Contextual App'ification

Christian Stary
University of Linz
Business Informatics –
Communications Engineering
Christian.Stary@jku.at

ABSTRACT
In this contribution, user-centered design and the generation of human support systems based on apps as behavior encapsulations are addressed from a conceptual and engineering perspective. Specific ingredients, in particular Complex Adaptive Systems (CAS) theory, a System-of-Systems (SoS) perspective, semantic interoperability of apps, and Subject-oriented Business Process Management (S-BPM) are detailed with respect to user-centered development. They allow the contextual alignment of application behavior from a role perspective, based on communication and interaction models that can be executed automatically. Users can set up the design space, validate specifications, and evaluate the context-sensitive alignment of apps. They primarily detail communication flows to meet their requirements and can experience the aligned apps interactively.

Author Keywords
System(ic) Thinking, Complex Adaptive Systems; System-of-Systems; Subject orientation; User-centered Design; App'ification; Choreography.

ACM Classification Keywords
D.2.10 Design: Methodologies; H.4.m. Information Systems Applications: Miscellaneous; H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Human Factors; Design.

INTRODUCTION

The design and development of interactive systems has changed significantly in the last few years, due to technological developments and resulting usage of applications (cf. [1][2][3][4]). Development has been driven by both, smart mobile devices, and dynamically adaptable application architectures. Using apps on mobile devices as intermediaries for accessing (cloud) services as standard interaction scenario leads to an app'ification of these services with a focus on user-specific needs. The term 'app'ification' or 'app-fication', respectively, has been used in several contexts, such as

- Personalization of (interactive) services [5]
- Security and privacy issues concerning interactive platforms (cf.[6])
- Speeding up of technology development and novel segmentation of system landscapes [7]
- Facilitating of third party developments for IoT (Internet of Things) platforms [8]
- Sustainability in a digital world and development of humanity when replacing human conversations, interactions and decisions through apps [9]

Gidey et al. [10] have been looking for architecturally significant requirements and architectural design decisions addressing several of the above mentioned issues. In their systematic elicitation study based on grounded theory they could identify 'appification' as *'the development and provision of smaller applications in a single feature set per application to work on devices with varying sets of platforms and form factors'* ([10], p.145). Following this understanding, a user-centered app'ification design approach requires dynamic composition and alignment facilities to adjust small applications to situation-sensitive needs and technological capabilities available in a specific situation.

As the number of apps a user utilizes increases the importance of contextual design involving various apps increases [8]. Consider a train information app. It can be of interest in the context of both, planning a trip, and being delayed on an ongoing trip. The latter requires real-time travel assistance in contrast to the first case which is based on planned schedules. From a user perspective, any difference in context should lead to different technical system behavior. Situation- or task-sensitive support requires dynamic design variants, albeit the need for consistent user experience of particular apps across multiple settings. Being able to adapt services in a context-dependent manner results in end-user computing, providing the capability to arrange apps according to an individual flow of control and dynamically changing requirements. Unlike the device-centric service computing the focus hereby is on a context-centric service experience caused by an app'ified design space. In such a space users should be able to design their digital services in a seamless manner.

Several challenges of contextual app'ification can be identified, when introducing a user-centered design approach, among them dynamic systems' thinking and generating diagrammatic representations that can be executed under user control. Dynamic systems' thinking is based on understanding interactive systems as Complex Adaptive Systems which is introduced in the next section. It allows considering app'ification as System-of-Systems design process. Its structure is revealed in this section. System-of-Systems thinking gives space to create semantically interoperable designs which are populated with apps, and thus, triggers a respective development process (i.e. app'ification).

Semantic interoperability as essential part of app'ification is discussed in the subsequent section. This part of the paper also introduces process prototyping based on subject-oriented design representations, as it features task-specific user experience. Subject-oriented modeling allows mapping app-specific and cross-app behavior to executable system specifications, which in turn enables prototyping of app'ified support systems. The approach is exemplified for meeting specific needs from a user perspective. The final section concludes the contribution, summarizing the objectives and achievements, and providing inputs to further research.

TAKING A SYSTEM-OF-SYSTEMS PERSPECTIVE

In order to capture the dynamics of evolving socio-technical systems based on user needs, in this section relevant perspectives and concepts, namely Complex Adaptive Systems and System-of-Systems thinking, are provided.

Complex Adaptive Systems

According to Chan [11] Complex Adaptive Systems (CAS) started in US to oppose the European "natural science" tradition in the area of cybernetics and systems. Although CAS theory shares the subject of general properties of complex systems across traditional disciplinary boundaries (like in cybernetics and systems) it relies on computer simulations as a research tool (as pointed out by Holland in 1992 initially [12]), and considers less integrated or "organized" systems, such as ecologies, in contrast to organisms, machines, or enterprises. Many artificial systems are characterized by apparently complex behaviors due to often nonlinear spatio-temporal interactions among a large number of component systems at different levels of organization, they have been termed Complex Adaptive Systems (CAS)

CAS are dynamic systems able to adapt in and evolve with a changing environment. It is important to realize that there is no separation between a system and its environment in the idea that a system always adapts to a changing environment. Rather, the concept to be examined is that of a system closely linked with all other related systems making up an ecosystem. Within such a context, change needs to be seen in terms of co-evolution with all other related systems, rather than as adaptation to a separate and distinct environment ([11] p.2). CAS have several constituent properties (ibid. p.3ff):

