security in SOA and generally in distributed systems is a challenging topic, and the AOP advantages in this area have been noted by various authors.

Baouab et al. (2009) addresses security of heterogeneous services in SOA. They aim to separate security from business services and propose a new model of three components including business services, security meta-services, and an orchestration service. They manage this through the concept of reusable meta-services, a helper to build business services or security services. Six categories of meta-security services are introduced. When a request is made a security gateway intercepts the call and directs it to various meta-services through BPEL to produce the response, enforcing security. Moreover, services are grouped under a particular policy. The approach separates the security concern to easily reuse it; however, no performance impact is studied or provided.

A similar outcome through a different use of the AOP mechanism was achieved by Cerny, Trnka, and Donahoo (2016). Service security rules and constraints are usually elements of the internal knowledge. However, we can separate out the service security concerns involving a domain-specific language (DSL) description, such as Drools or MPS. DSLs usually have well-described meta-models, allowing us to parse the captured rules and enforce them within the service. Moreover, the parsed rules can be transformed into a machine-readable format (JSON, XML) and shared with other services. Such sharing allows us to build a high-level picture of security, run a verification check on security correlation for verification, build a security-aware composite service or even reuse the knowledge in a user interface, applying client-side validation transforming the machine-readable format into JavaScript. In addition, parsing the rules as described above has the great advantage of self-propagation across services, becoming adaptive, enforceable and easy to read or maintain.

Sonchaiwanich et al. (2010) separates out security concerns from business processes. The authors highlight the option to divide development process between security experts and business process experts where only a limited correlation must exist between the groups, each having a specific responsibility. The security framework is packaged separately from the service application, bringing the ability for the security framework to be reused across different applications.

Similarly, Dell'Amico et al. (2012) considers hierarchical policy language capturing security concerns to govern SOA. Policy enforcement involves a distributed hierarchy of reference monitors that control the information flow between services. Security is a cross-cutting requirement that often ends up scattered over several pieces of code and locations. The authors suggest that AOP structuring and modularization improve both readability and enforcement. Introduced language provides rules to express various security policies, and, moreover, it specifies how the domain divides into a hierarchy (corporations, individuals, etc.). Policies regulate the exchange of information between different domains. They are enforced and monitored at the borders of policy domains. Information is expected to carry annotations with security metadata to indicate a join point. The language-specified security rules are handled exclusively by reference monitors running within each policy domain and controlling the flow of information crossing their borders.

Finally, Sendor et al. (2014) promise to use AOP for SOA, to handle the main OAuth protocol flow. However, their approach focuses solely on the use of chaining and interceptors. They achieve separating out the code implementing the OAuth protocol from the application business logic. When Java EE EJB 3.0 Specification[2] was published, it introduced a standard Interceptor API, but it was a long time before anyone brought this basic AOP feature and functionality to practice.

In work described below, Antunes and Vieira (2017) performs a security check on method calls to differentiate an attack from a conventional application usage.

4.9. *Testing*

An approach for designing vulnerability testing tools for web services is introduced byAntunes and Vieira (2017). Particular scenarios include penetration testing, attack signatures and interface

monitoring, and runtime anomaly detection. AOP is applied for service monitoring to intercept web service execution. This allows for the employment of vulnerability detection mechanisms. AOP intercepts all the calls that APIs use to execute SQL, XPath, etc., and delivers the trace to the information collector in a proposed runtime anomaly detection tool. In addition, the collected information can be applied in static vulnerability detection. Security checks are performed for each data access command executed during the attack phase. All commands are intercepted (using AOP), hashed and stored; next, they are compared to the values of valid commands.

5. Particular cross-cutting concerns in the current literature

In this section, we look into the particular cross-cuts described in current literature. In the existing work, we can find that researchers separate out cross-cutting concerns in various situations. It is well known that such as separation works well for decoupling logging, transaction management or exception handling from the core system functionality (Laddad 2009). But the separation can go beyond these. Researchers have applied it to separate our non-functional requirements (Hmida, Tomaz, and Monfort 2005) or to reuse them across services. The core services can act as the main component that is extended with contextual conditions (Wang and Wang 2013) or even extended with novel service versions, bringing the flexibility to maintain evolution management (Deepiga, Senthil, and Babu 2014).

Context awareness, versioning, and adaptability in conventional development usually lead to multiple software versions, requiring new descriptions of the entire application state transition options or even enumeration of every state of combination. This is simply because of the inability, or insufficient mechanisms, of conventional development approaches to deal with cross-cuts (Cerny et al. 2013). The literature usually depicts this through a sketch, such as the one in Figure 2, where the designer thinks of each aspect (e.g. presentation, layout, security, etc.) as individual Figure 2(a). However, when it comes to capturing the concern in the source code, (Figure 2(b)), all aspect collapses and scatters across various locations, introducing restatements, and worse in the distributed environment across distinct modules.

In AOP approaches, the prime 'must have' functionality defines the frame into which a particular aspect weaves in, based on the conditions, usually at runtime. With enough expressive join points and weavers, the designer may only pay attention only to a particular aspect, rather than on how they may 'possibly' weave together. This significantly reduces the concern reasoning from states in N dimensions to a single dimension. This can be seen again on Figure 2(a).

