The Internet of Actors - A Peer-to-Peer Agile Value Creation Network

Florian Strecker
florian.strecker@actnconnect.com
actnconnect
Nuremberg, Germany

Reinhard Gniza
reinhard.gniza@actnconnect.com
actnconnect
Nuremberg, Germany

Contents

1 INTRODUCTION

> Network commerce has consequences that go far beyond just a business model. [...] Markets are based on mistrust, networks on trust. Markets are based on the pursuit of self-interest, networks on shared interest. Markets are arm's-length transactions, networks are intimate relationships. Markets are competitive, networks are cooperative.
>
> *Jeremy Rifkin* ([18], p. 192 f)

The intra- and inter-company work models change from classic, pre-defined, inflexible business processes to agile value creation networks. Human, machines and software collaborate in an integrated and coordinated way to fullfill their objectives. This necessitates a new dimension of agility, speed of transformation and individuality. We strongly believe that we need an open network of self-coordinating, modular components that offer fully-fledged interoperability to tackle these challenges on the way to autonomous software systems. At the same time such a network will offer an economic alternative to the classic, centralized platform models and standards of the current IT-industry. These, e.g. BPMN, "fail to guarantee that standard-conforming business process models are interoperable (platform-independent)" ([4]).
Some scientists are already aware of these challenges and laid out fundamental groundwork. Christian Stary proposes "a System-of-Systems specification as a network of cooperating behavior entities" ([19]). These and similar ideas converge on the creation of a new era in systems design.
This whitepaper presents a methodology, a modular concept and the foundation of the *Internet of Actors* (*IoA*). The IoA will reduce complexity, programming efforts and unclear interfaces while creating an *Interoperability Network*. It is going to be an enabler for agile, decentralized, self-coordinating value creation networks, acting as catalyst for the Internet of Things, Digital Transformation and Industry 4.0.

2 SMART ACTORS

To create such an agile value creation network we first need to focus on the atomic components (= the nodes and their respective relations) the network is comprised of. *Smart Actors* (*SMA*) are the basic building blocks of the IoA. As the IoA is designed as a network, the network nodes need the respective capabilities to deal with challenges like inter-company collaboration in a (self-)coordinated, modular encapsulated manner (mentioned in 1 Introduction).

2.1 Requirements

The main requirements to the network nodes are similar to the requirements that can be observed with the architectural pattern of microservices (based on [11]), as microservices can also be interpreted as dedicated nodes in network-like applications:

(MS1) determine the functional scope based on a single business capability that is decomposed on its lowest level for the domain applicable
(MS2) consistent encapsulation leading to very clear interfaces to other network nodes (e.g. according to the Law of Demeter [25]: each unit should have limited knowledge about other units)
(MS3) "smartness" ("smart endpoints, dumb pipes" [11]) in a sense of being able to do everything needed to add value with its actions without doing "too much" (YAGNI-dilemma [31])
(MS4) independently releasable without being in the need of other components thus ensuring independent lifecycles of the different nodes
(MS5) self-organizing - in regards to building a network, the nodes must be able to connect to each other in a meaningful way thus ensuring added value in terms of Aristoteles' synergy-principle (the whole is greater than the sum of its parts [27]) and tackling the issue of messy service connections ([11]) by using a clear (and yet flexible) communication structure between different components
(MS6) asynchronous connections between single network nodes in order to ensure resilience to temporarily breakdowns (due to infrastructure, scalability or timing issues in parts of the network)

To meet these requirements, we propose the creation of *Smart Actor*s as nodes and therefore as core building blocks of the peer-to-peer agile value creation network.
Therefore, a *Smart Actor* is able to:

(R1) fullfill a well-defined task
(R2) "know" the execution-logic needed to fulfill this task, including decision-rules, different paths etc.
(R3) gather and hold all data needed for the task
(R4) offering a means to extract the essence of the task in a machine-readable way (as an enabler for self-organization)
(R5) connect to arbitrary software-interfaces or physical machines in order to exchange data
(R6) present its data to human users and offer possibilities to change these data or take decisions where defined by the *SMA* (= "frontend"/ UI [30])
(R7) communicate with other *SMAs* via well-defined protocols to gather information and/ or to propagate its output(s)
(R8) has its own release cycle (= being indepently releasable) without being directly affected by the release cycles of others
(R9) can be executed in a software system by interpreting or compiling its definition/ source code

The foundation of our *Smart Actor*s can be found in software design concepts reaching back to the 1970's.