- *Distributed Control: There is no single centralized control mechanism that governs system behavior.* Although the interrelationships between elements of the system produce coherence, the overall behavior usually cannot be explained merely as the sum of individual parts.
- *Connectivity*: A system does not only consist of relations between its elements, but also of relations with its environment. Consequently, a decision or action by one part within a system influences all other related parts.
- *Co-evolution With co-evolution, elements in a system can change based on their interactions with one another and with the environment. Additionally, patterns of behavior can change over time.*
- *Sensitive Dependence on Initial Conditions: CAS are sensitive due to their dependence on initial conditions. Changes in the input characteristics or rules are not correlated in a linear fashion with outcomes. Small changes can have a surprisingly profound impact on overall behavior, or vice-versa, a huge upset to the system may not affect it. ... This means the end of scientific certainty, which is a property of "simple" systems (e.g., the ones used for electric lights, motors and electronic devices*). Consequently, socio-technical systems, are fundamentally unpredictable in their behavior. *Long-term prediction and control are therefore believed to not be possible in complex systems.*
- *Emergent Order: Complexity in complex adaptive systems refers to the potential for emergent behavior in complex and unpredictable phenomena.* Once systems are not in an equilibrium they tend to *create different structures and new patterns of relationships. … Complex adaptive systems function best when they combine order and chaos in an appropriate measure –* this phenomenon has been termed *Far from Equilibrium*. CAS in their dynamics combine both order and chaos, and thus, *stability and instability, competition and cooperation, order and disorder –* being termed *State of Paradox*.

In Figure 1 a schema of complex socio-technical system as a group of different types of elements is shown, existing far from equilibrium, when forming interdependent, dynamic evolutionary networks that are sensitive dependent and fractionally organized (cf. [13]). Taking a CAS perspective requires system thinking in terms of networked, however modular elements acting in parallel (cf. [14]). In socio-technical settings, these elements can be individuals, technical systems or their features. Understood as CAS they *form and use internal models to* ***anticipate*** *the future, basing current actions on expected outcomes. It is this attribute that distinguishes complex adaptive systems from other kinds of complex systems; it is also this attribute that* *makes the emergent behavior of complex adaptive system intricate and difficult to understand* ([12], p.24). Figure 2 visualizes the behavior-specific dynamics of CAS.

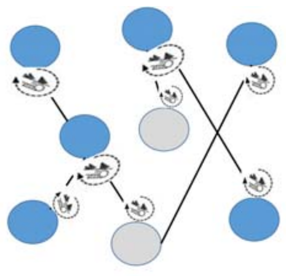

Figure 1. A schema of a complex system.

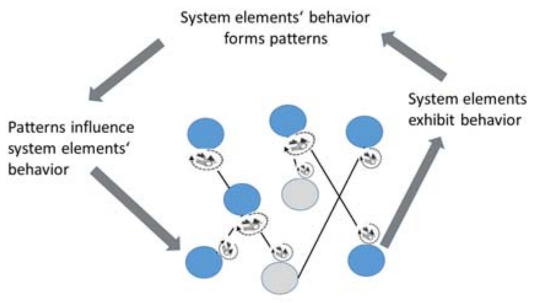
System elements' behavior forms patterns
System elements exhibit behavior
Patterns influence system elements' behavior

Figure 2. A schema of a Complex Adaptive System.

According to CAS theory, in CAS settings each element sends and receives signals in parallel, as the setting is constituted by each element's interactions with other elements. Actions are triggered upon other elements' signals. In this way, each element also adapts and thus, evolves through changes over time. Self-regulation and self-management have become crucial assets in dynamically changing socio-technical settings, such as organizations ([15][16]). Self-organization of concerned stakeholders as system elements is considered key handling requirements for change. However, for self-organization to happen, stakeholders need to have access to relevant information of a situation. Since the behavior of autonomous stakeholders cannot be predicted, a structured process is required to guide behavior management according to the understanding of stakeholders and their capabilities to change their situation individually [15] [17].

From the interaction of the individual system elements arises some kind of global property or pattern, something that could not have been predicted from understanding each particular element [11]. A typical emergent phenomenon is a Social Media momentum stemming from the interaction of the users when deciding upon a certain behavior, such as spontaneous meetings. Global properties result from the aggregate behavior of individual elements. Although it is still an open question how to apply CAS to engineering systems with emergent behavior (cf. [14]), in case of socio-technical system design pre-programmed behavior is a challenging task, as humans may change behavioral structures in response to external or internal stimuli. As such, stakeholders in these systems (self-)organize evolvement and adapt to a changing environment, usually generating more complexity in the process.

System-of-Systems

System-of-Systems (SoS) is considered an effective way of handling CAS, in particular when developing complex artefacts in a structured way [18]. According to IEEE's Reliability Society, thereby a system is *a group of interacting elements (or subsystems) having an internal structure which links them into a unified whole. The boundary of a system is to be defined, as well as the nature of the internal structure linking its elements (physical, logical, etc.). Its essential properties are autonomy, coherence, permanence, and organization* ([19], p.1). A System-of-Systems (SoS) is a system that involves several systems *that are operated independently but have to share the same space and somehow cooperate* (*ibid*, p.2).