AOP mechanisms have to be adapted for distributed systems. It is no longer sufficient to see the AOP mechanism originally introduced by Kiczales et al. (1997) in 1997 working on a single code base. The shift calls for distributed weaving mechanisms and unified join points. For instance, as suggested by Cerny and Donahoo (2015), the weaver can be divided into multiple parts, one part that provides meta information to the other part responsible for the weaving. This goes hand in hand with SOA and other distributed solutions when we deal with service integration, categorizing services into aspectual and core ones.

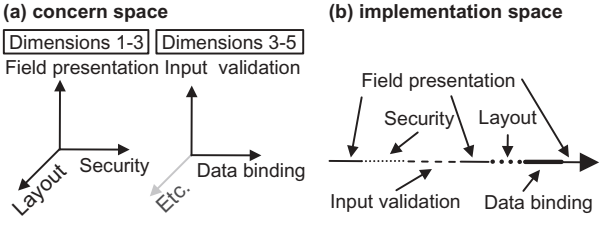

Figure 2. (a) Concern/(b) implementation space.

AOP researchers have demonstrated examples of modules successfully divided into smaller 'aspectual' web services (Zhang, Meng, and Liu 2009; Yumei et al. 2009) or security services (Baouab et al. 2009) and orchestrated by a weaver in composition services. This brings the capability of applying a single change from one location to all services. This is applicable when we deal with policy enforcement (Baligand and Monfort 2004), global governance or security (Dell'Amico et al. 2012). These concerns are usually scattered throughout the system in a conventionally designed system, including restatements and replications that deteriorate maintenance and readability.

Another example of cross-cuts can be seen in business processes where Charfi and Mezini (2004) or Wang and Wang (2013) strip out constraints and business rules from the process, which enables their reuse. This usually brings another benefit. Rules, processes and the system itself are described through different languages, either general purpose (GPL) or domain specific (DSL). Having a limited, single-focus DSL allows utilization of a domain expert (Sonchaiwanich et al. 2010) in the development process, which simplifies development, changes management and evaluation.

As noted by Wang and Wang (2013), AOP help when services must be modified or changes must be enforced across the enterprise. In conventional approaches, change impact tracking can be difficult since services are rarely adaptive. By nature of its design, AOP facilitates adaptation (Cerny and Donahoo 2016). AOP makes services flexible enough to allow modification because it is based on considering/modularizing particular concerns rather than enumerating all options at one place.

The advantages of AOP go beyond modularization and apply to the communication aspect as well. For instance, large blocks of data containing repetition can be replaced by concurrently-delivered concerns assembled together at the destination (Cerny and Donahoo 2015). Moreover, services can be weaved on a custom chain in order to address performance (Seiler et al. 2011). However, such chaining can be easily analyzed because each aspect has already been considered at the modeling level (Ermagan, Krüger, and Menarini 2008).

Changing business requirements together with communication may impact service composition, which could be addressed through a mediation layer (Cibrán et al. 2007). Such a layer may intercept service invocation and weave in aspects for billing, caching, management, redirection or monitoring, all separately defined. AOP's advantages can also be applied to initiate communication and influence initiation, decouple critical sections and help to define strategies for rollbacks, checkpoints or consequent recovery (Han, Jung, and Yeom 2011) (Marzouk and Jmaiel 2013). The same layer may be built on top of a particular server, enforcing QoS with rules and strategies encapsulated in aspects. In Giunta et al. (2015), the authors show the efficiency of such a layer on rerouting requests across local/remote services.

AOP mechanisms can also help when sensing quality attributes in distributed systems (Balakrishnan and Sangaiah 2017), and based on current metrics' values, determine rules to improve the overall systems (Giunta et al. 2015). Such rules can include enforcing priories in communication and triggering relocation or initiation of certain service modules to heal the system (Baresi, Guinea, and Pasquale 2007), which may further enforce agreements on quality assurance (Kim and Hong 2011). This approach can accommodate exceptions-handling across a distributed environment and customizing system recovery mechanisms (Castor, Leite, and Rubira 2012).

Monitoring services impact system performance, since every call may be monitored. AOP brings novel capabilities not only to adjust the current system metrics, but also to determine attacks (Antunes and Vieira 2017), or to weave in security guarantee aspects (Dell'Amico et al. 2012; Baouab et al. 2009).

AOP elements have been incorporated into BPMN and BPEL, where many researchers add an option to weave in concerns such as non-functional properties (Wada, Suzuki, and Oba 2008), monitoring and replacement strategies (Kim and Hong 2011), flexibility (Wang and Wang 2013), logging etc. AOP applies to all the stages of the life-cycle from analyses (Souza et al. 2011), design or runtime. It may help with business compliance crosscuts in heterogeneous environments (Wang,

Bandara, and Pahl 2010) or even stream business constraints and rules across services in machine-readable format (Cemus, Cerny, and Donahoo 2015).

Identifying aspects and their repetition across services in the analytical phase helps the later design phase to avoid restatements (Souza et al. 2011; Mosser et al. 2011). It also helps to define service scope (Hassan, Ali, and Bahsoon 2017). Moreover, AOP facilitates the transformation for system prototyping (Krüger, Mathew, and Meisinger 2006).