2.2 The Foundation of Smart Actors: PASS

In 1994, Albert Fleischmann proposed a concept called "SAPP/ PASS" (Structured Analysis of Parallel Programs in conjunction with Parallel Activities Specification Scheme, [8], pp. 204 ff). This

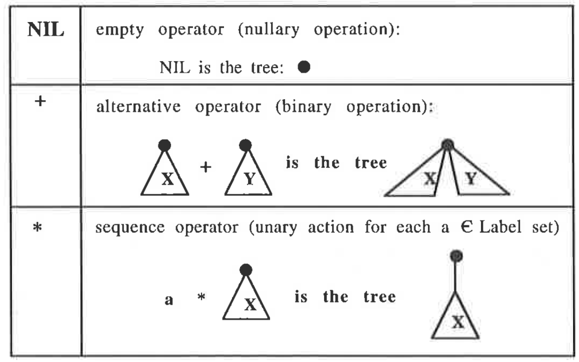

Figure 1: Basic Operations of CCS ([8], p. 130)

Associativity:	(X+Y)+Z = X+(Y+Z)
Commutativity:	X+Y = Y+X
Nullity:	X+NIL = X

Figure 2: Basic Laws in CCS ([8], p. 131)

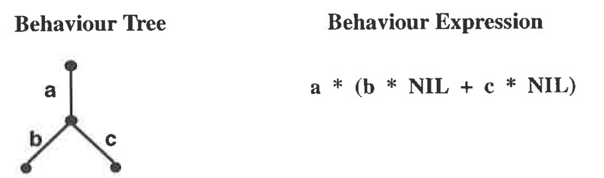

Figure 3: Example of Behaviour Expressions and Trees ([8], p. 131)

scheme is based on Milner's and Hoare's Calculus of Communicating Systems (CCS, [15]) which has later been developed into the Pi-calculus ([16]).

2.2.1 Calculus of Communicating Systems (CCS). Fleischmann describes the CCS as "a theoretically very important technique for describing the behaviour of processes[1] explicitly" ([8], p. 130). Paraphrasing [8] (pp. 130-134), the CCS offers the following: The behaviour of a process[2] is described as a rooted, unordered, finite branching tree. The initial state of the process is represented by its root. Branches are labelled and represent actions or transitions to a next state. The CCS distinguishes observable and unobservable actions[3]. The trees offer elementary algebraic operations (see figure 1). These operations obey basic algebraic laws (associativity, commutativity and nullity, see figure 2). The trees are called behaviour trees (describing processes). Expressions represent these trees (see figure 3): After step a has been executed, steps b (x)or c can be executed next. It is also possible to model agents[4] with infinite behaviour in CCS. Figure 4 illustrates agents X and Y, both with

[1]In our case, a "process" will become a *Smart Actor*
[2]= its algorithm
[3]observable or unobservable from outside of the process
[4]in our terms later: *Smart Actors*

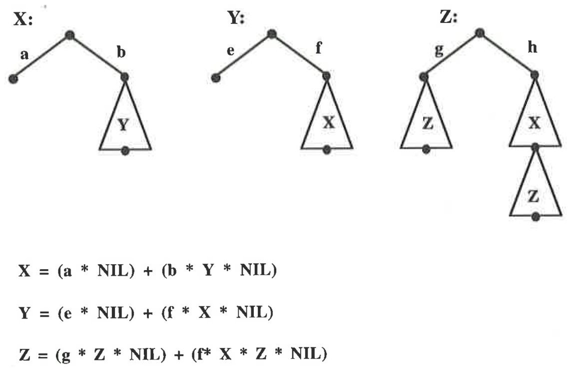

Figure 4: Infinite Process Behaviour ([8], p. 132)

potentially finite behaviour[5], and agent Z with a definite infinite behaviour. Albert Fleischmann also focuses on the fact that for each observable action there exists a complementary (or: inverse) action in CCS. It is possible to link multiple agents together. If there exist complementary actions in the different behaviour trees, these agents can communicate in synchronous message exchanges[6] (see [8], p. 132).
In general, CCS offers a graph-style[7] approach to model complex behaviours[8] and communication between these.