As such, they have several properties in common: operational and managerial independence, geographical distribution, emergent behavior, evolutionary development, and heterogeneity of constituent systems (*ibid*.). These properties affect setting the boundaries of SoS and the internal behavior of SoS, and thus, influences methodological SoS developments ([20], p. 206). SoS are distinct with respect to

(1) autonomy where constituent systems within SoS can operate and function independently and the capabilities of the SoS depends on this autonomy,

(2) belonging (integration), which implies that the constituent systems and their parts have the option to integrate to enable SoS capabilities,

(3) connectivity between components and their environment,

(4) diversity (different perspectives and functions),

(5) emergence (foreseen or unexpected) (ibid.)

Several structures and categorization schemes have been used when considering complex systems as System-of Systems, ranging from closed coupling (systems within systems) to loosely coupling (assemblage of system). They constitute embodied systems cooperating in an interoperable way (cf. [21]), allowing for autonomous behavior of each system while contributing through collaboration with other systems, in order to achieve the objective of the networked systems (SoS) [22] – see also next section.

Referring to structural and dynamic complexity, *structural complexity derives from i) heterogeneity of components* *across different technological domains due to increased integration among systems and ii) scale and dimensionality of connectivity through a large number of components (nodes) highly interconnected by dependences and interdependences. Dynamic complexity manifests through the emergence of (unexpected) system behavior in response to changes in the environmental and operational conditions of its components* ([19], p.1).

A typical SoS example are contextualized apps available on a smartphone. Each of them can be considered as a system. When adjusting them along a workflow, e.g., to raise alert and guide a patient to the doctor, in case certain thresholds with respect to medical conditions are reached for a specific user, several of these systems, such as blood pressure app, calendar app, and navigation app, need to be coordinated and aligned for personal healthcare, updating the task manager of the involved users. In this case, the smartphone serves as SoS carrier, supporting the patient-oriented redesign of the workflow, and thus, the SoS structure. The apps of the smartphone can still be used stand-alone, while the smartphone serves as a communication infrastructure and provider of networked healthcare-relevant subsystems. It is the latter property qualifying the smartphone as carrier of a SoS.

APP'IFIED SYSTEM DESIGN

In order to align apps to meet user or situation requirements they need to be interoperable from a semantic perspective. In the following semantic interoperability is discussed in the context of System-of-Systems and dynamic adaptation. As subject-oriented models allow for CAS-specific behavior representations of apps, a corresponding development process can be defined. It is given in the second subsection, demonstrating modeling and process-driven prototyping.

Semantically Interoperable Systems

We recognize appi'fication from a System-of-Systems (SoS) perspective, which requires recognizing context of app use to scope the SoS and the process of adaptation. Stakeholders need to *understand the whole system beyond its elements, sub-systems, assemblies and components, and recognize how each element / sub-system / assembly / component functions as part of the entire system. They are multifaceted, able to consider issues from a wide range of perspectives and points of view and possess a generalist's perspective* ([23], p.276). They also need to *understand the interconnections and the mutual influences and interrelations among system elements. Systems thinking involves thinking about the system's interactions, interrelationships, and interdependencies of a technical, social, socio-technical or multi-level nature* (*ibid*.).

In doing so, developers lay ground for emergent properties of systems, effecting perspectives beyond engineering an isolated system. A systemic representation, such as a SoS specification, enables designing user-specific systems. Thereby, several constellations may occur ([24], p. 28).

- ***Integrated Systems*** *(not interoperable): The systems functionally depend on each other. Once one of the systems fails, the overall system fails.*
- ***Integrated Interoperable Systems****: The systems share the same meta-model. While remaining independent systems, the syntax and semantics of the exchanged information are well defined and may be exchanged seamlessly.*
- ***Unified Interoperable Systems***: *Here an abstract layer is introduced in the communication. Referring to elements on that layer requires abstracting from individual details. However, the individual systems are free to modify their own syntax or semantics as long as it is possible to relate information to the abstraction layer commonly agreed upon.*
- ***Federated Interoperable Systems****: This is the loosest interoperable approach. It allows, but at the same time also requires, negotiating in an ad-hoc manner with respect to exchanged information syntax and semantics.*
- ***Compatible but not interoperable Systems****: Compatible systems exist next to each other, but no meaningful information is exchanged between them. These systems do not interfere with the other system. However, it is not possible to provide larger functionality through a meaningful combination of the individual system's functions.*

Although with respect to semantic interoperability, the state of the art is to use ontologies (cf. [25]), in app'ification systemic relationships require a more dynamic protocol [26]. It needs also to go beyond Domain Specific Languages (DSL) (cf. [27]), to become more agile and proactive. App'ified SoSs need to be described and represented as being autonomous while being interconnected with each other in order to contribute to a goal at a higher level.

Interoperability is required on the one hand to recognize autonomous system nature and on the other hand, at the same time to recognize emergent behavior due to systems' connectivity to other systems [28]. In case of autonomous systems kept isolated, neither connectivity nor diversification of behavior is enabled. As such, these system cannot be aligned explicitly according to a common goal, as required for designing semantically interoperable system [18]. The involved systems need to have the ability to cooperate with each other, in particular agreeing upon a common way of interaction in order to collaborate and share information (cf. [29]).

