Focus auf Komposition von Bausteinen. Weniger Beispiel als in dem HeraklitCafe von der Modellierung.

Modelling service-oriented systems and cloud services with Heraklit

Peter Fettke[1,2][0000−0002−0624−4431] and Wolfgang Reisig[3][0000−0002−7026−2810]

[1] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
peter.fettke@dfki.de
[2] Saarland University, Saarbrücken, Germany
[3] Humboldt-Universitt zu Berlin, Berlin, Germany
reisig@informatik.hu-berlin.de

Abstract. Modern and next generation digital infrastructures are technically based on service oriented structures, cloud services, and other architectures that compose large systems from smaller subsystems. The composition of subsystems is particularly challenging, as the subsystems themselves may be represented in different languages, modelling methods, etc. It is quite challenging to precisely conceive, understand, and represent this kind of technology, in particular for a given level of abstraction. To capture refinement and abstraction principles, various forms of "technology stacks" and other semi-formal or natural language based on presentations have been suggested. Generally, useful concepts to compose such systems in a systematic way are even more rare. Heraklit provides means, principles, and unifying techniques to model and to analyze digital infrastructures. Heraklit integrates composition and hierarchies of subsystems, concrete and abstract data structures, as well as descriptions of behaviour. A distinguished set of means supports the modeler to express their ideas. The modeller is free to choose the level of abstraction, as well as the kind of composition. Heraklit integrates new concepts with tried and tested ones. Such a framework provides the foundation for a comprehensive Systems Mining as the next step after Process Mining.

Keywords: systems composition · data modelling · behaviour modelling · composition calculus · algebraic specification · Petri nets · Systems Mining

1 Introduction

The development of big service-oriented systems is challenging. Traditionally, models have been a central tool for designing such systems. Currently used modelling methods reach their limits and should be replaced by better concepts. The currently prevailing way of developing service-oriented systems is unsatisfactory in many aspects. The development process and its result must be: (a)

presented at the *16th International Workshop on Engineering Service-Oriented Applications and Cloud Services*, Heraklion, Greece, September 28-30, 2020 more manageable for the developer, (b) easy to understand for the user, (c) less error-prone and verifiable, (d) easier to change, faster reachable and cheaper especially for really large systems. These and similar requirements have long been discussed in the relevant literature.

The development of a complex service-oriented system is always preceded by a planning process in which models are used to formulate the structure, function, intended effects etc. of the intended product. In comparison to other engineering disciplines, models are generally not used very often in computer science and business informatics. This is mainly due to the fact that up to now not much benefit can be derived from models. In the practice of system design nowadays mainly diagrams using the Business Process Modeling Notation (BPMN) are propagated for describing the business logic. Such diagrams are limited to the identification of elementary activities and the representation of the control flow. More comprehensive models that take more aspects into account and are more intuitive would be extremely helpful for computer science and business informatics.

We argue for a modelling method whose models are suitable for much more than just the representation of elementary activities and control flows. In particular, a good modelling method should meet the requirements mentioned above. From a more technical point of view, such a method should:

– support the structuring of a large service-oriented systems into modules;
– technically simple, but expressive to compose modules to large service-oriented systems;
– describe the discrete steps in large systems only locally in individual modules;
– represent modules intended for implementation and modules not intended for implementation integrated with the same concepts;
– represent data and consider data dependencies in the control flow;
– abstract from concrete data in order to create instantiations with the same behavior in a schematic way;
– add under-specified data aspects in the later design process or in the event of changes of the system systematically;
– describe activities and events at any level of abstraction and hierarchy levels;
– generate models that are scalable, changeable and expandable;
– support the proof that a model has desired properties;
– extend the proven techniques of Data Mining and Process Mining to general Systems Mining.

In this paper we propose Heraklit as a modelling method that meets these requirements. It combines proven mathematically based and intuitively easy to understand concepts that are already used for system specification; we recombine them and complement them with concepts for composition and hierarchical refinement of local components, making this technique suitable for modeling large operational systems.