2.2.2 Parallel Activities Specification Scheme (PASS). Based on the CCS[9] Albert Fleischmann developed the Parallel Activities Specification Scheme (PASS) in conjunction with a method for Structured Analysis of Parallel Programs (SAPP).
SAPP is a means to decompose complex systems into small, easy-to-handle components. These components are able to be executed in parallel and to call each other via clearly defined message channels ("message-types" [8], p. 205). Figure 5 exemplifies such a SAPP specification, where T can send messages of type $M3$ to P and P can send type $M2$ to T[10]. This type of diagram represents possible communication channels, but has no time, sequence or frequency restrictions.
PASS is used to define a single "process" ([8])[11]. A PASS description contains information about synchronization of messages ("input pool"[12]), a graph representing the process behaviour (= algorithm/ "intelligence" of the *SMA*) and process-refinements handling data operations.
The SAPP/ PASS concepts have also been adopted by parts of the

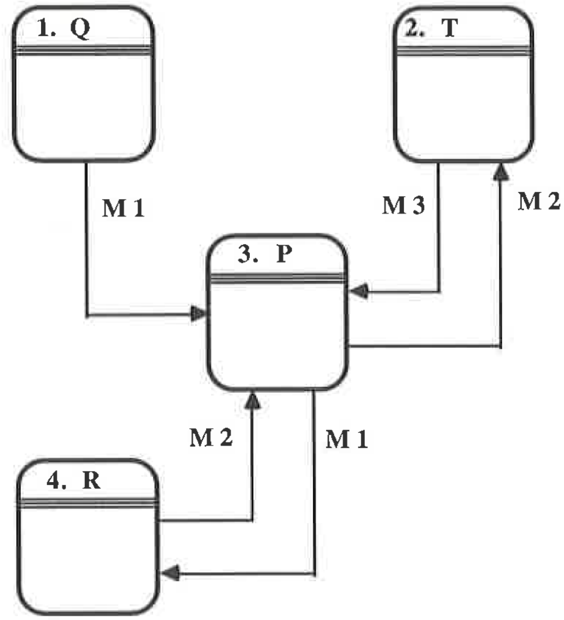

Figure 5: SAPP Specification ([8], p. 207)

Business Process Management (BPM) community ([9][13]). In contrast to classic BPM methodologies, PASS is able to focus on actors and/ or agents involved in business processes. These can be humans, software components or machines in mixed setups.
One reason for this adoption by the BPM community, similar to the advantages/ main features of software system development using PASS, is its Turing Completeness ([28]). This has been proven by an interpreter model based on Abstract State Machines (ASM, [12] and [6]), which are based on Finite State Machines. As PASS graphs themselves represent state machines, they can be interpreted like program source code. Therefore it is possible to execute a PASS model instantly without being forced to transform or manually code it. Egon Börger created such an interpreter "for both simulation (testing) and verification (logical analysis of properties of interest) of classes of S-BPM[14] processes" ([5]).
The original SAPP/ PASS methodologies unfortunately define that "in the PASS model each system consists of a fixed number of processes[15] and each process has a unique name" ([8], p. 205). But to achieve independent releasability it is paramount that *SMAs* do not have fixed communication channels to other actors and allow agile reconfiguration of the communication network they are part of.

2.3 Independent Releasability of Smart Actors

To tackle the requirement of independently releasable components, we created a concept of well-defined but loose-bound communication channels.
This concept is based on CCS, ASM and PASS. Even if PASS in

[5] if transition $X(a)$ or $Y(e)$ is executed
[6] as opposed to asynchronous message exchange, which we will focus on later
[7] corresponds to tree-style
[8] matches in this case the term "algorithms"
[9] and other methods, [8], p. 201
[10] for completeness of the example: Q can send $M1$ to P, P can send $M1$ to R, R can send $M2$ to P
[11] = $Smart\ Actor$ in our terms
[12] Input pools are one of the most important concepts of PASS as they enable messages to be sent either synchronous or asynchronous between components and even offer possibilities to restrict the number of messages being received at a certain time etc. Without input pools, especially the crucial asynchronous communication would not work. We do not explain the details of this concept, as they are not needed for grasping the general communication of $SMAs$.