For instance, in case of interoperable home health care, task manager app entries can represent both, links to other apps including transfer of data, e.g., blood pressure to be sent, and individual action items, e.g., reminders to perform a certain activity such as a yoga session. Interoperability is essential not only to overcome system isolation, but also to enable systems to be diverse and emergent in behavior. Hence, both, interaction patterns, behavior requires representation and are subject to structural change. Changes may occur on the level of individual systems, as well as on the level of interaction of systems, affecting the overall SoS behavior (cf. [30][31][21]). From the perspective of sharing data, a blog entry referring to blood pressure may be relevant not only for individual reflection on individual healthcare development, but also for adjusting medical treatment with physicians. As such, the blog entry could be part of a patient's individual knowledge repository, and of a medical record of the patient.

Dynamic arrangements require knowing the structure *and* behavior of a system. Hence, in case blog entries should be shared with other systems, blogging does not only require a data format for entries and some structure to inherently arrange them for blogging, but also facilities for editing both to adapt to SoS, e.g., sorting along SoS categories like all tasks referring to blood pressure data.

The integrated representation of the structure and behavior of systems including their environmental conditions can either take into account tasks and related processes, e.g., Kolb et al. [32], Stary et al. [33], allowing for automated creation and adaptation of user interfaces for specific user roles. In case of mapping interaction sequences to workflow, e.g., as proposed by Franke et al. [34] deviations between modeled and actual behavior can be identified, triggering the re-design of technically systems in SoS. In any case, semantic interoperability requires context-sensitive understanding and representations (cf. [35][36]). Different contexts of use leads to different requirements induce a federated approach to SoS engineering. Context-relevant but implementation-independent representations allow capturing and resolving interoperability issues at design time [21].

Subject-oriented Engineering

In this section we first provide the subject-oriented perspective on SoS development before discussing its application for contextual app'ification.

Subject-oriented modeling and execution support

Subject-orientation is rooted in perceiving the world as parallel processes, each of which encapsulating a certain behavior and being able to trigger other processes and being triggered by other processes [37][38]. In subject-oriented SoS design systems are viewed as emerging from both the interaction between systems (represented as subjects) and their specific behaviors encapsulated within the individual systems [39]. According to this perspective, systems operate in parallel and can exchange messages, with the latter establishing a SoS.

The relations between systems of the SoS represent their context and scope the SoS. In that SoS, the systems operate autonomously and concurrently. When they are represented in a subject-oriented way systems are termed subjects. As behavior encapsulations they can be assigned to an entity that is capable of performing the encapsulated actions, from an organizational and/or technological perspective. This entity can be a human, an app, a machine (e.g., a robot), a device (e.g., a sensor), or a combination of these. Subject-oriented systems execute two different types of actions:

- *System-specific actions*: They correspond to self-contained activities and do not involve interacting with other systems, such as checking the blood pressure of a patient.
- *Communication with other systems*: These actions are concerned with exchanging messages between systems, i.e. sending and receiving messages, e.g., connecting a medical information system to the blood pressure measurement system of a patient, in order to receive medical feedback after collecting blood pressure data.

In subject-oriented representations, systems are represented as one of five core symbols listed in Figure 3. In Figure 4 subjects (i.e. systems) are Customer, Order Handling, and Shipment. Subject-oriented design requires two types of diagrams:

- *Subject Interaction Diagrams (SIDs).* They provide a global view of a SoS, comprising its systems and the messages they exchange. The SID of a simple ordering setting is shown in Figure 4.
- *Subject Behavior Diagrams (SBDs).* They provide an internal view on individual systems through sequences of states representing local actions (functions) and communicative actions (sending and receiving messages). State transitions are represented as arrows, with labels indicating the outcome of the preceding state (see Figure 5).

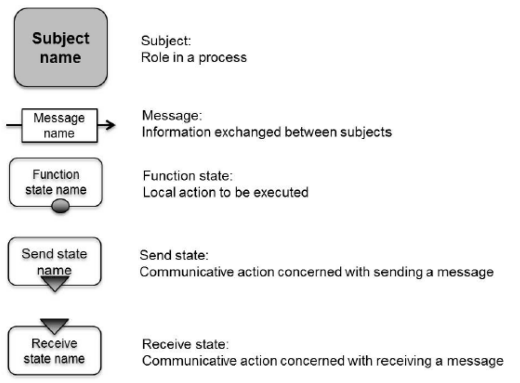
Subject name
Subject:
Role in a process
Message name
Message:
Information exchanged between subjects
Function state name
Function state:
Local action to be executed
Send state name
Send state:
Communicative action concerned with sending a message
Receive state name
Receive state:
Communicative action concerned with receiving a message

Figure 3. Diagrammatic elements of subject-oriented specifications.

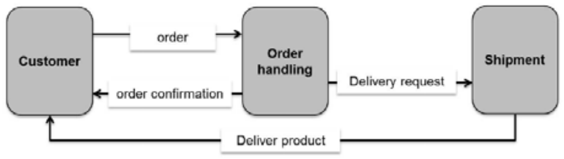

Figure 4. Handling orders – a SoS view via a Subject Interaction Diagram (SID).

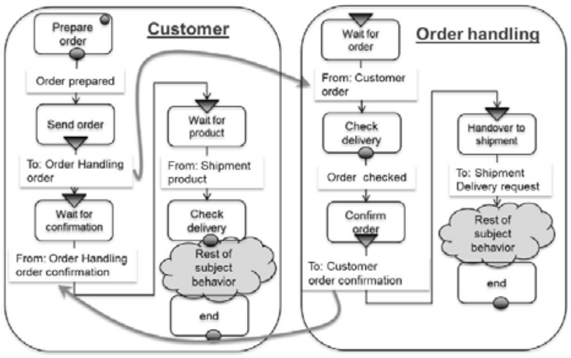

Figure 5. System interaction for handling orders between Customer (SBD) and Order handling system (SBD).