The objective of this paper is to present an overview on Heraklit. Therefore we shortly introduce the central modelling principles in Section 2. Section 3 presents a concise case study using Heraklit. The paper closes with a disucssion of related work (Section 4) and some conclusions (Section 5).

2 Principles of HERAKLIT

2.1 Big systems

What are the implications of the statement that a system is "big"? Firstly, some concepts that suit "small" systems do not suit large systems. One of the most obvious of these concepts is the assumption of global states and steps that update global states [17]. Global states and steps adequately describe, for example, the behaviour of s small digital circuit. To describe the behaviour of stakeholders of a business as a sequence of global steps, is, however, conceptually not adequate. In a big system, e.g. a business, cause and effect of a step are locally confined; and this confinement is essential to understand behaviour. As another specific concept, a big system requires conventions to confine validity of names, i.e. to avoid globally valid names, with a few exceptions such as URLs.

In HERAKLIT, single behaviours (runs, executions) of a subsystem can be represented by means of states and steps that are global only within the subsystem. Upon composing two such systems, those local states and steps are not necessarily embedded into global states and steps of the composed system. Instead, single behaviours of the composed system are represented without assuming global states and steps. Local names of a subsystem are confined to the subsystem and its direct neighboring subsystems.

2.2 Composition of systems

Every "big" real life system is composed from subsystems that are mutually related: they may exchange messages or jointly execute activities. The composition of subsystems is particularly challenging, as the subsystems themselves may be represented in different languages, modelling methods, etc.[6] Modelling techniques for such systems must provide means to compose models of subsystems. Many modelling techniques provide such means; they all come with specific, frequently parameterized composition operators, concentrating on special ways to exchange data, e.g. synchronously or asynchronously. A "big" system, composed from many systems $S_1, \ldots, S_n$, is favorably written

$$S = S_1 \bullet \cdots \bullet S_n, \tag{1}$$

with "$\bullet$" being any version of a composition operator. This bracket free notation requires that the composition operator is associative, i.e. that for any three models R, S, T hold: $(R \bullet S) \bullet T = R \bullet (S \bullet T)$. Typical examples for the notation (1) include supply chains, sequences of production machines in a factory, etc. Associativity of composition is rarely discussed explicitly, but frequently assumed without saying [16].

HERAKLIT offers a simple, universally employable and associative composition operator. In HERAKLIT, the diversity of specific, parameterized composition operators is expressed by help of *adapters*: Specific aspects and properties of the composition $R \bullet S$ of two models R and S are formulated in an adapter A, such that $R \bullet A \bullet S$ expresses the wanted properties. The advantages of this concept are obvious: One technical composition operator fits all content-wise requirements, adapters can themselves be composed, etc.

2.3 Abstraction and refinement

A number of general principles has been proposed in literature, to adequately cover the abstraction and refinement of systems. In particular, it is most useful to start out with an abstract specification and to refine it systematically, such that properties of the refined system imply the relevant properties of the abstract system. Vice versa, a given system may be abstracted, yielding a more compact version.

Abstraction and refinement should harmonize with the composition. To refine a part T of a system S, one would partition S into T and the environment of T, and then refine T. The remaining subsystems in the environment of T should not be affected by this procedure. Systems on different refinement levels should be composable; an overall concept of hierarchy levels for subsystems should not be required. HERAKLIT suggests concepts for refinement and abstraction that respect these requirements.

2.4 Modelling of data and things equally

In a big system, data, physical items, algorithms, activities of persons, steps of organizations, etc., are entangled. They must be modelled by similar means that differentiate between them only in pragmatical aspects: data can be generated, deleted, transformed into different representations, manipulated by computers, copied, updated, composed, etc. Physical items behave differently: A physical item always occupies a distinguished place in space. In models, one frequently does not want to distinguish “equal” items explicitly; their number matters.

2.5 Behaviour

The behaviour of a large system is composed of single actions. An action updates some local state components. It is up to the modeler to embed local state components into more global views, if wanted. For a really large system, a single execution (run) should not be represented as a sequence of actions (though one may argue that all behaviour occurs along a global time scale). Independence of actions should explicitly be represented and not be spoiled by representing them in an arbitrary order.