[13] English, enhanced version of the original (German) book: [10]
[14] S-BPM corresponds to PASS in our context
[15] again: process maps to $Smart\ Actor$ in our context

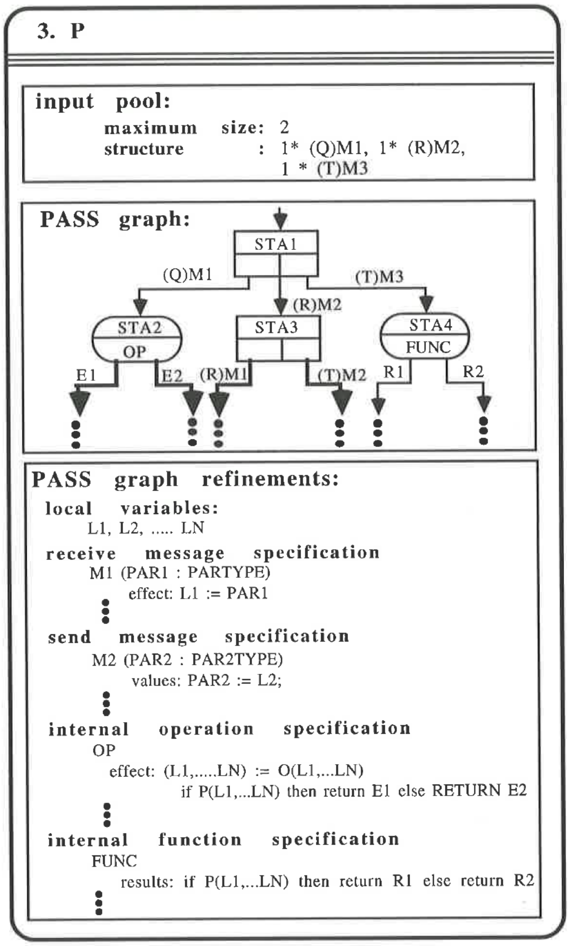

Figure 6: PASS Specification ([8], p. 208)

conjunction with SAPP basically tries to model and execute complete, enclosed systems[16], PASS graphs can be used to discern one or more observable behaviours[17]. We use a PASS graph as constituting component of a *Smart Actor*. An observable behaviour is basically a reduced version of the PASS graph, focusing on the communication[18] that can potentially happen with one specific communication partner (= another PASS graph). This can be named "role-based behaviour interface" (*RBI*), as every communication partner acts in a specific role in its relation to the PASS graph in focus. It follows that a *RBI* can be created for every potential communication partner of a PASS graph.

Figure 7 shows a graph X that depicts communication with 2 different partners (partner $'$ (A) and partner $''$ (B))[19]. Therefore 2 *RBI*s can be extracted from this graph. The *RBI* from X to A, $RBI(X, A)$

[16] with asynchronous communication within the system

[17] see also CCS's observable actions, section 2.2.1. PASS graphs can also be transformed to CCS behaviour expressions, which aren't as powerful as some PASS features ([8], p. 259 ff), but are therefore suitable to "observe" the communication behaviour

[18] sending messages or receiving messages

[19] We intentionally left out information about the types of the graph's states (*send*, *receive*, *do* (internal)) to focus on the *RBI* explanation.

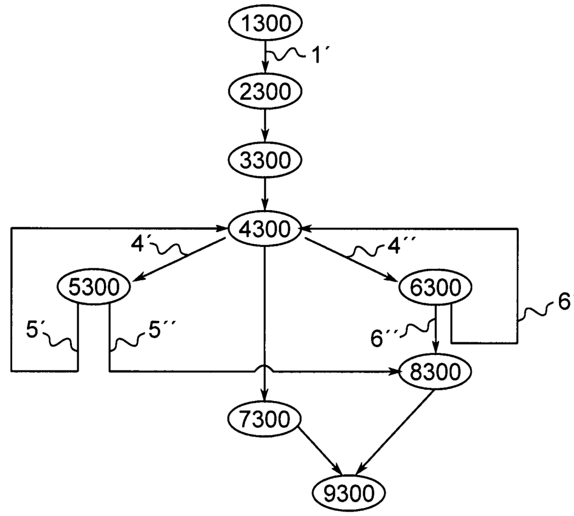

Figure 7: Example of a graph to discern a *RBI* ([20])

consists of all possible message flows between X and A, their direction and their structural content[20]. To the relation between X and B applies the same, there is an $RBI(X, B)$.