Given these notational capabilities SoS designs are characterized by

- a simple communication protocol (using SIDs) to relate systems
- standardized system interfaces (enabled by SBDs encapsulating system behavior)
- scalability in terms of complexity and scope, since each system can be decomposed according to functional or interaction patterns.

Subject-oriented representations allow representing each system's context in terms of communication relations to other systems. Hence, context-specific requirements, e.g., when creating a SoS for home health care, need to be mapped to relevant patterns of interactions between systems. The approach scales when SoS can be defined through dynamic and situation-sensitive system formations of systems in the sense of Complex Adaptive Systems (see section on CAS).

Since the communication relations between systems define a control flow for SoSs, validated subject-oriented specifications can be executed without further transformation (see Figure 6). In this way, users and stakeholders can be involved in modeling, refining, and implementing their SoS specifications.

Their modular structure is compatible with similar implementation approaches, such as service-oriented architectures and agent-oriented software systems [40].

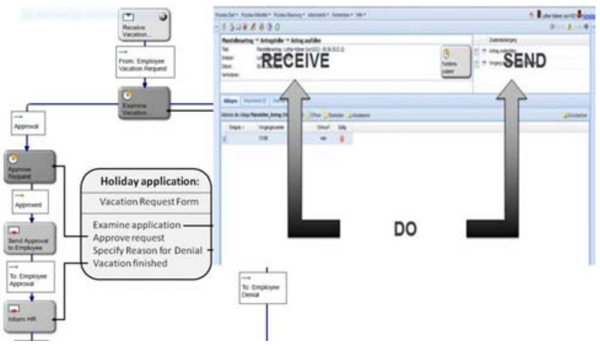

Figure 6. Interactive experience when executing subject-oriented system specifications.

Supporting context-sensitive SoS app'ification

Subject-oriented SoS design seeks to assist user-centered system development by providing a methodology that presents behavior-relevant information encoded in apps. These characteristics enable users to be engaged more effectively via behavior representations. The SoS-based app'ification procedure has been designed according to the principles of subject-oriented system modeling and is structured as follows (cf. [41], Appendix):

(1) An app is a system which is represented by its behavior.
(1.1) A specification of an app as system does not imply any actor or technology that could be used to execute the described behavior. It is implementation-independent.
(1.2) Apps can communicate with each other by exchanging messages (2).

(2) A Message has a name and a payload.
(2.1) The name should express the meaning of a message informally.
(2.2) The payload corresponds to the data transported.

(3) Apps have behavior representations. Internally,
(3.1) an app executes local activities
(3.2) an app sends messages to other apps
(3.3) an app expects messages from other apps
(3.4) an app performs all these activities in sequence – they are defined in an app's behavior specification.

(4) Subject-oriented SoS specifications concern a specific behavior embedded in some context.
(4.1) Context is defined by the needs of users and properties of the situation at hand, including a set of apps (1,3).
(4.2) Context is expressed in terms of messages (2) for each app (1).
(4.3) Context is also provided by the technological infrastructure by which a system is technical part of, allowing to execute a specified app behavior (3).

Hence, contextual app'ification is based on the behavioral entities or abstract resources involved in a user-relevant setting. It is enabled and set up by systems termed subjects.

The behavior of a resulting SoS is determined by the subjects' exchanges of messages (i.e. system interactions).

(5) Modeling requires several activities, namely, the specification of
(5.1) the scope of modeling
(5.2) the apps to be processed for task completion
(5.3) interactions the apps are part of
(5.4) the messages they send or receive through each interaction
(5.5) the behavior of each app encapsulating functions and interactions

(6) Subject Interaction Diagram (SID)
(6.1) (5.2) to (5.4) constitutes a Subject Interaction Diagram representing an app'ified SoS.
(6.2) A Subject Interaction Diagram is the most abstract diagrammatic level of describing how task are completed
(6.3) For each subject of a Subject Interaction Diagram a Subject Behavior Diagram (7) needs to be constructed for a complete and coherent app'ified SoS model.

(7) Subject Behavior Diagram (SBD)
(7.1) (5.5) for each subject constitutes a Subject Behavior Diagram.
(7.2) Once a app has been identified in (6), the behavior of each app can be defined in a subject-oriented way.
(7.3) An app's behavior is described by three states (send, receive, internal function) and transitions between these states. Hence, when specifying the behavior of each subject, a sequence of sending and receiving messages, and activities to be set for task accomplishment need to be represented.
(7.4) The subject-oriented description of an app defines the sequence of sending and receiving messages, or the processing of internal functions, respectively.
(7.5) A Subject Behavior Diagram is the most concrete diagrammatic level of adjusting apps, thus establishing a subject-oriented SoS.

(8) The states of a Subject Behavior Diagram represent operations.
(8.1) They are active elements of the subject description.
(8.2) States are implemented by functions or services.
(8.3) State transitions are necessary to exchange and manipulate data objects.

(9) Data objects are
(9.1) data affected by operations of an app.
(9.2) exchanged between apps by sending and receiving messages
(9.3) processed through services.

In the following we exemplify a scheme how context-sensitive app'ification develops. We consider a home healthcare setting, as this kind of settings is under research increasingly from a socio-technical perspective (cf. [42]).

