Automating the Dynamic Interactions of Self-governed Components in Distributed Architectures

Sebastian R. Bader(✉)

Institute AIFB, Karlsruhe Institute of Technology (KIT), 76133 Karlsruhe, Germany
sebastian.bader@kit.edu

Abstract. The ongoing digitalization and penetration of the Web into each aspect of software development creates new possibilities and challenges. The flexible reuse of components promises to drastically reduce the implementation and maintenance effort. But growing complexity in terms of variety and dynamic changes bring monolithic approaches to their limits. In this paper, an approach is presented which enables components in distributed systems to observe, judge and independently react to dynamic changes in their neighborhood. Reducing the overall complexity to smaller and easier to manage subproblems leads to more flexible and reliable systems. The target is a delegation of decision making to the single components.

Keywords: Distributed systems · Component coordination · Dynamic web

1 Introduction

Initially, the Web started as a Web of documents. Through constant evolution we now see a more and more automatically processable Web of data and services. Semantic technologies allow specifications of characteristics and descriptions both interpretable for humans and machines. They take part to drive the Web from a static information provider to a decentralized interaction platform where data and functionalities are offered, accessed and consumed. The continuously growing number of publicly available Web APIs[1] emphasizes this development.

In addition, the current Web is already a dynamic environment in itself. Resources existing at one point in time can not be taken for granted, as providers stop hosting services, rebrand and relocate APIs. For example, even on the commercial IBM Bluemix platform, Web APIs can disappear without any transparent reason or machine processable information[2]. Contrary, APIs can still be available even if the provider marked them as inactive (e.g. the former Google Web Search API).

[1] http://www.programmableweb.com/api-research.

[2] https://concept-insights-demo.mybluemix.net/ Accessed 05.12.2016.

E. Blomqvist et al. (Eds.): ESWC 2017, Part II, LNCS 10250, pp. 173–183, 2017.
DOI: 10.1007/978-3-319-58451-5_12

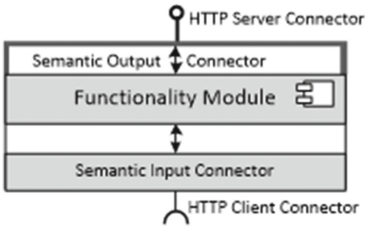

Fig. 1. Self-governed component: Functionality is wrapped by semantic connectors. The input connector independently establishes connections to other components.

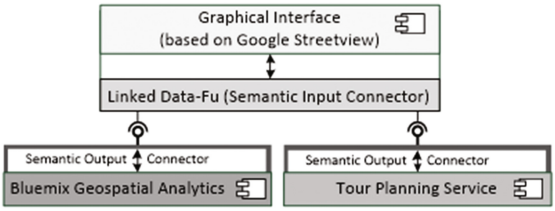

Fig. 2. Prototypical implementation of self-governed components, so far with only one connector (either input or output)

A methodology is outlined to enable distributed Web components to diagnose unforeseen changes and providing adaptive reaction capabilities. Components, as regarded in this work, are software modules which encapsulate a single functionality and are annotated with semantic descriptions. They can be initially designed Linked Data APIs but also occur in the form of translation instances for lifting and lowering the original Web API to Linked Data. The components communicate via RESTful Web APIs and have the ability to select and invoke other Web components autonomously, therefore they are denoted as *self-governed components* in the following.

A self-governed component (Fig. 1) consists of two semantic connectors responsible for its semantic descriptions, the data lifting and lowering, and provision and request to and from, but not limited to, other self-governed components. The offered core functionality is therefore independent of the communication methods and may directly be integrated by code, occur as a remote Web service or a database.

Parts of the concept of self-governed components were implemented for the use case of dispatching field technicians for industrial maintenance. As shown in Fig. 2, a IBM Bluemix analytic service for geospatial data[3] and a commercial tour planning heuristic have been enabled to RESTful communication with RDF through their respective *output connectors*[4]. The *input connector*, realized by a Linked Data-Fu [9] instance, consumes the Web APIs and fills a Google Street View based visualization component.