HERAKLIT suggests to base the description of behaviour on Petri nets with data carrying tokens [15]. This choice is motivated by multiple aspects:

- Petri nets can easily be specialized to include interfaces: Just select some places, transitions, and even arcs to serve as interface elements.
- The composition of Petri nets with interfaces is again a Petri net with interfaces.
- Petri nets suggest the notion of concurrent runs that partially order actions of a run, thus, avoiding them to be mapped onto a global time scale.

2.6 Describing systems on a schematic level

Data, real life items, as well as entire systems must be describable on an abstract, schematic level. In particular, it must be possible to describe just the existence of data, items, functions, etc., without any concrete description of how they look like, how many of them there are, etc. On this schematic level, it should be possible to describe activities in systems, e.g. the principles of executing a clients order of an enterprise. A concrete enterprise is then an instantiation of the schema.

HERAKLIT provides techniques to model such schemata, and to characterize concrete enterprises as instantiations of such a schema. Here, we adapt notions such as structures, signatures and instantiations of signatures, that are well-known from first order logic and algebraic specifications. (Technically, a signature is just a set of sorted symbols for sets, constants, and functions. An instantiation interprets these symbols consistently). We extend signatures by requirements to exclude "unwanted" instantiations, in the spirit of specification languages such as the Z language.

Signatures and their instantiations can naturally be transferred to define Petri net schemata we call them HERAKLIT schemata. Such a schema can be instantiated in different ways; each instantiation results in a concrete Petri net. This concept is useful to model, for example, not just a distinguished business, but a class of businesses that all follow the same business rules. Hence, HERAKLIT strongly supports the idea of reference modelling, a core topic of business informatics [13].

2.7 Verification

The notion of correctness has many implications for big systems. Some ideal properties of a big system can be composed of corresponding properties of the component systems. Not all relevant properties can formally be captured, yet they deserve a proper framework to reason about them. Particularly interesting are methods to prove properties at run-time.

HERAKLIT integrates a number of formal and semi-formal verification techniques to support structured arguments about the correct behaviour of modules.

3 Modules and their composition

3.1 Modules

In Section 1 we discussed a number of principles that are inevitable for modelling big systems: no globally effective structures, associative composition of models of any two systems, composition must be compatible with abstraction, modelling of data and real items, modelling of behaviour, parameterized models. Now we must model systems in such a way that all these principles are met.

We start out with the obvious observation that a real system in general consists of interdependent subsystems. This paves the way for the central notion of Heraklit-modules: A Heraklit module is a model, graphically depicted as a rectangle, with two decisive components:

- Its inner: this may be any kind of graph or text. Three variants are frequent: (a) the inner consists only of the name of the module, (b) it consists of (connected) submodules, (c) it describes dynamic behaviour.
- Its surface: this consists of gates, each gate is labelled, i.e. inscribed by a symbol. The gates of the surface are arranged on the surface of the modules rectangle. Alternatively, each gate is represented as a line, linking the modules rectangle with the gates label.

The following Fig. 2 shows typical Heraklit modules.

3.2 Composition of modules

Composing two modules A and B follows a simple idea: two equally labelled gates of A and B are glued and turned into an inner element of the module $A \bullet B$. However, in this simple version, the composition is fundamentally flawed: Upon composing three or more modules, the order of composition matters: for three modules A, B, and C, the two modules $(A \bullet B) \bullet C$ and $A \bullet (B \bullet C)$ differ from one another. In technical terms: this version of composition is not associative. But associativity is a central requirement, as discussed in Chapter 1.2.

To solve this problem, we return to modules shaped $S = S_1 \bullet \cdots \bullet S_n$. As discussed in Sec. 1.2: each module S_i generally has a left and a right neighbor (S_0 has no left, S_n has no right neighbor). S is composed by composing S_{i-1} with S_i (for $i = 2, \ldots, n$). In the real world, systems frequently exhibit this kind of structure, physically or conceptually.

