Middleware for Work Support in Industrial Contexts (MiWSICx)

Hitesh Dhiman[a] and Carsten Röcker[b]

[a]TH OWL University of Applied Sciences and Arts, Lemgo Germany
E-mail: hitesh.dhiman@th-owl.de
[b]TH OWL University of Applied Sciences and Arts and Fraunhofer IOSB-INA, Lemgo Germany
E-mail: carsten.roecker@th-owl.de

Abstract

It is generally acknowledged that technological innovation is leading to an increase in the complexity of industrial work. Hence, *work assistance* has emerged as an important theme in the context of cyber-physical production systems and Industry 4.0 to assist workers in assembly, logistics, maintenance and supervision. Recent research in this domain has focused on demonstrating assistance applications using mobile computing devices such as tablets, smartphones, AR/VR glasses and wearables, but the aspects of technology induced complexity in industrial work - *distribution*, *concurrency, information complexity*, and *variability of information interaction*, and their subsequent effect on human workers is yet to be tackled.

This paper has two core contributions: first, it reframes the problem of complex industrial work through *activity theory*, which leads to a conceptual model that couples human information needs to interactive artefacts within an *activity context*. Second, the problem of assistance is viewed as managing *information flow* between multiple devices grouped into fluid and adaptive activity contexts, managed by MiWSICx, (Middleware for Work Support in Industrial Contexts) a novel, distributed middleware designed using the *actor model* of concurrent computation.

Keywords –
Cyber physical production systems, assistance, complexity, middleware, actor model.

1 Introduction

Owing to extensive miniaturization and digitalization, the factory floor is marked by an increasing use of cyber-physical systems (CPS), or in manufacturing, cyber-physical production systems (CPPS), referring to the interconnected, yet distributed, nature of physical processes and their control. In the German speaking part of the world, the term Industry 4.0 is used [1]. With advancement in communication technologies and machine learning algorithms, it is postulated that many existing activities will be automated and newer, complex activities will be created for human workers.

These activities stem from the need to prevent breakdowns in workflow on increasingly automated manufacturing lines. This is because as the degree of automation increases, it also exposes the complexity, inconsistencies and variation in manufacturing, the effects of which were until now smoothed over by human workers' ingenuity and experience [2]. As such, introducing technology does not always replace a human weakness, it perpetuates new strengths and weaknesses, in often unanticipated ways [3]. While automating a system, by corollary, implies reducing its dependency on human intervention, in most cases, the human intervenes only when the system cannot handle a situation.

The proliferation of multiple modes of interaction via mobile devices has enriched the possibilities of designing and configuring assistance – it is argued that by increasing the number and modality of devices, the corresponding flexibility of providing situational assistance is improved [4][5], but there exists a gap between the paradigms that have led the development of distributed interaction thus far, and what is needed in the approaching era of CPPS. Without a proper understanding of the underlying context of human work, we lack the conceptual tools to design and deploy multi-device assistance.

In the next sections, we explore both the conceptual and technical foundations that led to the development of MiWSICx (Middleware for Work Support in Industrial Contexts). Section 2 introduces the nature of work in CPPS, in view of which the contribution of activity theory is highlighted in section 3, following which section 4 develops the technical foundation of MiWSICx. Section 5 concludes the paper.

2 Nature of Work in a CPPS

Industry 4.0 commits to a future consisting of smart factories as opposed to deserted factories [6]. According to Gorecky et al. [7], human workers are seen "as the most flexible entity in cyber-physical production systems", and they will be "faced with a large variety of jobs ranging from specification and monitoring to verification of production strategies." Supported by technology, human workers will achieve this flexibility by relying on their own skills, knowledge and creativity, supported by an ecology of both specialized and multi-purpose devices, novel ways of interaction, analysis and visualization of data distributed across multiple spatial contexts.

In the following sections, we elucidate some of the qualities of future production work both in its technical and human form and investigate how the former affects the latter.