[3] https://console.ng.bluemix.net/catalog/services/geospatial-analytics.
[4] https://github.com/sebbader/BlueWrapper.

For now, the integration instructions are static rules and incrementally executed. In contrast, self-governed components need to be equipped with context-aware reaction capabilities. That includes mechanisms for detecting possible issues, recognizing and evaluating alternative partner components and establishing communication channels.

2 State of the Art

Components are regarded by Morrison [16] as "black boxes" with a strong focus on reusability. He states that in productive systems, the application does not require insights into the functionality of the used components but only on the delivered and consumed information packets. Microservices as described by Thönes [23] further limit the amount of provided functionalities to singular, easy to understand tasks. This drastically reduces the complexity of the necessary descriptions and eases the reuse in unforeseen scenarios.

The Semantic Web Stack [2] constitutes a set of technologies to handle both syntactical and semantical interoperability issues. It thereby defines the architecture of the Semantic Web with central technologies like URIs to identify resources, RDF to encode data, ontologies define meaning and the semantic query language SPARQL.

Semantic descriptions of Web components can be formulated in various languages and ontologies. Currently most important are the Web Service Ontology Language (WSMO) [20], OWL-S [14] and Linked USDL [19]. RESTdesc [26] utilizes a N3 Syntax to specify input and output parameters and how they are connected. Similarly, Dimou et al. [8] combine access information with data mappings in the RDF Mapping language (RML). Verborgh et al. [25] provide a survey on machine-interpretable Web API descriptions.

The types of descriptions can be organized in the categories behavioral, functional and non-functional. The technical details to operate the component are part of the behavioral sections whereas functional statements include basic information on the component's purpose. Non-functional information contain additional details on e.g. prices, provenance or provided Quality of Service.

Also, central registries for Web APIs like RapidAPI[5] or ProgrammableWeb[6] mostly do not provide semantic information, making an automated discovery a hard task. In the approach of Sande et al. [24] for Linked Data sets the data server recognizes other components by dereferencing its existing RDF data or utilizing the Referer Header of incoming HTTP requests and therefore gains knowledge about other data sources.

In order to automatically combine the components, existing approaches [3,21] mostly use centralistic optimization during the design phase. Contrary, Web components are not static elements but can and do change over time. In general, they follow a life cycle as shown by Wittern and Fischer [27]. Mayer et al. [15] extend RESTdesc to cope with a dynamic environment by introducing states.

[5] https://www.rapidapi.com/.
[6] https://www.programmableweb.com/category/all/apis.

Similarly, Alaya et al. suggest oneM2M, a IoT approach to gain machine-to-machine (M2M) interoperability with a semantic reasoner [1]. But still, a central organizer with knowledge about the whole network is required.

One way to enable a more flexible way to determine component compositions are policies. The non-functional characteristics of available components are regarded with semantic reasoners [22] in order to match and rank (Palmonari et al. [17]) them against predefined requirements. La Torre et al. [13] propose the dynamic context of the consuming client as a selection criteria for components. Context here is regarded as social media information of e.g. Facebook but also physical data like GPS coordinates. Although only human users have been regarded, it should be possible to transfer the approach to automated components, having context like the location of the hosted server or the company running it.

3 Problem Statement and Contributions

As outlined, self-governed components can perceive spontaneous changes and adjustments. In particular, changes of the produced data model and the technical interaction patterns in addition to modifications of functionality and the component's location are possible without former notice. Together with the rising number of heterogeneous components, the complexity of coordinating the individual elements of Web applications increases further. No central coordinator can guarantee sufficient performance when no specified size limits can be set and potential changes of components can happen at any time.

Although switching from central to distributed architectures can reach supplies the necessary scalability, it does not solve the problem of sudden changes of components. While in the first case a central coordinator was responsible, the adjustments now have to accomplished by the self-governed components themselves. Therefore, this paper focuses to answer the following research question:

Research Question: *How can self-governed components in distributed architectures cope with dynamic and unforeseen changes of other self-governed components which they depend on?*

3.1 Validating and Updating Semantic Descriptions of Web Components

The Web is a dynamic environment where resources and services are not static and may appear, disappear, and change their characteristics spontaneously. If an API or the functionality of an used self-governed component changes, an affected consuming component will only recognize it when its procedures begin to fail.