Therefore, Heraklit partitions the surface of a module L into its left and right interface, written $^*L$ and L^*, resp. To compose two modules L and M, equally labelled gates of L^* and $^*M$ are glued and turn into inner elements of $L \bullet M$. The remaining elements of L^* go to $(L \bullet M)^*$ (together with M^*), and the remaining elements of $^*M$ go to $^*(L \bullet M)$ (together with $^*L$). Most important: A general theorem guarantees that this kind of composition is associative [16].

4 Case study: a service system

4.1 The different modules of the system

Today, many organizations offer a complex service portfolio for their customers or clients [3,4]. Typical examples are banking or financial services, insurance services, legal services, and the medical or health services offered by a hospital or a medical center.

Here, we model the organization of such a service system, serving clients, customers, or patients that want confidential consultation about particular services or a particular treatment, provided by experts.

Fig. 1 shows the signature of the system: there are five sorts of elements in a service system, indicated by C, E, R, A, and S. Their intuitive meaning

basic sorts
C *clients*
E *experts*
R *rooms*
A *admins*
S *services*

constant symbols
EX : *P*(E) *experts at work*
RO : *P*(R) *available rooms*
AD : *P*(A) *admins at work*

function symbol
f: S ⟶ *P*(E) *experts for services*

variables
c: C
e: E
r: R
a: A
s: S

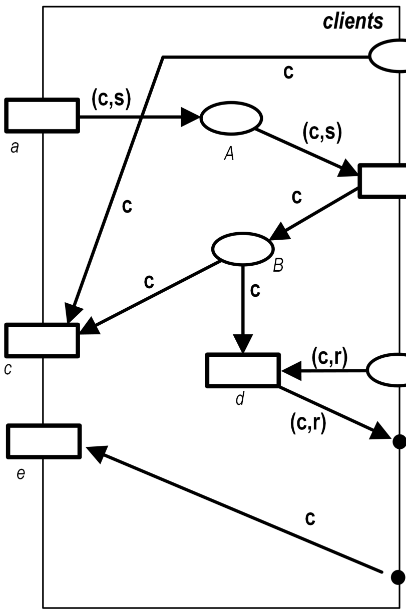
Fig. 1: Signature of the service system

is indicated in italic. In a concrete service system, there are sets of experts, available consulting rooms, and admins, symbolically represented by EX, RO and AD. Their type is $P(E)$, $P(R)$, and $P(A)$, resp., with $P(\cdot)$ standing for "powerset". Furthermore, we need a function symbol f and five variables, one for each basic sort. An *instantiation* assigns each basic sort an arbitrarily chosen concrete set, each constant symbol a set of elements of the indicated sort, and f a function that assigns each service the set of experts that offer consultations for this service.

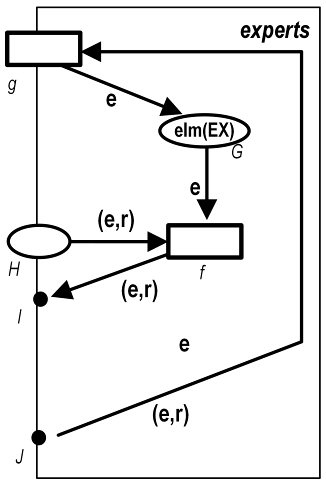
Fig. 2a shows a module that represents the behaviour of clients: For every instantiation of the variables c and s by a client and a service, resp., transition a is enabled. Transition a represents the policy that any client may enter the service system with any kind of wish for consultation for a service s. Hence, place A may eventually hold any number of tokens, with each token consisting of a client and a service. Transition b indicates the service systemss help desk, accepting each clients wishes and asking them to wait at place B. There, a client will eventually receive a message either at place C or at place D. A message at place C indicates that no expert is available; so the client leaves the service system along transition c. A message at place D indicates that the client should proceed to the consulting room named or numbered r. The client will do so along transition d and arrow E. He will later on return along arrow F and leave the service system by transition e.