5.1. *The role of AOP in IoT*

Current works applying AOP in IoT are rather limited (Balakrishnan and Sangaiah 2017) (Balakrishnan and Sangaiah 2015; Spieldenner et al. 2017). However, the described problems addressed by AOP fit well to IoT solutions. Consider Guinard et al. (2010) that puts together SOA with IoT, leading to an Internet of Services. In their work, the authors expect that dynamic network discovery, queries, selection and on-demand provisioning of web services will be of crucial importance to IoT. Similarly, Issarny et al. (2016) emphasizes the requirements for ultra-large scale, extreme heterogeneity and the dynamics of the IoT. The authors believe that SOA as a middleware can address the challenges posed by the IoT for the development of distributed applications. While SOA has undoubtedly shaped the industry, newly developed systems should move towards MSA (Cerny, Donahoo, and Trnka 2018). MSA enables large scaling for selected services, continuous deploy, and isolation or resilience to failures, which provides a better fit and necessary agility for future IoT middleware demands. Although device heterogeneity will still be a challenge, considering the distinct protocols of interaction, various works we discussed consider an additional layer that can easily extend the integrability (Balakrishnan and Sangaiah 2015; Cibrán et al. 2007).

Balakrishnan and Sangaiah (2017) discussed challenges related to the unknown topology and data estimation, to address temporary IoT sensor unavailability. Their weaving mechanisms consider service monitoring, which is important to obtain context. The weaving mechanisms also use the context for decision making and software reconfiguration to meet the QoS attributes. With real-time systems, this can be a challenge (Xu et al. 2017). Thus aspects define a dynamic reconfiguration logic as well as a functional substitution in case the sensors are temporarily not available. Missing data can be replaced with approximated data instead. The IoT service discovery mentioned by Guinard et al. (2010), can be addressed by AOP, especially when dealing with context-awareness. The authors emphasize that IoT solutions require the conception and development of dynamic adaptation and autonomic capabilities, including mechanisms for discovering available resources and capabilities.

Balakrishnan and Sangaiah (2015) suggests placing middleware between the IoT hardware and the application to process the data gathered from the device component. Such middleware is responsible for interacting with IoT device and management of the information, as well as solving heterogeneity challenges. The middleware layer may perform functional substitution when necessary, but it primarily brings reconfigurability to support the heterogeneity of IoT sensors. The AOP is employed to design such a layer, for instance, considering features as aspects that are dynamically enabled for given context. It can also be used to create a set of agents for given challenges, as proposed by Spieldenner et al. (2017). The new aspects can be introduced to adapt to an evolution, e.g. new sensors. However, the crucial part of the middleware is to know the context, which requires monitoring of the infrastructure. Existing work, e.g. Baresi, Guinea, and Pasquale (2007), discusses AOP monitoring. These requirements seem very suitable for AOP design. Baresi, Guinea, and Pasquale (2007) provides a large section of existing work on monitoring and reconfiguration. This approach of agent design through AOP is quite common in works we considered in this survey.

Similarly, Spieldenner et al. (2017) considered a dynamic IoT environment perspective. To deal with the ever-changing requirements, the authors proposed an architecture that accepts plugins

(separate modules) with a particular logic. These are meant to provide extensibility and adaptation to new requirements, e.g. a new sensor protocol support. When compared to Balakrishnan and Sangaiah (2015), plugins and agents that address the same challenges and serve the same purpose.

The same approach, to simplify service integration with an additional mediation layer, is introduced by Cibrán et al. (2007) to simplify service integration. In the QoS perspective, Giunta et al. (2015) introduced a QoS-related mediation layer on top of a web server.

The work of Kim and Hong (2011) can be addressed to face fault-tolerance in IoT. Similar to previous approaches, they propose detecting faults through aspect processes monitoring the service invocation on top of WS-BPEL processes and suggesting their replacements when necessary. Castor, Leite, and Rubira (2012) then addresses the fault-tolerance through coordinated exception handling.

6. Aspect weavers

This section introduces the weavers used in the evidence. AspectJ and SpringAOP are among the most well-known ones; however, proprietary weavers are often designed on top of language metamodels. Unspecified weavers have also been developed. The below list shows the weavers with a dedicated website.

AspectJ - AspectJ weaver is used in (Yumei et al. 2009; Deepiga, Senthil, and Babu 2014; Krüger, Mathew, and Meisinger 2006; Sonchaiwanich et al. 2010), AspectJ with Aspicere (Han, Jung, and Yeom 2011), AspectJ with ActiveBPEL (Seiler et al. 2011),
(https://eclipse.org/aspectj)

AO4BPEL - AO4BPEL is used in (Charfi and Mezini 2004; Wang and Wang 2013) and mentioned in (Marzouk and Jmaiel 2013), (https://github.com/alook/ao4bpel2/wiki)

SpringAOP - SpringAOP weaver is used in (Corradi et al. 2009; Balakrishnan and Sangaiah 2017; Balakrishnan and Sangaiah 2015), (http://spring.io)

AspectFaces - AspectFaces weaver is used in (Cerny and Donahoo 2016; Cerny and Donahoo 2015; Cerny, Trnka, and Donahoo 2016; Cerny et al. 2013) AspectFaces with RuleGuvnor is used in (Cemus, Cerny, and Donahoo 2015), (http://www.aspectfaces.com)

AMBIENT-PRISMANET - AMBIENT-PRISMANET middleware AOP for componentbased application (Hassan, Ali, and Bahsoon 2017), (http://prisma.dsic.upv.es)

JAsCo & JAsCo.Net - AspectJ alternative/complement with aim to combine AOP with component-based software development introducing aspect beans and connectors (Cibrán et al. 2007), (http://ssel.vub.ac.be/jasco/what.html)