Consider the following case: An elder patient needs particular home healthcare support. Her blood pressure needs to be checked every other day. The patient needs to be reminded to trigger the measurement. Based on the results of the blood pressure check medical advice could be consulted. Three apps need to be aligned for effective monitoring and consultancy: (i) blood pressure measurement, (ii) medical expert consultancy, (iii) medication reminder & scheduling. In case the blood pressure reaches a certain threshold, medical experts need to be consulted, and eventually individual medication needs to be adjusted, triggering the reminder & scheduling system.

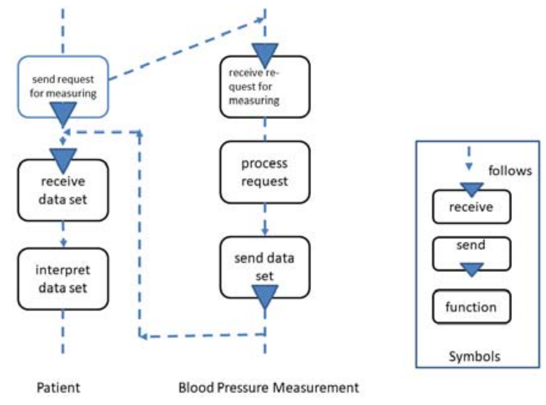

Figure 7. SBD-home healthcare app'ification example involving Patient and Blood Pressure Management.

In line with the subject-oriented SoS modeling each app is represented as individual system (subject), leading to a Subject Interaction Diagram involving 4 different systems and exchanging messages - in addition to the listed apps the patient also needs to be represented, in order to capture her behavior. As shown in the SBD interactions in Figure 7, the patient triggers the blood pressure measurement by sending a message to the corresponding subject (app), and receives a data set for interpretation.

Activating the medical advice could be modeled in a SBD through standard send-receive pattern with a medical consultancy system. The decision making procedure could also be added later, and the development requires CAS capabilities for app'ified SoSs. Putting the patient in control of triggering medical advice as dynamically occurring requirement means she triggers medical consultancy after checking her blood pressure, providing an up-to-date data set.

In order to mark such behavioral changes, and even adapt them further according to changing requirements, Fleischmann et al. [43,44] suggested the message guard concept for exception (i.e. non-routine behavior) and event handling. Events in subject-oriented representations occur in a particular domain, such as home health care, and represent a novel need or a change in requirements. Events are interpreted as messages that can contain structured (data objects) or unstructured data (attachments).

The message guard reacting to the new behavior request is specified in Figure 8. Event producer, sending events in that case is the patient, requesting to decide by herself on a case-by-case basis. The event consumer is the message guard extending behavior capabilities. It receives events and processes them. In the example it may lead to involving medical advice, when the patient feels not comfortable with her interpretation of blood pressure date (see right branch). A function of a SBB allowing for behavior extension is marked with a triangle – see 'interpret data set' in Figure 8.

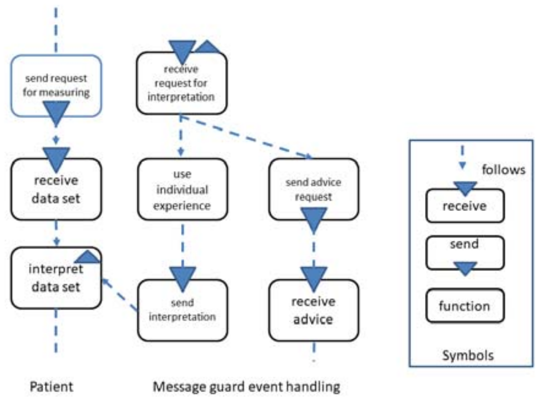

Figure 8. Capturing dynamic user control on medical advice.

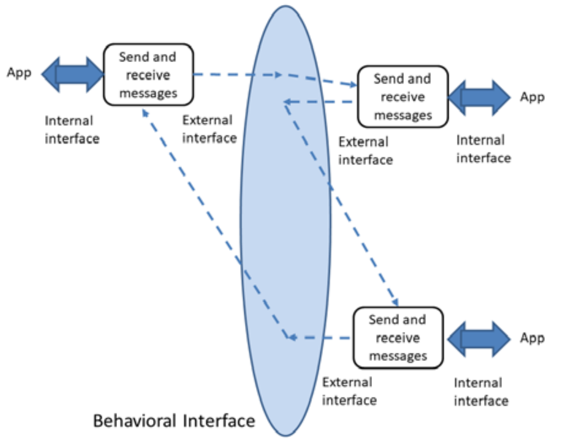

Figure 9. The Behavioral Interface enabling cross-app and cross-platform communication.

Utilizing subject-orientation, message exchange is the core feature for SoS app'ification including dynamic adaptation. For cross-app and cross-platform communication a message-based middleware could provide effective runtime support. Meyer et al. [45, 46] have developed a corresponding framework as shown in Figure 9. Each app has an external and internal interface. The external one could be part of a routing component allowing to interact with other apps while the internal behavior is not visible to other apps. The Behavioral Interface keeps track, in which sequence messages are exchanged between apps, even across platforms. In this way also events can be exchanged, triggering behavior modifications, and thus, ad-hoc changes while maintaining semantic interoperability of the involved apps (cf. [47]).