The module in Fig. 2b represents the behaviour of the service systems experts. There is a set of experts, depicted as EX, fixed when the schema is instantiated, and initially represented as unengaged at place G. One might expect this to be expressed by the symbol EX at place G. However, this would indicate one token at place G. This is not what we want: we want each single expert to be represented as a token. This is achieved by means of the function elm: Applied to a token that represents a set M, $elm(M)$ returns each element of M as a token. For an expert e, the message (e, r) arriving at place H indicates that e must go to consulting room r, due to transition f and arc I. He will eventually return along arc J, release room r, and will be again unengaged at place G.

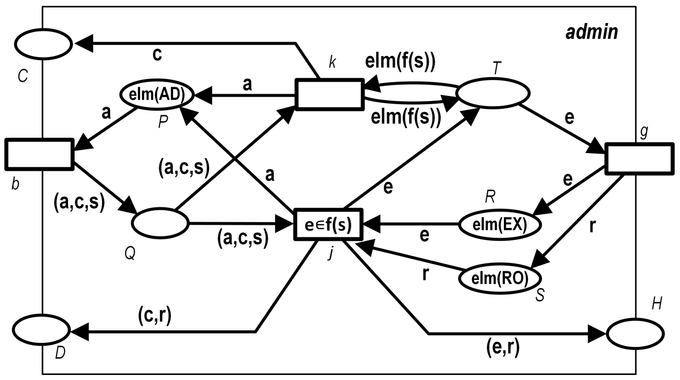
The module in Fig. 2d shows the consulting rooms: A client c and an expert e arriving at room r along the arcs E and I, resp., start their consultation by transition h, end it by i, and leave the room by arcs F and J.