To every $RBI(graph_x, graph_y)$ (1) there exists an inverse $RBI'(graph_x, graph_y)$ (2).

$$RBI(graph_x, graph_y) \quad \text{with } graph_x = \text{ original Smart Actor} \quad \text{and } graph_y = \text{ potential Communication Partner} \tag{1}$$

$$RBI'(graph_x, graph_y) = RBI(graph_z, graph_x) \quad \text{with } graph_x, graph_y, graph_z \in SmartActors \tag{2}$$

This inverse RBI' is a tree performing the same message flows like *RBI*, but does a *send* where in *RBI* there is a *receive* and a *receive* where there is a *send*. It is possible to find any *SMA Z* that has one of its *RBI* that corresponds to RBI' (3).

$$RBI(Z, Z') = RBI'(X, Z) \tag{3}$$

This means that these PASS graph representations of *SMAs* are able to communicate with each other; in this case, they can be connected (4 - 6):

$$\Rightarrow Z' = X \tag{4}$$

$$\Rightarrow RBI(Z, X) = RBI'(X, Z) \tag{5}$$

$$\Rightarrow RBI(X, Z) = RBI'(Z, X) \tag{6}$$

If there are a lot of *Smart Actor*s known[21], there may be more than one *Smart Actor* that can be found as a potential communication partner. In this case, rules on the communication context

[20] by structural content we mean data structures (and semantics) transported by a message

[21] or are being able to be discovered via address books, Smart Actor Name Systems (SMANS) etc.

could have been set within the *SMAs* themselves, further restricting the possible communication partners. Selection strategies (like "*bestRated*", "*nearest*", "*firstComeFirstServe*", ...) can apply additionally.

This approach shows that a *Smart Actor* does not have to be connected with its communication partners before deployment for execution, as the *RBI*s make it possible to find suitable communication partners even at execution time (see also [20]). This represents also a difference to Agha's approach ([2]) to software actors, as we do not need direct addressing of other actors, but can use rule based addressing (RBA) via RBIs.

2.4 Definition of Smart Actors

Keeping the requirements (see 2.1), the foundational concepts CCS (see 2.2.1), ASM, PASS (see 2.2.2) and our thoughts on independent releasability (see 2.3) in mind, we can design a *Smart Actor* as follows:

(SMA1) The pivotal part is the definition of an internal behaviour graph[22]. The behaviour graph consists of states with the 3 basic operations (*send*, *receive*, *do*), transitions between these states and references to the *SMA*'s data (see (SMA2)). The behaviour represents the smartness (or: algorithm), helping the *Smart Actor* to
  - (SMAB1) fullfill its task (requirement (R1)),
  - (SMAB2) know the execution logic needed (requirement (R2)),
  - (SMAB3) represent decision- and other rules as these are mapped to behaviour and communication (requirements (R2) and (R7))
  - (SMAB4) extract the *RBI*s for defining the communication interfaces to other *SMAs* (requirement (R7)) - thus creating interoperable building blocks

(SMA2) Additionally, a complete data structure can be defined for every *Smart Actor*. This data structure contains everything the *SMA* needs within its behaviour (requirement (R3)).

(SMA3) If the *Smart Actor* is a *Service Actor* or *Physical Actor*, meaning it connects to microservices or arbitrary software components or software interfaces, it needs a *Service Mapping* for each of its *do*-states, defining calls and data exchange to and from these systems (requirement (R5)).

(SMA4) In case the *Smart Actor* is intended to support a human in executing a task[23], user interfaces (UI) can be defined by mapping to the *do*-states[24] and data structures available, thus creating executable views on the *Smart Actor* (requirement (R6)).

(SMA5) Metadata in a structured format are intended to offer machine- and human-readable data about the *SMAs* essence, intentions (see [22]), creators, costs and other useful information (requirement (R4)).