2.1 Operational Flexibility and Concurrency

While principles of *cellular manufacturing* and *group technology* have focused on optimizing the components of a manufacturing process, for example machines, material handling, product mix, part routing/sequencing and workstation layout, many models of manufacturing in Industry 4.0 aim for flexible and complex production environments via distributed services [8] or agent-based systems [9]. In such smart factories [10], on-demand and completely customizable smart products [11], can be manufactured on the same manufacturing lines.

While this technical flexibility allows technology reconfiguration and deployment across various hierarchical levels, it does not imply that technology alone can handle this variability. For a human worker, tasks become increasingly *concurrent* - the variety and inter-dependency of technical and logistical tasks, both in terms of their location and activities, is higher, demanding not only explicit but also tacit knowledge. As fewer workers tend to more and more machines, mobility and task switching is necessary.

2.2 System Complexity and Variability of Information Interaction

There seems to be no one definition of complexity, nonetheless, complexity in the industrial domain has drawn a large amount of research. According to El Marghay et al. [12], the complexity of any system manifests itself in *functional*, *structural*, *spatial*, and *temporal* domains. Lee and Wieringa [13] identify the shaping factors for process and control system complexity, and list three factors – *variety (of)*, *number (of)* and *links (between)* components, loops, variables, etc. A CPPS thus exhibits complexity in all four core domains of any computational system - *function*, *structure*, *space* and *time*.

The role of HMI in reducing perceived complexity has been previously studied in [14]. However, how this variety can be achieved by human workers is not possible by just implementing individual applications on different devices, since the situation in which the workers find themselves changes the kind of information they look for [15]. The structural ambiguity of complex tasks tends to place a higher cognitive and informational demand on the worker, based on the individual experience and knowledge [16]. Further, the level of a-priori determinability of a task's information requirements shapes how one searches for information – as complexity increases, the number of sources increases, while their specificity reduces [17].

2.2.1 Summary

To recap, a CPPS is characterized by its compositional and informational variety, as well as the sequential and parallel flow of information within these spatially and temporally distributed components. The effect of this variety is not seen at a component level, but at a system level as errors and breakdowns due to unexpected disturbances. Tasks are spatially and informationally diverse, coupled with sources of information and actuation. To handle complexity, workers need to gather information from different sources.

A conventional approach to assistance based on desktop or mobile applications is only sufficient in cases where information needs are a-priori determinable. As an example, it has been demonstrated time and again, in various studies, that help manuals in desktop applications are rarely used [18] [19]. The cited reason is that software designers can hardly anticipate the myriad combinations of *how*, *why*, *what* and *where am I* questions that users ask during application use [20]. The context of assistance is therefore intimately tied to the nature of work, or activities, that users perform, instead of the designed application workflow.

3 From Applications to Activities

A review of industry 4.0 literature reveals that the focus hitherto has been on developing digital applications [21]. The area of human machine interaction in this respect has only recently been explored, where assistance is limited to a single interactive device with assembly or maintenance manuals, usually coupled to a particular workstation or a task [22]. *Multi-device* and *multi-activity assistance* is a field yet to be explored in the industrial domain; at the time of writing this paper, no such framework for industrial work assistance has been found in our literature review.

In section 2, we discussed that the nature of work in a CPPS is distributed and informationally diverse. While situated, task-based applications offer assistance in their own context, combining and switching between multiple activity contexts demands a higher-level view of human activity. One of the prominent fields in HCI research that puts activity at its core is *activity-centric computing* [23], discussed in the next section.

3.1 Activity Centric Computing

Bardram et al. [23] define the following principles of *activity-centric computing*:

a) *Activity-Centered* - A 'Computational Activity' collects in a coherent set a range of services and data needed to support a user carrying out an activity.

b) *Activity Suspend and Resume* - A user can suspend, resume and alternate between activities. Resuming an activity should bring forth the data and services needed for that activity.

c) *Activity Roaming* - An activity supports distribution, such that it can be suspended on one device and resumed on another.

d) *Activity Adaptation* - An activity adapts to the resources available on the device on which it is consumed.

e) *Activity Sharing* - An activity is shared among collaborating users, such that a participant can resume an activity and continue the work of another user.

f) *Context Awareness* - An activity adapts itself to the user's constraints as defined by the availability of resources in the proximity.