Even though the provider may announce the modifications upfront, it is usually not done via machine interpretable channels. Self-governed components in critical applications therefore need analytical mechanisms to self-detect such incidents. In the example, the GeospatialAnalytics and the tour planning component provide data in the WGS84 format. An update e.g. could set the tour

planning component back to its default settings, where country, address and street name specify a location instead of WGS84 coordinates.

Bhargava and Lingayat [4] try to tackle the topic with local and global monitoring components to discover validations of service level agreements but do not regard any (semantic) descriptions. SHACL [12] on the other hand allows the comparison of incoming Linked Data against expected patterns but can hardly cope with non-functional aspects. Consequently, the following question has to be answered:

RQ 1. *How can the changes in functional, behavioral and non-functional descriptions of self-governed components be validated and, in the case of mismatches, be modified?*

3.2 Spontaneous Connector for Self-governed Components

Although self-governed components share a common stack of technologies like URIs, semantic interfaces and RDF, they still allow nearly endless variations of implementations. A self-governed component which identified a suitable functional input source is not capable to simply consume the API without further specification. Missing data mappings, unknown interface invocation and other behavioral requirements prohibit a plug-and-play like connection. In the described example, the self-governed UI component has to be able to switch from the commercial tour planning to e.g. an instance of the open-source OpenTripPlanner[7] (see Fig. 3):

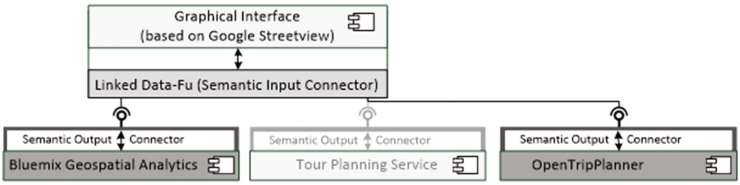

Fig. 3. Exchanging the tour planning component with the similar OpenTripPlanner

For that, Keppmann et al. [11] introduce adjustable "Smart Components" which can change their program code – and thereby also their data sources – during runtime. Nevertheless, these components can not individually customize their connectors to alternative sources. In contrast to that, the system of Bhargava and Lingayat [4] dynamically configures the network but relies on a central coordinator which poses all information about the environment.

This leads to the following research question:

RQ 2. *How can one self-governed component autonomously consume another without formerly specifying connection details and requirements?*

[7] http://www.opentripplanner.org/.

3.3 Delegation of Communication Responsibility in Distributed Architectures

In the example, only the self-governed UI component once has to find a replacement for its data provider. No other component is affected as long as the UI component can find a sufficient substitute. This leads to the question how self-governed components know which providers to select in order to gain the required input data.

Cao et al. [6] solve dynamic composition in P2P networks where the participants iteratively create a workflow chain. Although they regard nonfunctional characteristics, the parameters of interest have to be introduced during design time. Additional desired parameter can not be considered. Similarly, Cardellini et al. [7] do not regard the inherent complexity of nonfunctional requirements.

Policies like in [18,22] enable the components to act independently in a surrounding with incomplete information and spontaneously occurring changes. Comprehensible methods have to be developed or adjusted in order to rank available candidates. The consuming self-governed component has to conclude first whether an offering component complies with the defined policies, then select the most appropriate, and establish a connection:

RQ 3. *How can self-governed components in distributed architectures independently derive selection criteria from abstract policies in order to appropriately classify available, alternative self-governed components?*

Summarizing, RQ 1 targets the discovery of evolving problems, RQ 2 examines methods to technically enable reactions and RQ 3 develops approaches to select suitable reactions. In combination, they enable self-governed components to state, if necessary, how and in which manner they react to dynamic changes in distributed architectures and thereby answer the main research question.

4 Research Methodology and Approach

The self-governed components as regarded in this paper rely on the Semantic Web Stack. In particular, URIs are used to identify and locate components, in combination with RDF as the data model, and ontologies to reason about delivered data and semantic descriptions. In addition, this paper only regards self-governed components with RESTful APIs, based on HTTP communication.