(SMA6) As the extracted *RBI*s define communication interfaces instead of a tight coupling to other components, a *SMA*'s releasability/ deployability is completely independent from others. Therefore, a *Smart Actor* represents a self-contained component (requirement (R8)).

(SMA7) Due to its ASM-based, Turing-complete nature, a *SMA* can be executed by interpreting its behaviour graph (requirement (R9)).

In summary, these *Smart Actors* fulfill all requirements listed in section 2.1 Requirements. Therefore, *SMA*s can also be used to execute *SmartContracts* ([26] and [32] p. 37 f) or act as *DApp*s ([23] and [32] p. 39 f). A possible XML data format for describing *SMAs* is proposed in [21].

3 A PEER-TO-PEER AGILE VALUE CREATION NETWORK

The concept of *Smart Actor*s allows the creation of a *peer-to-peer agile value creation network* which we call *The Internet of Actors* (*IoA*). We want this network to facilitate a clean architecture and stick to an important principle (known from bitcoin [17], p. 8): "The network is robust in its unstructured simplicity. Nodes work all at once with little coordination", whereas 'coordination' is equivalent to 'no centralized orchestration'.

3.1 Definition of the Peer-to-Peer Agile Value Creation Network

The *SMAs* represent the main building blocks and are the nodes within the network. The characteristics of this network are:

Peer-to-Peer The *SMAs* can be executed in decentralized environments. If a *Smart Actor* has the means to find communication partners and message with them (see 3.3), the whole network can be built up in a distributed manner without having a centralized structure.

Agile *Smart Actor*s do not employ a tight coupling, but they are entirely uncoupled and independent from other *SMAs*. That allows the (self-sustaining) configuration of the network to change at any time with the addition, the removal or the update of new or existing *Smart Actor*s (see 3.3). Each network node is part of one or more choreographies[25] that emerge from the communication protocols (= *RBI*) and rules set by any *SMA*.

Value Creation Objective of every *SMA* is to create a value (i.e. in terms of a capability), by transforming any inputs to any outputs[26]. Thus, in the network as a whole, there will be a multitude of focal areas (= choreographies) of value creation. Due to the synergy-principle ([27]) the total of value creation will be higher than the combined value-deltas of every single *SMA* involved.

Network We talk of one single network instead of multiple networks, as - due to its decentralized character - all *Smart Actor*s can contribute and will be part of the global network of collaboration and (therefore) interoperability.

Similar to most blockchain-based peer-to-peer technologies, this network shall offer a standardized, open access for everyone who wants to contribute to the network, e.g. by creating new *Smart*

[22] A *Smart Actor* can have more than one behaviour graph for specific applications like *Message Guards*" (see [10], p. 120 ff), *Behavior Macros* (see [10], p. 112 ff) etc.

[23] we call that a *Business Actor*

[24] there are also special UI's possible for *receive*

[25] for definitions of the term "choreography" see [14]

[26] in- and outputs of *SMAs* are always messages within their respective communication patterns

*Actor*s. To facilitate this open access it is paramount to create two areas of standardization.

3.2 Internet of Actors Notation (IoAN)

One area of standardization is the *Internet of Actors Notation* (*IoAN*) which will cover

(IoAN1) the complete definition of *SMAs* (see 2.4 Definition of Smart Actors),
(IoAN2) the notation of the *RBI*s,
(IoAN3) means to show and visualize choreographies of *SMAs*.

As mentioned before, first concepts regarding (IoAN1) have already been proposed ([21]) and implemented. Furthermore there is a standardization group working on an OWL-definition of an exchange format ([7]).

3.3 Smart Actor Operating System (SMAOS)

The second area of standardization is a *Smart Actor Operating System* (*SMAOS*). As discussed before (see 2.4 Definition of Smart Actors), *Smart Actor*s can be executed without any further preparations. For this execution, an operating system is needed, that serves three main purposes. The SMAOS

(SMAOS1) can interpret and therefore execute *SMAs*,
(SMAOS2) facilitates communication between *Smart Actor*s on the same SMAOS instance and across different SMAOS instances, and
(SMAOS3) enables the discovery of other *SMAs* as potential communication partners.