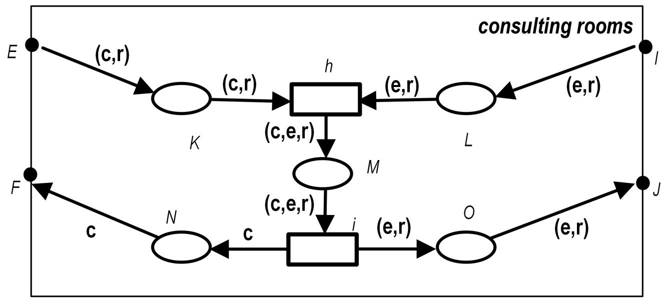
(a) clients



(b) experts



(c) admin



(d) consulting rooms

Fig. 2: The four modules of the system

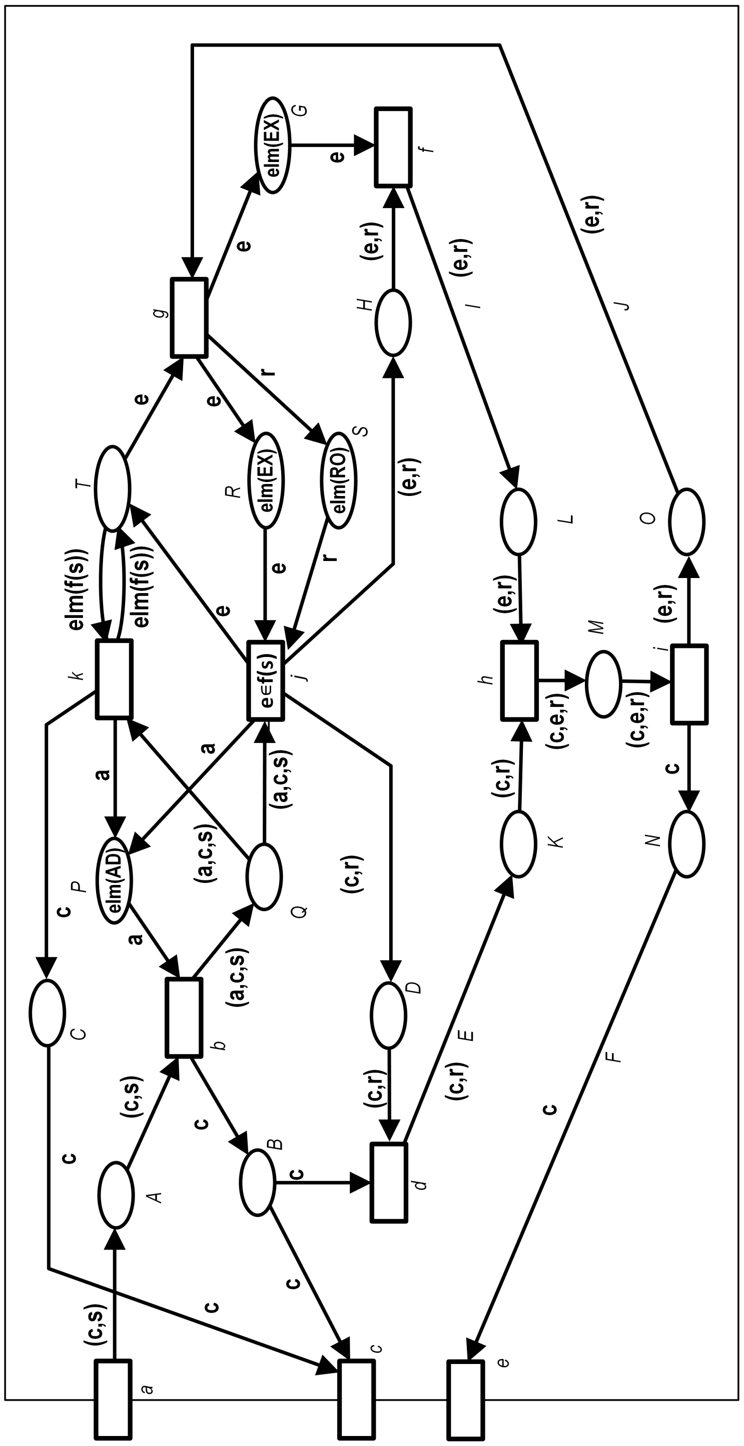


Fig. 3: Overall model of a service system

The behaviour of clients, experts, and the consulting rooms must be properly synchronized. The admin module of Fig. 2c organizes this. Place P initially contains each admin as a token (we employ again the function *elm* as explained above for the experts). An admin a engages with a client c and their request for an expert for service s, along transition b. A token (a, c, s) on place Q then continues either along transition k or transition j. Transition j requires an expert e on place R, such that e offers the service s. The inscription of j indicates this requirement. R always contains a digital twin for each expert that is not engaged with a client. The place S always contains a digital twin of each empty consulting room. Hence, transition j is enabled with proper instantiations of all five variables a, c, s, e, and r. The occurrence of j then renders the admin a available in P for new clients, sends messages to the client c, and the expert e to proceed to room r, and moves the digital twin of e to place T. This way, the digital twin of each expert e is either a token in R or in T. With e in T, the expert e eventually indicates by transition g that they finished their consultation and they release the room r. Finally, transition k manages the case where for a token (a, c, s) no expert for service s is available in R. As discussed above, the digital twin of each such expert is a token in T. Hence, all tokens in the set $f(s)$ of experts for s are in T. This is tested by means of the loop between k and T. Occurrence of k then renders the admin a available in P for new clients and sends a corresponding message to the client c. Notice the subtle treatment of experts and rooms as a scarce resource: If no corresponding expert is available, a client is turned away, as it may take too long until an expert for s is available. But if no room is available, the client is just waiting as long as one room will be available.

4.2 Overall model and abstract composition

Fig. 3 finally glues the four modules into one big module. In Heraklit, this can just be written as: *clients* • *admin* • *consulting rooms* • *experts*.

Similarly, it is possible to construct an abstract composition of the system. Fig. 4 depicts such a composition of the four abstract modules by using the abstraction operator $[\cdot]$, which deletes the inner structure of a module. Formally written as: [*clients*] • [*admin*] • [*consulting rooms*] • [*experts*].