The notion of an activity comes from *activity theory*, introduced by Leontiev [24], who proposes that an activity is what links any subject, human or non-human, to objects in the world in which this subject exists. An activity is seen as a three-level hierarchy, shown in Fig. 1. At the highest level, an activity accomplishes a motive by reflecting on an object. An activity can be differentiated from another only when it is intended towards a different object. At the second level, actions are carried out to realize conscious goals. At the third and final level, actions are accomplished by means of subconscious operations which are internalized patterns of behavior acquired through learning or social interactions [25]. Operations are affected by the conditions in which they are carried out. Although a hierarchal relationship between activity, action and operation exists, they are not fixed in their relationship to one another. For example, learning to type using a keyboard is in the beginning an activity, which over time, turns into an operation. Conversely, a *breakdown* [26] forces us to consciously view operations as activities, for example when the keyboard stops responding.

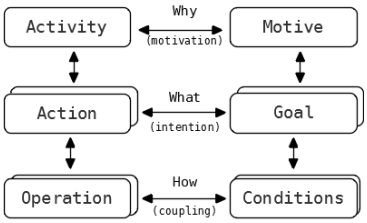

Figure 1. Activity hierarchy, taken from [24].

In activity theory, human activities between a *subject* and an *object* are always *mediated* via tools or *artefacts* [27], as shown in Fig. 2. In other words, we act on objects through tools and artefacts, and in a complex activity system, numerous varieties of mediating artefacts may be involved. Wartofsky [28] distinguishes between *primary artefacts* and *secondary artefacts*, the former being the most obvious in everyday operations, for example tools, while the latter being representations of tools as well as plans, official- documents, explanatory models and notes.

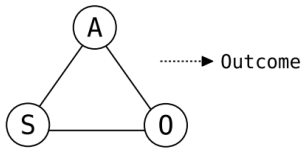

Figure 2. Activity Mediation as conceptualized by Vygotsky [27]. S, O and A refer to *subject*, *object* and *artifact*, respectively.

While activity centric computing has been used to design frameworks for supporting office work, it has not yet been used in the industrial domain. The next section develops a conceptual model that allows us to view a CPPS through an activity-centric perspective.

3.2 Activity Centric CPPS, or ACCPPS

Zamfirescu et al. [29] adopt a human-centric approach in defining a CPPS architecture for a smart factory, shown in Fig. 3.

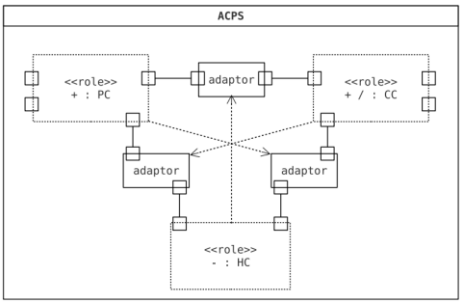

Figure 3. Anthropocentric CPPS model, from [29].

A CPS is divided into three constituent components: the *physical component* (PC), the *cyber/computational component* (CC) and the *human component* (HC). Each

of these components is connected outside the CPPS to a specific physical, computational and social dimension. Adapters transfer information between pairs of these components. From an engineering perspective, the CC and PC are separate components, but by adopting an activity-centric perspective, the CC and the PC are the mediating artefact and the object of the same activity. The mediating nature of the adapters becomes clearer when one turns the model 'inside out', and replaces the HC, CC and PC in a CPPS by their activity centric counterparts, as shown in Fig. 4. In this model, an interactive artefact, A_I allows the user to interact with the CC, whereas the PC is at the receiving end of the action.

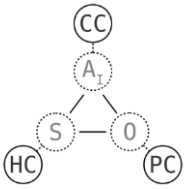

Figure 4. Activity-centric CPPS, based on Fig. 3

Adopting this model allows us to imbue objects and artefacts with spatial and interactive properties, thereby allowing activities to be distributed across objects and devices. However, the availability of a CC on different devices is dependent on the properties of the device, as discussed in the next section.