With actnconnect's *Actorsphere* ([1]) there exists a first implementation of a SMAOS.
To address the issue of messaging between *Smart Actor*s (see (SMAOS2)) in regards to transportation layers and formats, we propose the definition of a *Smart Actor Communication Protocol* (*SMACP*). The *SMACP* could also comprise possibilities to use blockchain technologies (like Hyperledger Fabric ([13] and [32] p. 40 f) or dedicated blockchains) for facilitating distributed storage of messages or even *SMA* system states.
The discovery of *SMAs* and/ or SMAOS instances anywhere in the IoA (see (SMAOS3)) requires a *Smart Actor Name System* (*SMANS*). Techniques like the Satoshi Client Node Discovery ([3]), the Domain Name System ([24]), UDDI ([29]) or similar might be used as an inspiration to define such *SMANS*.
Putting all these definitions together, the components of the IoA will be well-defined, executable, communicating and the network can be accessed openly without any discrimination or censorship.

4 CONCLUSION

The concept of *Smart Actors* presents a method and implementation of the central building blocks of a peer-to-peer agile value creation network.
From a methodological point of view the *SMAs* offer all basic capabilities to build such a network due to their clear and open communication interfaces (*RBI*s). They enable communication between different network nodes regardless of their nature. Therefore the *Internet of Actors* can connect humans, machines and software to facilitate their individual contributions to shared and individual objectives.

In a technical sense the *Smart Actor*s are not just "structured" microservices but represent the base components for a new internet, the IoA. Even the very foundations of this network, administrative tools, can be described and implemented by *SMAs*. The IoA is therefore self-sustaining and self-extending.
Individual value contributions[27] can be priced and settled by means of the IoA itself in connection with blockchain-based techniques.
Our further efforts need to go into

(ToDo1) the standardization of IoAN and SMAOS,
(ToDo2) the creation and facilitation of a growing community of (value) contributors for the IoA,
(ToDo3) the detailing of business models within the IoA and
(ToDo4) the creation of application examples for the building blocks and the value creation network itself.

The IoA offers a serious economic alternative to the centralized platform models available today.
The IoA comprises ideal conditions to "go far beyond just a business model", but to build cooperative relationships based on trust to pursue shared interests ([18], p. 192 f).

ACKNOWLEDGEMENTS

We want to thank Christian Stary, Albert Fleischmann, Felix Gniza, Anton Friedl, Werner Schmidt, Matthias Lederer and the members of the I2PM community for the fruitful discussions and their support.

[27] remember: outputs will be sent in form of messages = communication

REFERENCES

[1] actnconnect. *Actorsphere*. URL: http://actnconnect.de/actorsphere_en.

[2] G. A. Agha. *Actors: a model of concurrent computation in distributed systems*. MIT Press Cambridge, 1986. ISBN: 0-262-01092-5.

[3] bitcoin.it. *Satoshi Client Node Discovery*. URL: https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery.

[4] E. Börger. Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL. *Software & Systems Modeling*, 11(3):305–318, Sept. 2011. DOI: 10.1007/s10270-011-0214-z.

[5] E. Börger. A subject-oriented interpreter model for s-bpm. *Subject-Oriented Business Process Management*:315–363, 2012. A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Börger, editors. DOI: 10.1007/978-3-642-32392-8.

[6] E. Börger and R. Stärk. *Abstract State Machines*. Springer Berlin Heidelberg, 2003. DOI: 10.1007/978-3-642-18216-7.

[7] M. Elstermann and F. Krenn. The semantic exchange standard for subject-oriented process models. In *Proceedings of the 10th International Conference on Subject-Oriented Business Process Management - S-BPM One '18*. ACM Press, 2018. DOI: 10.1145/3178248.3178257.

[8] A. Fleischmann. *Distributed Systems*. Springer Berlin Heidelberg, 1994. DOI: 10.1007/978-3-642-78612-9.

[9] A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Börger. *Subjektorientiertes Prozessmanagement*. Carl Hanser Verlag GmbH & Co. KG, July 2011. DOI: 10.3139/9783446429697.

[10] A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Börger. *Subject-Oriented Business Process Management*. Springer, 2012. DOI: 10.1007/978-3-642-32392-8.