CONCLUSION

Digitalization of goods and processes enables informed design of services and systems by involved stakeholders, once they are provided with user-centered design instruments. Following a communication-driven perspective, system specifications can not only encapsulate behavior of apps, but also capture their context in terms of message exchanges. The underlying adaptive system-of-systems concept enables contextual app'ification of interactive systems, including the behavior of situation-or task-relevant actors, such as doctors, care takers, administration, and patients involved in home healthcare.

Subject-oriented representations contain functional and interactional aspects, and thus, contextual parameters to capture a situation relevant for system use. They can be used to generate interactive experience without touching the functionality of involved apps, rather extending their scope due to interactional alignment. At design and runtime semantic interoperability of involved systems can be ensured by context-sensitive behavioral interfaces.

Future research will, on the one hand focus on usability and acceptance of model-based design instruments for potential users, since, so far situation- or task-sensitive information has been captured. On the other hand, future work will focus on increasing the intelligence of appi'fied systems, as the existing structure of aligned apps can be changed dynamically. Activities could then be proposed by system-of-systems, in particular referring to re-occurring behaviors or reference models. The more digital footprints become available, the more context can be considered for re-designing complex systems, such as Internet of Things or cyber-physical systems (cf. [48]).

REFERENCES

1. Burroughs, B. YouTube kids: The app economy and mobile parenting. *Social Media+Society 3,* 2 (2017), DOI: https://doi.org/10.1177/2056305117707189.
2. Mansour, E., Sambra, A.V., Hawke, S., Zereba, M., Capadisli, S., Ghanem, A., Aboulnaga, A. and Berners-Lee, T. A demonstration of the solid platform for social web applications. *Proc. of the 25th International Conference Companion on World Wide Web*, International World Wide Web Conferences Steering Committee (2016), 223-226.

3. Manresa-Yee, C. and Amengual, E. Tailoring ISO/IEC 12207 for Usability Engineering. *International Journal of Engineering Education 32*, 2 (2016), 886-893.
4. Magües, D.A., Castro, J.W. and Acuña, S.T. Requirements Engineering related Usability Techniques adopted in Agile Development Processes. *Proc. SEKE 2016*, Knowledge Systems Institute (2016), 537-542.
5. Shelley, C. Social agendas. In *Design and Society: Social Issues in Technological Design*. Springer International Publishing (2017), 105-124.
6. Fahl, S., Harbach, M., Perl, H., Koetter, M., and Smith, M. Rethinking SSL development in an appified world. *Proc. of the 2013 ACM SIGSAC conference on Computer & communications security,* ACM Press (2013), New York, 49-60.
7. Batran, A., Erben, A., Schulz. R., and Sperl, F. *Procurement 4.0: A survival guide in a digital, disruptive world*. Campus Verlag, Frankfurt (2017).
8. Jia, Y. J., Chen, Q.A., Wang, S., Rahmati, A., Fernandes, E., Morley, M. and Prakash, A. ContexIoT: Towards providing contextual integrity to appified IoT platforms. *Proc. of the 21st Network and Distributed System Security Symposium (NDSS'17)*, Internet Society (2017), 1-15.
9. Leonhard, G. and von Kospoth, C.A.G. Exponential technology versus linear humanity: Designing a sustainable future. In *Sustainability in a Digital World*, Springer International Publishing (2017), 77-83.
10. Gidey, H. K., Marmsoler, D. and Eckhardt, J. Grounded Architectures: Using Grounded Theory for the design of software architectures. Proc. *IEEE International Conference on Software Architecture Workshops (ICSAW),* IEEE Computer Society (2017), IEEE Computer Society Press, New York, 141-148.
11. Chan, S. Complex adaptive systems. *ESD 83 Research Seminar in Engineering Systems*, 31 (2001).
12. Holland, J.H. Complex adaptive systems. *Daedalus 121,* 1 (1992), 17-30.
13. Fichter, Lynn S., Pyle, E.J. and Whitmeyer, S.J. Expanding Evolutionary Theory Beyond Darwinism with Elaborating, Self-Organizing, and Fractionating Complex Evolutionary Systems. *Journal of Geoscience Education 58*, 2 (2010), 58-64.
14. Holland, J. H. (2006). Studying complex adaptive systems. *Journal of Systems Science and Complexity*, *19*(1), 1-8.
15. Allee, V. Value-creating networks: organizational issues and challenges. *The learning organization 16*, 6 (2009), 427-442.
16. Firestone, J.M. and McElroy, M.W. *Key issues in the new knowledge management*. Routledge, New York (2003).
17. Stary, C. Non-disruptive knowledge and business processing in knowledge life cycles–aligning value network analysis to process management. *Journal of Knowledge Management 18*, 4 (2014), 651-686.
18. Jamshidi, M. (Ed.) *System of systems engineering: innovations for the twenty-first century* (Vol. 58). John Wiley & Sons, New York (2011).
19. IEEE-Reliability Society. Technical Committee on 'Systems of Systems'. *Systems-of-Systems*, White Paper, 5p., IEEE, IEEE Society Press (2014).
20. Jaradat, R. M., Keating, C. B. and Bradley, J. M. A. histogram analysis for system of systems. *International Journal of System of Systems Engineering 5, 3 (2014)*, 193-227.
21. Stary, C. and Wachholder, D. System-of-systems support—A bigraph approach to interoperability and emergent behavior. *Data & Knowledge Engineering* 105 (2016), 155-172.
22. Maier, M. W. Research challenges for systems-of-systems. *Proc. International Conference on Systems, Man and Cybernetics, Vol. 4,* IEEE, IEEE Society Press (2005), 3149-3154.
23. Frank, M. (2012). Engineering Systems Thinking: Cognitive Competencies of Successful Systems Engineers. *Procedia Computer Science* 8 (2012), 273-278.
24. Weichhart, G. and Stary C. Traceable pedagogical design rationales for personalized learning technologies: an interoperable system-to-system approach. *International Journal of People-Oriented Programming* 3, 2 (2014), 25-55.
25. Weichhart, G., Stary, C. and Vernadat, F.B. Enterprise Modeling for the interoperable and knowledge-based enterprise. *International Journal on Production Research 55* (2017), in press.
26. Zacharewicz, G., Diallo, S., Ducq, Y., Agostinho, C., Jardim-Goncalves, R., Bazoun, H., Wang, Z. and Doumeingts, G. Model-based approaches for interoperability of next generation enterprise information systems: state of the art and future challenges. *Information Systems and e-Business Management* (2016).
27. Weichhart, G. and C. Stary. A Domain Specific Language for Organisational Interoperability. *Proc. On The Move Conferences and Workshops*, Springer (2015), 117-126.
28. Bezerianos, A. and McEwan, G. Presence Disparity in Mixed Presence Collaboration. *Ext. Abstracts CHI 2008*, ACM Press (2008), 35–37.
29. Curry, E. System of systems information interoperability using a linked dataspace. *Proc. 7th International Conference* on *System of Systems*