The research approach is directly determined by the dependencies between the proposed modules.

The treatment of changing API descriptions (Sect. 4.1) relies on the spontaneous establishment of communication channels (Sect. 4.2) and vice versa. Both parts together will allow to solve the problem of decentralizing coordination responsibility (Sect. 4.3) in order to answer the main research question.

4.1 Validating and Deriving Descriptions of Self-governed Components

The required methods regarding the semantic description of APIs are divided into two tasks. In the first part, component descriptions are regarded as fixed

facts. They are compared against the observed data and communication pattern. The aim is to detect inconsistencies caused by e.g. applied upgrades or API changes of the observed self-governed components. On-the-fly reasoning of transferred RDF data allows the recognition of conflicting statements of the received data with their functional and non-functional descriptions. Violations in terms of behavioral aspects usually lead to transaction errors which have to be interpreted separately. Non-functional aspects need a more advanced handling. A mapping of behavior variants to an ontology will be developed to gain inputs for the semantic reasoner.

In addition to methods for validating assumptions, the automated proposal of semantic descriptions is regarded. Independently recognizing component's characteristics is essential to increase the amount of useful and machine interpretable descriptions in the Web as it lowers the effort in both time and necessary skills to deploy a self-governed component. Therefore, benefits for both the consumer (having detailed control mechanisms) and the provider (reducing the manual effort) can be accomplished.

4.2 Spontaneous Connector for Self-governed Components

Consuming components can interact with suppliers on the basis of their semantic descriptions. Data can be easily transmitted in the case of matching demanded and available resources. Nevertheless, in some cases meaningful interactions can still be accomplished although specifications and demands do not fit perfectly. Therefore, the approach will relax the retrieved descriptions of self-governed components. It is to verify whether neglecting parts of stated constraints improves the automated connection of components. Even though the intentional violation will produce mismatches, it has the potential to solve situations where overly restrictive descriptions prevent interactions.

The Linked Data streaming engine Linked Data-Fu as the input connector will serve as the mediator between two components. It is capable to request, process and forward RDF data. Linked Data-Fu will be extended towards an autonomously deciding connector. The challenge is to derive the interaction instructions solely relying on the component's (potentially mismatching) descriptive data and predefined ontological knowledge. For that, Linked Data-Fu provides on the fly semantic reasoning and SPARQL query execution which serve as the foundation for further developments.

4.3 Delegation of Communication Responsibility

The previously developed methods will be combined in a framework for delegating coordination to the component. The framework defines how abstract policies have to be formulated to specify the expected behavior but on the other hand contain enough flexibility to find a matching self-governed component. In general, components are not deployed with exactly the required use case in mind, and therefore have at least slightly divergent descriptions. Consequently, the stated policies need to allow a certain degree of freedom.

The framework also specifies how these user defined policies can be operationalized regarding a faced situation. The derived rules serve to filter and rank the existing alternatives regarding functional and non-functional characteristics and thereby allow decentralized decision making. In order to gain a consistent behavior, the policies together with the derived rules are also transferred to the involved components. Therefore, each self-governed component is capable to configure its neighborhood independently but according to specified manners.

5 Preliminary Results

Currently, the work is in the initial experiment stage. Together with research and industry partners from the STEP project[8], a starting set of self-governed components from various domains is established. This will be the foundation of the a planed Web Component Network (see Sect. 6).

The first conceptual ideas have already been presented at SEMANTiCS 2016[9]. Regarding the domain of industrial maintenance scheduling, the ongoing digitalization increases the requirements for maintenance providers. The heterogeneity of interfaces, changing needs for functionalities and new business models reveal the inadequacy of existing monolithic systems. It was outlined how semantically enriched self-governed components can provide flexible integration into distributed architectures. The next steps are the creation of a testing and development environment. The Web itself is not suitable as conditions can neither be repeated nor sufficiently controlled. On the other hand the Web is the targeted habitat for the investigated self-governed components. Building on the existing Web Service Challenges [5] a sufficiently large set of various self-governed components will be established.