[11] M. Fowler and J. Lewis. Microservices - a definition of this new architectural term. Technical report, ThoughtWorks, 2014. URL: https://www.martinfowler.com/articles/microservices.html.

[12] Y. Gurevich. Evolving algebras 1993: lipari guide. *Specification and Validation Methods*:9–36, 1995. E. Börger, editor. URL: https://web.eecs.umich.edu/~gurevich/Opera/103.pdf.

[13] Hyperledger. *Hyperledger Fabric*. URL: https://www.hyperledger.org/projects/fabric.

[14] K. Kolo. Ode to choreography. *Organizational Aesthetics*, 5(1):37–46, 2016. URL: https://digitalcommons.wpi.edu/oa/vol5/iss1/3.

[15] R. Milner, editor. *A Calculus of Communicating Systems*. Springer Berlin Heidelberg, 1980. DOI: 10.1007/3-540-10235-3.

[16] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(02):119, June 1992. DOI: 10.1017/s0960129500001407.

[17] S. Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. URL: https://bitcoin.org/bitcoin.pdf.

[18] J. Rifkin. *The European Dream: How Europe's Vision of the Future Is Quietly Eclipsing the American Dream*. Jeremy P. Tarcher/ Penguin, 2004.

[19] C. Stary. System-of-systems design thinking on behavior. EN. *Systems*, 5, Jan. 2017. DOI: 10.3390/systems5010003.

[20] F. Strecker and A. Fleischmann. Vorrichtungen, verfahren und computerprogramme zum erkennen koppelbarer schnittstellen. German. German patent request (DE 10 2015 107 150 A1). 2016. URL: https://register.dpma.de/DPMAregister/pat/PatSchrifteneinsicht?docId=DE102015107150A1&page=1&dpi=300&lang=de&full=true.

[21] F. Strecker, R. Gniza, T. Hollosy, and F. Schmatzer. Business-actors as base components of complex and distributed software applications. In J. L. Sanz, editor, *S-BPM '16 Proceedings of the 8th International Conference on Subject-oriented Business Process Management*. ACM, 2016. DOI: 10.1145/2882879.2882887.

[22] J. D. Velleman and M. E. Bratman. Intention, plans, and practical reason. *The Philosophical Review*, 100(2):277, Apr. 1991. DOI: 10.2307/2185304.

[23] Wikipedia. *Decentralized Application*. URL: https://en.wikipedia.org/wiki/Decentralized_application.

[24] Wikipedia. *Domain Name System*. URL: https://en.wikipedia.org/wiki/Domain_Name_System.

[25] Wikipedia. *Law of Demeter*. URL: https://en.wikipedia.org/wiki/Law_of_Demeter.

[26] Wikipedia. *Smart Contract*. URL: https://en.wikipedia.org/wiki/Smart_contract.

[27] Wikipedia. *Synergy*. URL: https://en.wikipedia.org/wiki/Synergy.

[28] Wikipedia. *Turing Completeness*. URL: https://en.wikipedia.org/wiki/Turing_completeness.

[29] Wikipedia. *UDDI*. URL: https://en.wikipedia.org/wiki/Web_Services_Discovery#Universal_Description_Discovery_and_Integration.

[30] Wikipedia. *User Interface*. URL: https://en.wikipedia.org/wiki/User_interface.

[31] Wikipedia. *YAGNI*. URL: https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it.

[32] X. Xu, I. Weber, and M. Staples. *Architecture for Blockchain Applications*. Springer International Publishing, 2019. DOI: 10.1007/978-3-030-03035-3.

ACRONYMS

ASM Abstract State Machines

BPM Business Process Management

CCS Calculus of Communicating Systems

DApp De-centralized App
DNS Domain Name System

IoA Internet of Actors
IoAN Internet of Actors Notation
IoT Internet of Things

OWL Ontology Web Language

PASS Parallel Activities Specification Scheme

RBA Rule Based Addressing
RBI Role-based Behaviour Interface

SAPP Structured Analysis of Parallel Programs
SMACP Smart Actor Communication Protocol
SMANS Smart Actor Name System
SMAOS Smart Actor Operating System

UDDI Universal Description, Discovery, and Integration

XML eXtensible Markup Language