The proprietary weavers with no website, unspecified and self-developed are found in (Anderson and Ahmed 2006; Hmida, Tomaz, and Monfort 2005; Zhang, Meng, and Liu 2009; Zhang, Meng, and Liu 2008; Baligand and Monfort 2004; Walraven et al. 2010;(Kim and Hong 2011; Baouab et al. 2009). Other research works provide more details on used weaver, however, we did not find a website or repository for them. The below list provides details on the mentions.

proprietary Jess rule engine extension - Extension of Jess rule engine (Wang, Bandara, and Pahl 2010)

proprietary - Aspect-oriented process modeling language (AOPML) with no details on the weaver (Souza et al. 2011)

proprietary Apache CXF extension - Apache CXF framework (Sendor et al. 2014).

Ark.bpmn - BPMN weaver (Wada, Suzuki, and Oba 2008)

Continuum/J - (Harrison 2011)

M2Code extensions - Weaver for Aspect Message Sequence Charts (Ermagan, Krüger, and Menarini 2008)

DEng - DSOL execution engine built on Apache CXF (Cugola, Ghezzi, and Pinto 2012)

ONCE-BPEL - introduced in (Huang, Wu, and Wei 2009)

EH-SCA -An Exception Handling System for Service Component Architecture (Castor, Leite, and Rubira 2012)

Kaqusei - Keen Aspect-oriented and Quality-oriented Server-side Infrastructure (Giunta et al. 2015)

RAD-WS - Runtime anomaly detection tool (Antunes and Vieira 2017)

Dynamo Build on top of JBoss Rule Engine, with ActiveBPEL (Baresi, Guinea, and Pasquale 2007)

HiPoLDS RM - Hierarchical Policy Language for Distributed Systems (HiPoLDS) with no details on Reference Monitors (RM) enforcing security policies that act as weavers (Dell'Amico et al. 2012)

Padus - Padus weaver in Service Creation Environment (SCE) which includes a BPEL engine (Braem et al. 2007)

MoDAS - Model-Driven Adaptable Services that generate Java and AspectJ code (Peinado, Ortiz, and Dodero 2015)

ADORE engine - Aspect-oriented Use Case Maps (AoUCM) mapping to ADORE aspectoriented business process models, using unspecified ADORE engine (Mosser et al. 2011)

PUMA4SOA - Performance from Unified Model Analysis for SOA (PUMA4SOA) (Alhaj 2011) (Alhaj and Petriu 2010)

7. Threats to validity

A primary threat to the validity of survey studies is inadequate coverage. We tried to address these issues by the breadth of our coverage and the design of our research parameters. Our evidence elimination mechanism is similar to guidelines for conducting systematic mapping studies as introduced by Petersen, Vakkalanka, and Kuzniarz (2015). However, we did not consider certain parameters such as countries of authors or institutions, or a particular count of resources per conferences, authors or year.

Although we cannot guarantee 100 percent coverage of related papers, we believe we have selected all relevant papers that are within the scope of this study. We addressed this threat by selecting and examining several search strings that fit our paper control set. Moreover, multiple considered papers reference each other.

Next, a potential threat is data extraction bias based on the human factor. We addressed this threat by conducting three individual reviews for each paper, with respect to classification, cross-cutting concerns, and weaver. We also addressed this threat by using a RAKE algorithm (Rose et al. 2010) to extract paper keywords, that we matched with the manually-extracted ones.

Another potential threat is the exclusion of relevant papers due to their scope. To address this, we followed methods for establishing the selection criteria suggested in (Petersen, Vakkalanka, and Kuzniarz 2015). We then spent considerable time reading the selected papers, as well as their related work, to assure that the papers fit within the considered scope. We excluded papers outside of the AOP or AOSD range and papers not relevant to SOA, Microservices, IoT, or close topics such as web services, or grids. We also excluded papers with no specific output, or papers that concluded with a suggestion or an opinion but no method proposal. Finally, we excluded papers not published in the considered academic databases or papers which had very limited content.

8. System integration summary

While broad surveys on system integration can be found (He and Xu 2014; Liu et al. 2008), they currently do not describe AOP use. However, many integration challenges can be addressed by AOP. For instance, the survey by He and Xu (2014) considers the integration that must take place on the communication layer, for data integration, for business logic integration or at the presentation layer integration across heterogeneous systems. A brief consideration of these four areas shows the potential of AOP.

The *communication layer*, can be delegated to SOA, which acts as the mediator, but the more up-to-date approach is MSA, where particular applications are connected individually (Cerny, Donahoo, and Trnka 2018). AOP addresses QoS for the communication, together with monitoring

and reconfiguration of services (further discussed in Sections 4.4 and 4.5). Moreover, when a particular choreography or orchestration is needed, a lot of work addresses the process flow together with monitoring and reconfiguration, such as (Charfi and Mezini 2004; Wang and Wang 2013; Marzouk and Jmaiel 2013), especially involving BPEL. More examples of the composition are discussed in Section 4.3. Multiple works mention an additional mediation layer where AOP weaving takes place (Balakrishnan and Sangaiah 2015; Cibrán et al. 2007).