Engineering, IEEE, IEEE Society Press (2012), 101-106.

30. Boardman, J. and Sauser, B. System of systems — the meaning of. *Proc. International Conference* on *System of Systems Engineering*, IEEE, IEEE Society Press (2006), 118-123.
31. Baldwin, W.B. and Sauser B. Modeling the characteristics of system of systems, *Proc. International Conference* on *System of Systems Engineering*, IEEE, IEEE Society Press (2009), 1–6.
32. Kolb, J., Hübner, P. and Reichert, M. Automatically Generating and Updating User Interface Components in Process-Aware Information Systems. *Proc. On the Move to Meaningful Internet Systems*, Springer 7565 Lecture Notes in Computer Science (2012), 444–454.
33. Stary, C. and Wachholder, D. Context Control of Interoperable Interactive Systems. *Proc. 17th Conference on Business Informatics* vol. 1, IEEE, IEEE Society Press (2015), 232-241.
34. Franke, J., Charoy, F. and El Khoury, P. Framework for coordination of activities in dynamic situations. *Enterp. Inf. Syst.* 7, 1 (2013), 33–60.
35. Panetto, H. and Cecil, J. Information systems for enterprise integration, interoperability and networking: theory and applications. *Enterp. Inf. Syst.* 7, 1 (2013), 1–6.
36. Vernadat, F.B. Technical, semantic and organizational issues of enterprise interoperability and networking. *Ann. Rev. Control* 34, 1 (2010), 139–144.
37. Fleischmann, A., Schmidt, W., Stary, C., Obermeier, S. and Börger, E. *Subject-oriented business process management*. Springer Publishing Company, Incorporated (2012).
38. Fleischmann, A., Schmidt, W. and Stary, C. Subject-oriented business process management. In *Handbook on Business Process Management 2*, Springer, Berlin (2015), 601-621.
39. Stary, C. System-of-Systems Design Thinking on Behavior. *Systems 5*(1), 3 (2017).
40. Fleischmann, A., Kannengiesser, U., Schmidt, W. and Stary, C. Subject-oriented modeling and execution of multi-agent business processes. *Proc. of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies Volume 02*. IEEE, Computer Society (2013), 138-145.
41. Fleischmann, A., Schmidt, W. and Stary, C. *S-BPM in the wild: Practical value creation*. Springer Publishing Company, Incorporated (2015).
42. Ackerman, M.S., Goggins, S.P., Herrmann, T., Prilla, M. and Stary, C. *Designing Healthcare that Works. A Socio-technical Approach*. Academic Press, Cambridge, MA (2017).
43. Fleischmann, A., Schmidt, W., Stary, C. and Strecker, F. Nondeterministic events in business processes. *Proc. International Conference on Business Process Management*. Springer (2012), 364-377.
44. Fleischmann, A., Schmidt, W. and Stary, C. Complex Event Processing in e-Services. *Proc. 9th IEEE International Conference on Developments in eSystems Engineering (DeSE).* IEEE Computer Society Press (2016), 251-259.
45. Meyer, N., Feiner, T., Radmayr, M., Blei, D., Fleischmann, A. Dynamic catenation and execution of cross organisational business processes-the jCPEX! approach. *Proc. International Conference on Subject-Oriented Business Process Management*. Springer (2010), 84-105.
46. Meyer, N., Radmayr, M., Heininger, R., Rothschädl, T., and Fleischmann, A. Platform for Managing and Routing Cross-Organizational Business Processes on a Network Router. *Proc. S-BPM ONE-Learning by Doing-Doing by Learning*, ed. Schmidt, W., Springer (2011), 175-189.
47. Rothschädl, T. Ad-hoc adaption of subject-oriented business processes at runtime to support organizational learning. *Proc. S-BPM ONE–Scientific Research*, ed. Stary, C., Springer, 104 Lecture Notes in Business Information Processing (2012), 22-32.
48. Neubauer, M. and Stary, C. *S-BPM in the Production Industry*. Springer International Publishing (2017).