3.2.1 Interaction Coupling

As mentioned in section 3.1, one of the main tenets of activity-centric computing is to manage the services and data needed to support a user carrying out an activity and to adapt it to the device on which it is consumed. However, with so many different devices at our disposal, we also need to manage the relationship between the device and the data and services it accesses at run time. If we take into consideration the fact that actions are afforded [30] by artefacts, user action can be digitally afforded or prompted via informational cues [31] on interactive artefacts. An action results in a feedback that the user receives directly through the object or mediated via the artefact. The artefacts mediate an intentional act of the user accomplishing a goal, while the feedback aids in reflection on the result of this consciously performed act [32].

This coupling can be better understood by leveraging the human-artefact model, developed by Bødker and Klokmose [33]. In this model, an artefact possesses both instrumental and operational capabilities. An artefact *instrumentalizes* action by helping the user achieve a goal and *operationalizes* it by virtue of its physical abilities. In other words, an interactive device can be characterized by *what* it affords, and *how* it affords it. Goumopoulos et al. [34] use the term *properties* for the physical and informational capabilities of a device. For use in an ACCPPS, the instrumental nature of interactive devices signifies their functionality, whereas properties such as form factor, ergonomics, modalities are all operational properties. Table 1 lists the instrumental and operational capabilities of commonly use interactive devices. These properties allow a matching of a *resource* or a *service* to a corresponding device on runtime.

	Instrumental Capability	Operational Capability
AR Glasses	visualize and anchor information in 3D space	Modalities: Audio,Visual Operations: Point, Grab, Drag Form Factor: Head Mounted Communication: Bluetooth, WLAN
Smartphone	View, edit and communicate visual information	Modalities: Audio,Visual, Haptic Operations: Click, Zoom, Speak Form Factor: Hand held Communication: Bluetooth, WLAN, 4G
Tablet	View, edit and communicate visual information	Modalities: Audio,Visual, Haptic Operations: Click, Zoom, Speak Form Factor: Hand held, large Communication: Bluetooth, WLAN
Wearables	Communicate visual and haptic information	Modalities: Visual, Haptic Operations: Touch Form Factor: worn on wrist Communication: Bluetooth

Table 1. Instrumental and operational capabilities of interactive artefacts.

3.2.2 Activity Context

In a multi-device scenario, each interactive artefact may be delegated a different role, that is, to act as a mediator for different services, resources, feedback, or action, depending on its instrumental and operational capabilities. Further, each device may have both unique and shared resources, allowing both *specialization* and *redundant* modes of interaction [35]. A tablet may for example, be used to search for information due to its visual capability, whereas a smartwatch can be used to deliver feedback due to its proximity to the human body. Several devices and interaction possibilities may be prioritized in terms device preference and availability. The concept of an *activity context* encapsulates this relationship between the action, feedback and resources assigned to each device, as summarized in Table 2.

User Activity Configuration			
	Action	Feedback	Resource
A_{I1}	Visualize data in 3D space.	Visual	Resource 1
A_{I2}	Run trouble-shooting manual, interact with machine	Visual	Resource 2
A_{I3}	Respond to system prompts	Haptic	Resource 1 Resource 2

Table 2. Multi-device, single-user activity context

For a multi-device, multi-user scenario, sharing and modification of resources, along with per user configurations of action and feedback mediation need to

be considered, as shown in Fig. 5. How MiWSICx mediates different activity contexts is explained in the next section.

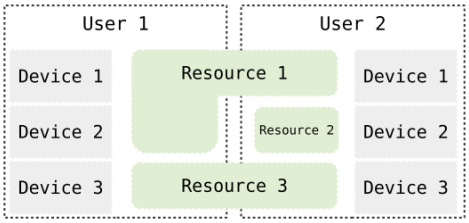

Figure 5. Multi-device, multi-user activity context

4 MiWSICx

As a middleware managing activities in an ACCPPS, MiWSICx manages the following tasks:

a) allow devices to discover a MiWSICx node;
b) communicate with these devices over various channels;
c) provide access to resources and services;
d) communicate with CPPS objects through various protocols;
e) support activity-centric computing for multiple users and on various devices;

The architecture of MiWSICx is composed of two core abstractions: the structural abstraction from which data structures are derived, and the MiWSICx communication protocol. MiWSICx is thus platform agnostic and is designed to be deployed across different machines as nodes in both horizontal and vertical configurations, as shown in Fig. 6.