There is rather limited work on *data integration* with AOP. Balakrishnan and Sangaiah (2017) mention data approximation when sensors are temporarily unavailable; this simplifies the data integration design. However, when it comes to different data schemes across various services, the MSA with bounded context is the right direction to consider (Cerny, Donahoo, and Trnka 2018). When a global or a centralized perspective of information is needed, MSA does not provide any mechanism to do that an requires revisiting each service. One possible solution, suggested by Cerny and Donahoo (2015), is to apply code-inspection and AOP transformation which produces a data representation of a particular service in machine-readable formats that can be easily centralized or processed by consumer services, and this builds a bigger picture.

When it comes to *business logic integration*, SOA services offload the processes to the ESB, and thus all the process flow is governed from a centralized location (Charfi and Mezini 2004; Wang and Wang 2013; Marzouk and Jmaiel 2013). This topic is discussed in Section 4.6. When we move to MSA, the centralization is lost, as each MSA contains its own process flow. Cemus et al. (2017) showed that when capturing business processes and rules in a domain-specific language, through AOP they can be inspected and through AOP enforced throughout the service, and, moreover, they can be extracted and transformed to machine-readable formats and thus shared across services to build a large picture. One particular case is transforming business rules to JSON and providing them to the application front-end for a client-side enforcement.

Finally, the *presentation layer* is currently addressed mostly be client-side frameworks (Cerny, Donahoo, and Trnka 2018), while SOA preference used to be portals liu2008manufacturing. AOP can help address development and maintenance efforts, adaptation and context-awareness. For instance, Cerny et al. (2013) suggested utilizing AOP to derive data presentations in user interfaces from data layers directly while enabling full customization, integrating validation, overcoming constraints and enforcing security (all considered as distinct aspects). Cerny and Donahoo (2015) also introduced platform-independent data presentations that can be streamed to any platform to build a presentation on a native platform using its specific components and widgets. Here AOP is utilized to determine the context and situation for which the presentation should be provided. One beneficial consequence is that any data change in the service provider is directly propagated to all consumer services avoiding correlation errors or incompatibility. When multiple distinct or heterogeneous services provide the data presentations in the platform-independent format, the presentation integration becomes similar to presentations within homogeneous application backends.

Liu et al. (2008) mentions process integration, which is usually base on a business-oriented layer on top of applications. The processes and their integration in the scope of AOP are detailed in Section 4.6. The same authors also discuss integration on the middleware level, where message brokers, transactions, and queuing are considered ((e.g. Messaging Systems). Specifically, for messaging, Han, Jung, and Yeom (2011) considers different types of communication in MPI processes in a distributed environment and cluster computing improving readability and maintainability with AOP. They look into checkpointing, critical sections or rollbacks. More work on messaging and particular message types are further discussed by Ermagan, Krüger, and Menarini (2008) with AOP messages that influence service behavior. With the modeling and analytic perspective Alhaj and Petriu (2010) and Krüger, Mathew, and Meisinger (2006) involve message charts to evaluate the run-time performance or analyze routing mechanisms and communication

through ESB. Finally, the security perspective for system integration involving AOP is detailed in Section 4.8.

9. Conclusion

This paper provides a roadmap of the AOP accomplishments and challenges in the areas of System Integration, SOA, Microservices, and IoT. It categorizes the gathered evidence into service modularization, a composition of services, monitoring and reconfiguration, quality of service, visualization, prototyping and modeling, business processes, security, and testing and provides detailed discussion.

Next, it describes where specifically cross-cutting concerns can be found in such distributed system design and interaction. Various AOP tools evidenced in the literature were described to help the reader jump to integration rather than reinventing the wheel. This roadmap will help the community understand and apply the existing knowledge to deal more efficiently with difficulties in application design and interaction in related areas.

To our best knowledge, there is no similar survey for AOP and System Integration or distributed systems.

Notes

1. http://www.foolabs.com/xpdf/.
2. https://www.jcp.org/en/jsr/detail?id=220.

Disclosure statement

No potential conflict of interest was reported by the author.

ORCID

Tomas Cerny http://orcid.org/0000-0002-5882-5502

References

Alhaj, M. 2011. "Automatic Generation of Performance Models for SOA Systems." In *Proceedings of the 16th International Workshop on Component-Oriented Programming, WCOP '11*, 33–40. New York, NY: ACM. doi:10.1145/2000292.2000299.

Alhaj, M., and D. C. Petriu. 2010. "Approach for Generating Performance Models from UML Models of SOA Systems." In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '10*, 268–282. Riverton, NJ: IBM Corp. doi:10.1145/1923947.1923975.

Alshuqayran, N., N. Ali, and R. Evans. 2016. "A Systematic Mapping Study in Microservice Architecture." In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, 44–51. Macau: IEEE.

Anderson, M., and L. Ahmed. 2006. "An Aspect-Oriented Approach to Developing a Teleworking Grid." In *2006 IEEE International Conference on e-Business Engineering (ICEBE'06)*, 461–465. Shanghai: IEEE.

Antunes, N., and M. Vieira. 2017. "Designing Vulnerability Testing Tools for Web Services: Approach, Components, and Tools." *International Journal Information Security* 16 (4): 435–457. doi:10.1007/s10207-016-0334-0.

Balakrishnan, S. M., and A. K. Sangaiah. 2015. "Aspect Oriented Middleware for Internet of Things: A State-Of-The Art Survey of Service Discovery Approaches." *International Journal of Intelligent Engineering and Systems* 8: 16–28. doi:10.22266/ijies.