The descriptions of the created components will include incorrect and lacking annotations, syntactic and semantic errors and changes over time. The advantage of a controlled environment is the ability to control the mutations and thereby compare different strategies. The dynamic aspects, as (dis-)appearance and sudden modifications of components, will be implemented by both predefined, repeatable sequences and randomly triggered changes. Similarly to Joshi et al. [10] the system is configurable via seed parameters and creates dynamic but repeatable scenarios.

6 Evaluation Plan

This simulated Web environment will work in the same way and behave following the same dynamic principles as the real Web but at smaller and therefore better treatable scale. The target is a Gold Standard which serves as a testing and evaluating environment for the developed methods but will also be part of the

[8] https://www.projekt-step.de/.

[9] http://www.slideshare.net/semanticsconference/sebastian-bader-semantic-technologies-for-assisted-decisionmaking-in-industrial-maintenance.

contributions to the research community. The reproducible conditions of the network will allow other researchers to compare their approaches with the results of this research and to further improve the state of the art.

In order to judge the quality for recognition and adjusting of API descriptions of self-governed components (RQ 1) the performance will be measured with this environment.

Combining formerly unconnected Web Components (RQ 2) is a problem also faced in frameworks for service composition. In the described problem the connection effort shall be accomplished independently and self-organized by the component facing a problem. Nevertheless, the performance of tools like Medley [28] can be seen as baseline approaches.

In addition to the proposed evaluation environment, the developed concepts and implementations regarding the delegation of the interaction channels will also be implemented through the industry project STEP[10]. Existing company policies will be provided and automatically translated into technical instructions by the self-governed components. The resulting behavior is then compared to the preferred choices of responsible managers.

7 Conclusion

The variety and amount of available components is a significant advantage of applications running in the Web. A well known set of standard protocols and principles enables the fast reuse of software functionalities. But existing Web based services show that the heterogeneity of their behavior and unforeseeable changes in the implementations require constant manual adjustments. Therefore, a stronger utilization is prohibited.

Self-governed components are one method to enhance the degree of automatization in distributed architectures. The proposed methods target the local decision making to further empower the single components. The ability to independently react on changes minimizes required maintenance of distributed Web applications and decreases the barriers of component reuse. This results in faster deployments, decreasing maintenance efforts and reduced complexity. The additionally provided evaluation environment makes the proposed approaches comparable and opens the paths to continuous improvements.

References

1. Alaya, M.B., Medjiah, S., Monteil, T., Drira, K.: Toward semantic interoperability in oneM2M architecture. IEEE Commun. Mag. **53**(12), 35–41 (2015)
2. Berners-Lee, T., Weitzner, D.J., Hall, W., O'Hara, K., Shadbolt, N., Hendler, J.A.: A framework for web science. Found. Trends Web Sci. **1**(1), 1–130 (2006)
3. Beygelzimer, A., Riabov, A., Sow, D., Turaga, D.S., Udrea, O.: Big data exploration via automated orchestration of analytic workflows. In: ICAC 2013, pp. 153–158 (2013)

[10] https://www.projekt-step.de/en/.