5 Related work

Modelling is typically understood as an interdisciplinary field that is used in many different disciplines as a method or instrument to capture knowledge or to assist other (research) actions [6,2]. As we discuss above, Heraklit mainly does not invent new modeling concepts but integrates proven and well-known modelling approaches. Compared to other integrated approaches which currently dominate the modelling practice, e.g. BPMN, Heraklit provides integrated means to descrive model structure, data, and behaviour. In the central concept of a module, Heraklit combines three proven, intuitively easy to understand,

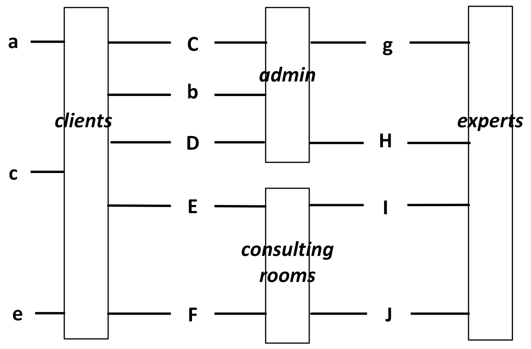


Fig. 4: Abstract composition of the overall model

and mathematically sound concepts that have been used for the specification of systems in the past:

1. Abstract data types and algebraic specifications for the formulation of concrete and abstract data: since the 1970s such specifications have been used, built into specification languages, and often used for (domain-specific) modelling. The book [18] presents systematically the theoretical foundations and some applications of algebraic specifications. Abstract state machines [8] also belong to this context.

2. Petri nets for formulating dynamic behaviour: HERAKLIT uses the central ideas of Petri nets. A step of a system, especially a large system, has locally limited causes and effects. This allows processes to be described without having to use global states and globally effective steps. This concept from the early 1960s [12] was generalized at the beginning of the 1980s with predicate logic and *colored marks* [7,10]. The connection with algebraic specifications is established by [14]. HERAKLIT adds two decisive aspects to this view: uninterpreted constant symbols for sets in places that use the *elm* function to hold instantiations with many possible initial marks, and the *elm* function as an inscription for an arrow to describe flexible mark flow.

3. The composition calculus for structuring large systems: this calculus with its widely applicable associative composition operator is the most recent contribution to the foundations of HERAKLIT. The obvious idea, often discussed in the literature, of modeling composition as a fusion of the interface elements of modules is supplemented by the distinction of left and right interface elements, and composition $A \bullet B$ as a fusion of right interface elements of A with left interface elements of B. According to [16,17], this composition is associative (as opposed to the naive fusion of interface elements); it also has a number of other useful properties. In particular, this composition is compatible with refinement/coarseness and with individual (distributed) runs.

These three theoretical principles harmonize with each other and generate further *best practice* concepts that contribute to a methodical approach to modeling with HERAKLIT, and which will only be touched upon in this paper. On the down-side, industrially mature modelling tools for Heraklit are still under development.

6 Conclusions

The presented case study clearly demonstrates how Heraklit provides an integrated view on structure, data, and locally defined behaviour. Hence, Heraklit covers all central aspects of every computer-integrated system. Such a description can be used for different purposes, e.g. business process management, service engineering, software analysis, design, verification, and development. The used techniques are well-known but combined in a novel and innovative way.

By providing such an integrated method for system specification, Heraklit paves the way for many important innovations which are currently so much in need [2,9]. In particular, we like to introduce the idea of *Systems Mining*. While Data Mining and Process Mining [1] exploit the knowledge implicitly represented in data tuples and event sequences, respectively, Systems Mining is able to analyze the structure, data, and behaviour of a system. For such analysis, Heraklit provides the necessary techniques to specify all essential characteristics of a system. The observed structure of the system can be represented as modules, the observed data is captured by both concrete and abstract data structures, and the observed behaviour is specified as (distributed) runs. Based on such a powerful framework, Systems Mining provide a much richer picture of and deeper insights into big systems.

The presented case study of a service system illustrates powerful possibilities. Based on these Heraklit models, Systems Mining can answer a wide spectrum of interesting questions: (1) Do typical communications patterns between the modules of the system exist? (2) Which services are often requested by customers? (3) Do customers follow particular patterns for requesting services? (4) Which particular service requests and assignments of experts and rooms typically cause long waiting times for a customer? (5) Are there particular behaviour patterns and service requests which typically cause customers to leave the service system without getting a service or treatment?