Balakrishnan, S. M., and A. K. Sangaiah. 2017. "MIFIMMiddleware Solution for Service Centric Anomaly in Future Internet Models." *Future Generation Computer Systems* 74 (C): 349–365. doi:10.1016/j.future.2016.08.006.

Baligand, F., and V. Monfort. 2004. "A Concrete Solution for Web Services Adaptability Using Policies and Aspects." In *Proceedings of the 2nd International Conference on Service Oriented Computing, ICSOC '04*, 134–142. New York, NY: ACM. doi:10.1145/1035167.1035187.

Baouab, A., O. Perrin, N. Biri, and C. Godart. 2009. "Security Meta-Services Orchestration Architecture." In *IEEE Asia-Pacific Services Computing Conference APSCC 2009*. Biopolis, Singapore: IEEE. December. https://hal.inria.fr/inria-00431678.

Baresi, L., S. Guinea, and L. Pasquale. 2007. "Self-Healing BPEL Processes with Dynamo and the JBoss Rule Engine." In *International Workshop on Engineering of Software Services for Pervasive Environments: In Conjunction with the 6th ESEC/FSE Joint Meeting, ESSPE '07*, 11–20. New York, NY: ACM. doi:10.1145/1294904.1294906.

Braem, M., N. Joncheere, W. Vanderperren, R. Van Der Straeten, and V. Jonckers. 2007. "Concern-Specific Languages in a Visual Web Service Creation Environment." *Electronic Notes in Theoretical Computer Science* 163 (2): 3–17. Proceedings of the Second International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems (ABMB 2006). doi:10.1016/j.entcs.2006.10.012.

Castor, F., D. S. Leite, and C. M. F. Rubira. 2012. "An Exception Handling System for Service Component Architectures." *Journal of the Brazilian Computer Society* 18 (1): 43–59. doi:10.1007/s13173-012-0059-5.

Cemus, K., T. Cerny, and M. J. Donahoo. 2015. "Automated Business Rules Transformation into a Persistence Layer." *Procedia Computer Science Journal* 62: 312–318. doi:10.1016/j.procs.2015.08.391.

Cemus, K., F. Klimes, O. Kratochvil, and T. Cerny. 2017. "Separation of Concerns for Distributed Cross-Platform Context-Aware User Interfaces." *Cluster Computing* 20 (3): 2355–2362. doi:10.1007/s10586-017-0794-7.

Cerny, T., K. Cemus, M. J. Donahoo, and E. Song. 2013. "Aspect-Driven, Datareflective and Context-Aware User Interfaces Design." *ACM SIGAPP Applied Computing Review* 13 (4): 53–66. doi:10.1145/2577554.

Cerny, T., and M. J. Donahoo. 2016. "Survey on Concern Separation in Service Integration." In *Proceedings of the 42nd International Conference on SOFSEM 2016: Theory and Practice of Computer Science Volume 9587*, 518–531. New York, NY, USA: Springer-Verlag New York. doi:10.1007/978-3-662-49192-8_42.

Cerny, T., and M. J. Donahoo. 2015. "On Separation of Platform-Independent Particles in User Interfaces." *Cluster Computing* 18 (3): 1215–1228. doi:10.1007/s10586-015-0471-7.

Cerny, T., M. J. Donahoo, and M. Trnka. 2018. "Contextual Understanding of Microservice Architecture: Current and Future Directions." *SIGAPP Applications Computation Reviews* 17 (4): 29–45. doi:10.1145/3183628.3183631.

Cerny, T., M. Trnka, and M. J. Donahoo. 2016. "Towards Shared Security through Distributed Separation of Concerns." In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems, RACS '16*, 169–172. New York, NY: ACM. doi:10.1145/2987386.2987394.

Charfi, A., and M. Mezini. 2004. "Hybrid Web Service Composition: Business Processes Meet Business Rules." In *Proceedings of the 2nd International Conference on Service Oriented Computing, ICSOC '04*, 30–38. New York, NY: ACM. doi:10.1145/1035167.1035173.

Cibrán, M. A., B. Verheecke, W. Vanderperren, D. Suvée, and V. Jonckers. 2007. "Aspect-Oriented Programming for Dynamic Web Service Selection, Integration and Management." *World Wide Web* 10 (3): 211–242. doi:10.1007/s11280-006-0017-2.

Corradi, A., F. Di Marco, S. Monti, and S. Pasini. 2009. "Facing Crosscutting Concerns in a Middleware for Pervasive Service Composition." In *2009 IEEE Symposium on Computers and Communications*, 73–79. Sousse: IEEE.

Cugola, G., C. Ghezzi, and L. S. Pinto. 2012. "DSOL: A Declarative Approach to Self-Adaptive Service Orchestrations." *Computing* 94 (7): 579–617. doi:10.1007/s00607-012-0194-z.

Deepiga, A. S., V. S. Senthil, and C. Babu. 2014. "Empirical Investigation of Introducing Aspect Oriented Programming across Versions of an SOA Application." In *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, 1732–1739. Ramanathapuram: IEEE.

Dell'Amico, M., G. Serme, M. S. Idrees, A. S. De Olivera, and Y. Roudier. 2012. *HiPoLDS: A Security Policy Language for Distributed Systems*, 97–112. Berlin, Heidelberg:Springer Berlin Heidelberg. doi:10.1007/978-3-642-30955-7_10.