4. Bhargava, B., Angin, P., Ranchal, R., Lingayat, S.: A distributed monitoring and reconfiguration approach for adaptive network computing, pp. 31–35. IEEE (2015)
5. Bleul, S., Weise, T., Geihs, K.: The web service challenge-a review on semantic web service composition. Electron. Commun. EASST 17 (2009)
6. Cao, X., Kapahnke, P., Klusch, M.: SPSC: Efficient composition of semantic services in unstructured P2P networks. In: Gandon, F., Sabou, M., Sack, H., d'Amato, C., Cudré-Mauroux, P., Zimmermann, A. (eds.) ESWC 2015. LNCS, vol. 9088, pp. 455–470. Springer, Cham (2015). doi:10.1007/978-3-319-18818-8_28
7. Cardellini, V., D'Angelo, M., Grassi, V., Marzolla, M., Mirandola, R.: A decentralized approach to network-aware service composition. In: Dustdar, S., Leymann, F., Villari, M. (eds.) ESOCC 2015. LNCS, vol. 9306, pp. 34–48. Springer, Cham (2015). doi:10.1007/978-3-319-24072-5_3. http://www.ce.uniroma2.it/publications/esocc2015_xweb.pdf
8. Dimou, A., Verborgh, R., Sande, M.V., Mannens, E., Van de Walle, R.: Machine-interpretable dataset and service descriptions for heterogeneous data access and retrieval, pp. 145–152. ACM Press (2015)
9. Harth, A., Knoblock, C.A., Stadtmller, S., Studer, R., Szekely, P.: On-the-fly integration of static and dynamic linked data. In: COLD 2013, pp. 1–12 (2013)
10. Joshi, A.K., Hitzler, P., Dong, G.: LinkGen: Multipurpose linked data generator. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) ISWC 2016. LNCS, vol. 9982, pp. 113–121. Springer, Cham (2016). doi:10.1007/978-3-319-46547-0_12
11. Keppmann, F.L., Maleshkova, M., Harth, A.: Semantic technologies for realising decentralised applications for the web of things. In: ICECCS, pp. 71–80 (2016)
12. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL) (2017). http://www.w3.org/TR/shacl/
13. La Torre, G., Monteleone, S., Cavallo, M., D'Amico, V., Catania, V.: A context-aware solution to improve web service discovery and user-service interaction, pp. 180–187. IEEE (2016)
14. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T.: OWL-S: Semantic markup for web services. W3C Member Submission 22 (2004). 2007–04
15. Mayer, S., Verborgh, R., Kovatsch, M., Mattern, F.: Smart configuration of smart environments. IEEE Trans. Autom. Sci. Eng. **13**(3), 1247–1255 (2016)
16. Morrison, J.P.: Flow-based Programming. In: Proceedings of the 1st International Workshop on Software Engineering for Parallel and Distributed Systems, pp. 25–29 (1994)
17. Palmonari, M., Comerio, M., Paoli, F.: Effective and flexible NFP-based ranking of web services. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC/ServiceWave -2009. LNCS, vol. 5900, pp. 546–560. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10383-4_40
18. Paoli, F.D., Palmonari, M., Comerio, M., Maurino, A.: A meta-model for non-functional property descriptions of web services, pp. 393–400. IEEE (2008)
19. Pedrinaci, C., Cardoso, J., Leidig, T.: Linked USDL: A vocabulary for web-scale service trading. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 68–82. Springer, Cham (2014). doi:10.1007/978-3-319-07443-6_6
20. Roman, D., Lausen, H., Keller, U.: Web Service Modeling Ontology (WSMO) (2006). http://www.wsmo.org/TR/d2/v1.3/
21. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Web Semant. **1**(4), 377–396 (2004)

22. Speiser, S.: Semantic annotations for WS-Policy. In: ICWS, pp. 449–456 (2010)
23. Thönes, J.: Microservices. IEEE Softw. **32**(1), 116–116 (2015)
24. Sande, M.V., Verborgh, R., Dimou, A., Colpaert, P., Mannens, E.: Hypermedia-based discovery for source selection using low-cost linked data interfaces. IJSWIS **12**(3), 79–110 (2016)
25. Verborgh, R., Harth, A., Maleshkova, M., Stadtmller, S., Steiner, T., Taheriyan, M., Van de Walle, R.: Survey of semantic description of REST APIs. In: Pautasso, C., Wilde, E., Alarcon, R. (eds.) REST Advanced Research Topics and Practical Applications, pp. 69–89. Springer, New York (2014)
26. Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., Van de Walle, R., Vallés, J.G.: Description and interaction of restful services for automatic discovery and execution. In: International Workshop on AFMS. FTRA (2011)
27. Wittern, E., Fischer, R.: A life-cycle model for software service engineering. In: Lau, K.-K., Lamersdorf, W., Pimentel, E. (eds.) ESOCC 2013. LNCS, vol. 8135, pp. 164–171. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40651-5_13
28. Yahia, E.B.H., Réveillère, L., Bromberg, Y.-D., Chevalier, R., Cadot, A.: Medley: An event-driven lightweight platform for service composition. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 3–20. Springer, Cham (2016). doi:10.1007/978-3-319-38791-8_1