Such questions and many more can easily be specified with Heraklit. Additionally, Heraklit provides a richer foundation for predictive and prescriptive process management as well as deeper insights for explaining process behaviour [5,11]. Hence, Heraklit lays the foundation for the next step after Process Mining.

References

1. van der Aalst, W.: Process mining. Communications of the ACM **55**(8), 76–83 (2012)
2. Beverungen, D., Buijs, J.C.A.M., Becker, J., Ciccio, C.D., van der Aalst, W.M.P., Bartelheimer, C., vom Brocke, J., Comuzzi, M., Kraume, K., Leopold, H., Matzner, M., Mendling, J., Ogonek, N., Post, T., Resinas, M., Revoredo, K., del Río-Ortega,

A., Rosa, M.L., Santoro, F.M., Solti, A., Song, M., Stein, A., Stierle, M., Wolf, V.: Seven paradoxes of business process management in a hyper-connected world. Business & Information Systems Engineering **Online-First** (2020)

3. Bhmann, T., Leimeister, J., Mslein, K.: Service systems engineering. Business & Information Systems Engineering **6**, 73–79 (2014)
4. Chesbroug, H., Spohrer, J.: A research manifesto for service science. Communications of the ACM **49**(7), 35–39 (2006)
5. Evermann, J., Rehse, J., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129–140 (2017)
6. Frank, U., Strecker, S., Fettke, P., vom Brocke, J., Becker, J., Sinz, E.J.: The research field "modeling business information systems" – current challenges and elements of a future research agenda. Business & Information Systems Engineering **6**(1), 39–43 (2014)
7. Genrich, H.J., Lautenbach, K.: The analysis of distributed systems by means of predicate/transition-nets. In: Kahn, G. (ed.) Semantics of Concurrent Computation, Proceedings of the International Symposium, Evian, France, July 2-4, 1979. Lecture Notes in Computer Science, vol. 70, pp. 123–147. Springer (1979)
8. Gurevich, Y.: Evolving algebras: the lipari guide. In: Borger, E. (ed.) Specification and Validation Methods. pp. 9–36. Oxford University (2012)
9. Houy, C., Fettke, P., Loos, P., van der Aalst, W.M.P., Krogstie, J.: Bpm-in-the-large – towards a higher level of abstraction in business process management. In: Janssen, M., Lamersdorf, W., Pries-Heje, J., Rosemann, M. (eds.) E-Government, E-Services and Global Processes – Joint IFIP TC 8 and TC 6 International Conferences, EGES 2010 and GISP 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings. IFIP Advances in Information and Communication Technology, vol. 334, pp. 233–244. Springer (2010)
10. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use. Springer (1982)
11. Mehdiyev, N., Fettke, P.: Prescriptive process analytics with deep learning and explainable artificial intelligence. In: Rowe, F., Amrani, R.E., Limayem, M., Newell, S., Pouloudi, N., van Heck, E., Quammah, A.E. (eds.) 28th European Conference on Information Systems – Liberty, Equality, and Fraternity in a Digitizing World, ECIS 2020, Marrakech, Morocco, June 15-17 (2020)
12. Petri, C.A.: Kommunikation mit Automaten. Ph.D. thesis, Institut für instrumentelle Mathematik der Universität Bonn (1962)
13. Rehse, J., Fettke, P.: A procedure model for situational reference model mining. Enterprise Modelling & Information Systems Architecture – International Journal of Conceptual Modeling **14**, 3:1–3:42 (2018)
14. Reisig, W.: Petri nets and algebraic specifications. Theoretical Computer Science **80**, 1–34 (1991)
15. Reisig, W.: Understanding Petri Nets. Springer (2013)
16. Reisig, W.: Associative composition of components with double-sided interfaces. Acta Informatica **56**(3), 229–253 (2019)
17. Reisig, W.: Composition of component models – a key to construct big systems (2020)
18. Sanella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Springer (2012)