Ermagan, V., I. H. Krüger, and M. Menarini. 2008. "Aspect-Oriented Modeling Approach to Define Routing in Enterprise Service Bus Architectures." In *Proceedings of the 2008 International Workshop on Models in Software Engineering, MiSE '08*, 15–20. New York, NY: ACM. doi:10.1145/1370731.1370735.

Giunta, R., F. Messina, G. Pappalardo, and E. Tramontana. 2015. "Providing QoS Strategies and Cloud-Integration to Web Servers by Means of Aspects." *Concurrency and Computation: Practice and Experience* 27 (6): 1498–1512. doi:10.1002/cpe.3031.

Guinard, D., V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. 2010. "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services." *IEEE Transactions on Services Computing* 3 (3): 223–235. doi:10.1109/TSC.2010.3.

Han, H., H. Jung, and H. Y. Yeom. 2011. "Aspect-Oriented Development of Cluster Computing Software." *Cluster Computing* 14 (4): 357–375. doi:10.1007/s10586-011-0166-7.

Harrison, W. 2011. "Modularity for the Changing Meaning of Changing." In *Proceedings of the Tenth International Conference on Aspect-Oriented Software Development, AOSD '11*, 301–312. New York, NY: ACM. doi:10.1145/1960275.1960313.

Hassan, S., N. Ali, and R. Bahsoon. 2017. "Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity." In *2017 IEEE International Conference on Software Architecture (ICSA)*, 1–10. Gothenburg: IEEE.

He, W., and L. D. Xu. 2014. "Integration of Distributed Enterprise Applications: A Survey." *IEEE Transactions on Industrial Informatics* 10 (1): 35–42. doi:10.1109/TII.2012.2189221.

Hmida, M. M. B., R. F. Tomaz, and V. Monfort. 2005. "Applying AOP Concepts to Increase Web Services Flexibility." In *International Conference on Next Generation Web Services Practices (NWeSP'05)*, 6. August.

Huang, T., G.-Q. Wu, and J. Wei. 2009. "Runtime Monitoring CompositeWeb Services through Stateful Aspect Extension." *Journal of Computer Science and Technology* 24 (2): 294–308. doi:10.1007/s11390-009-9225-4.

Issarny, V., G. Bouloukakis, N. Georgantas, and B. Billet. 2016. "Revisiting Service-Oriented Architecture for the IoT: A Middleware Perspective." In *Service-Oriented Computing*, edited by Q. Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri, 3–17. Cham: Springer International Publishing.

Kiczales, G., J. Irwin, J. Lamping, J.-M. Loingtier, C. V. Lopes, C. Maeda, and A. Mendhekar. 1997. "Aspect-Oriented Programming." In *IECOOP'97- Object-Oriented Programming, 11th European Conference*. Vol. 1241. 220–242. Jyväskylä: Springer. June.

Kim, J. P., and J. E. Hong. 2011. "Dynamic Service Replacement to Improve Composite Service Reliability." In *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement*, 182–188. Jeju Island: IEEE.

Kratzke, N., and P.-C. Quint. 2017. "Understanding Cloud-Native Applications after 10 Years of Cloud Computing A Systematic Mapping Study." *Journal of Systems and Software* 126: 1–16. doi:10.1016/j.jss.2017.01.001.

Krüger, I. H., R. Mathew, and M. Meisinger. 2006. "Efficient Exploration of Service-Oriented Architectures Using Aspects." In *Proceedings of the 28th International Conference on Software Engineering, ICSE '06*, 62–71. New York, NY: ACM. doi:10.1145/1134285.1134296.

Laddad, R. 2009. *AspectJ in Action: Enterprise AOP with Spring Applications*. 2nd ed. Greenwich, CT: Manning Publications.

Liu, X., W. J. Zhang, R. Radhakrishnan, and Y. L. Tu. 2008. "Manufacturing Perspective of Enterprise Application Integration: The State of the Art Review." *International Journal of Production Research* 46 (16): 4567–4596. doi:10.1080/00207540701263325.

Marzouk, S., and M. Jmaiel. 2013. "A Policy-Based Approach for Strong Mobility of Composed Web Services." *Service Oriented Computing and Applications* 7 (4): 293–315. doi:10.1007/s11761-013-0131-9.

Mosser, S., G. Mussbacher, M. Blay-Fornarino, and D. Amyot. 2011. "From Aspect-Oriented Requirements Models to Aspect-Oriented Business Process Design Models: An Iterative and Concern-Driven Approach for Software Engineering." In *Proceedings of the Tenth International Conference on Aspect-Oriented Software Development, AOSD '11*, 31–42. New York, NY: ACM. doi:10.1145/1960275.1960281.

Peinado, S., G. Ortiz, and J. M. Dodero. 2015. "A Metamodel and Taxonomy to Facilitate Context-Aware Service Adaptation." *Computers & Electrical Engineering* 44: 262–279. doi:10.1016/j.compeleceng.2015.02.004.

Petersen, K., S. Vakkalanka, and L. Kuzniarz. 2015. "Guidelines for Conducting Systematic Mapping Studies in Software Engineering: An Update." *Information and Software Technology* 64 (Supplement C): 1–18. doi:10.1016/j.infsof.2015.03.007.

Prasad, G. V. R. J. S., S. Chimalakonda, V. Choppella, and Y. Raghu Reddy. 2017. "An Aspect Oriented Approach for Renarrating Web Content." In *Proceedings of the 10th Innovations in Software Engineering Conference, ISEC '17*, 56–65. New York, NY: ACM. doi:10.1145/3021460.3021466.

Rose, S. J., D. W. Engel, N. O. Cramer, and W. E. Cowley. 2010. "Automatic Keyword Extraction from Individual Documents." *Text Mining: Applications and Theory,* 1–20. Wiley-Blackwell. doi:10.1002/9780470689646.ch1

Seiler, D., E. Juhnke, R. Ewerth, M. Grauer, and B. Freisleben. 2011. "Efficient Data Transmission between Multimedia Web Services via Aspect-Oriented Programming." In *Proceedings of the Second Annual ACM Conference on Multimedia Systems, MMSys '11*, 93–104. New York, NY: ACM. doi:10.1145/1943552.1943564.

Sendor, J., Y. Lehmann, G. Serme, and A. Santana De Oliveira. 2014. "Platform-Level Support for Authorization in Cloud Services with OAuth 2." In *2014 IEEE International Conference on Cloud Engineering*, 458–465. Boston, MA: IEEE.

Sonchaiwanich, E., J. Zhao, C. Dowin, and M. McRoberts. 2010. "Using AOP to Separate SOA Security Concerns from Application Implementation." In *2010 MILCOM 2010 Military Communications Conference*, 470–474. San Jose, CA: IEEE.

Souza, A., C. Capelli, F. Santoro, L. G. Azevedo, J. C. S. D. P. Leite, and T. Batista. 2011. "Service Identification in Aspect-Oriented Business Process Models." In *Proceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE)*, 164–174. Irvine, CA: IEEE.

Spieldenner, T., S. Byelozyorov, M. Guldner, and P. Slusallek. 2017. "FiVES: An Aspect-Oriented Virtual Environment Server." In *2017 International Conference on Cyberworlds (CW)*, 103–110. Chester: IEEE.

Wada, H., J. Suzuki, and K. Oba. 2008. "Early Aspects for Non-Functional Properties in Service Oriented Business Processes." In *2008 IEEE Congress on Services Part I*, 231–238. Honolulu, HI: IEEE.

Walraven, S., B. Lagaisse, E. Truyen, and W. Joosen. 2010. *Dynamic Composition of Cross-Organizational Features in Distributed Software Systems*, 183–197. Berlin, Heidelberg:Springer Berlin Heidelberg. doi:10.1007/978-3-642-13645-0_14.

Waltman, L., and N. J. van Eck. 2012. "A New Methodology for Constructing A Publicationlevel Classification System of Science." *Journal of the American Society for Information Science and Technology* 63 (12): 2378–2392. doi:10.1002/asi.22748.

Wang, J., D. Liu, W. H. Ip, W. Zhang, and R. Deters. 2014a. "Integration of System-Dynamics, AspectProgramming, and Object-Orientation in System Information Modeling." *IEEE Transactions on Industrial Informatics* 10 (2): 847–853. doi:10.1109/TII.2014.2300703.

Wang, J. W., H. F. Wang, J. L. Ding, K. Furuta, T. Kanno, W. H. Ip, and W. J. Zhang. 2016. "On Domain Modelling of the Service System with Its Application to Enterprise Information Systems." *Enterprise Information Systems* 10 (1): 1–16. Cited By 5. doi:10.1080/17517575.2013.810784.

Wang, J. W., H. F. Wang, W. J. Zhang, W. H. Ip, and K. Furuta. 2014b. "On a Unified Definition of the Service System: What Is Its Identity?" *IEEE Systems Journal* 8 (3): 821–826. doi:10.1109/JSYST.2013.2260623.

Wang, M., K. Y. Bandara, and C. Pahl. 2010. "Distributed Aspect-Oriented Service Composition for Business Compliance Governance with Public Service Processes." In *2010 Fifth International Conference on Internet and Web Applications and Services*, 339–344. Barcelona: IEEE.

Wang, Y., and Y. Wang. 2013. "A Survey of Change Management in Service-Based Environments." *Service Oriented Computing and Applications* 7 (4): 259–273. doi:10.1007/s11761-013-0128-4.

Wieringa, R., N. Maiden, N. Mead, and C. Rolland. 2006. "Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and A Discussion." *Requirements Engineering* 11 (1): 102–107. doi:10.1007/s00766-005-0021-6.

Xu, G., J. Wang, G. Q. Huang, and C. H. Chen. 2017. "Data-Driven Resilient Fleet Management for Cloud Asset-Enabled Urban Flood Control." *IEEE Transactions on Intelligent Transportation Systems PP* 99: 1–12.

Yumei, W., C. Lu, R. Lina, and Z. Lin. 2009. "AN AOSD-based Method for Integrated Services Implementation in Telecommunication Field." In *2009 2nd IEEE International Conference on Broadband Network Multimedia Technology*, 772–776. Beijing: IEEE.

Zhang, J., F. Meng, and G. Liu. 2008. "Research on SOA-Based Applications Based on AOP and Web Services." In *2008 International Conference on Computer and Electrical Engineering*, 753–757. Phuket: IEEE.

Zhang, J., F. Meng, and G. Liu. 2009. "Research on Multi-Tier Distributed Systems Based on AOP and Web Services." In *2009 First International Workshop on Education Technology and Computer Science*, Vol. 2, 203–207. Wuhan: IEEE. March.