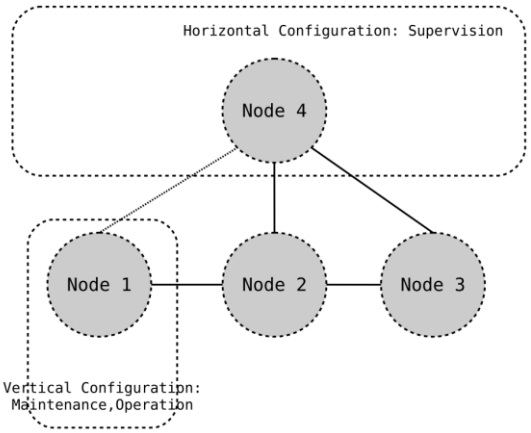

Figure 6. MiWSICx node configurations

4.1 Design

A detailed ontology of the activity entities in MiWSICx is given in Fig. 7. Similar ontologies have been developed by Bardram [36] and Moran et al. [37], but the ontology model used in MiWSICx differs from these ontologies in two ways. First, to adapt to a CPPS it realizes the concept of an object as something that is subject to a change of state as a result of user action and adaptive to this change, and second, it introduces the resource as a facilitator of this change of state.

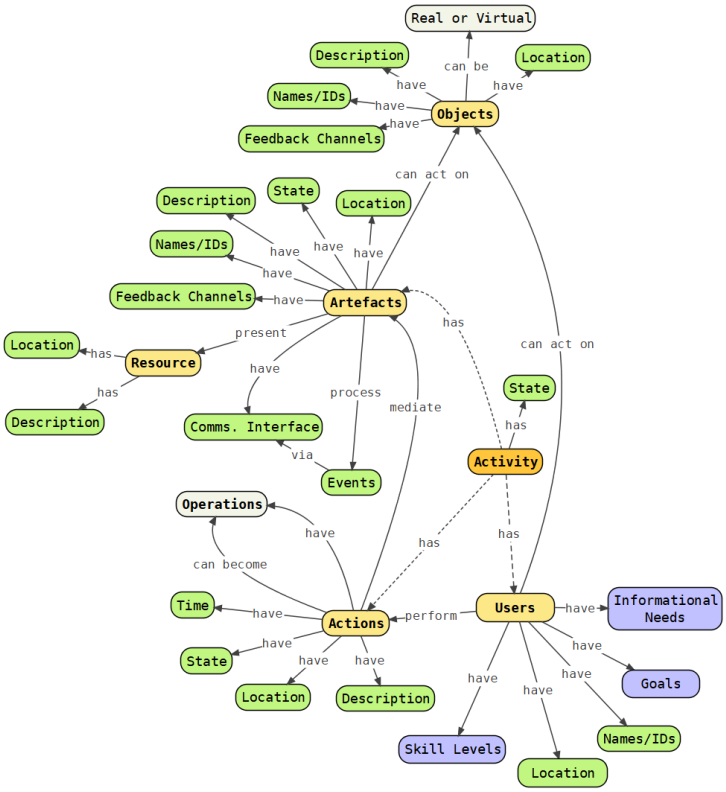

Figure 7. MiWSICx core ontology

In MiWSICx, an *activity* consists of *users*, *artefacts* and *resources*. As noted in section 3.1, the subject, or the user, is the source of an activity, and has motives and goals that are accomplished via actions afforded by resources. The user may rely on different modes of skills, rules and knowledge-based behavior [38], and therefore has an adaptable information need met by multiple devices and resources. An activity supports actions for activity management, for example, starting, suspending and pausing and switching an activity.

An *artefact*, or a *device*, is what mediates user action. It is uniquely identified by its description, name and location. Most digital devices support various modalities and communication interfaces, or *capabilities* through which they can exchange information. An activity can contain more than one such device. Artefacts use resources to support interactivity, and a resource points to objects which needs to be uniquely identifiable and locatable, both physically and/or digitally. Depending on

its properties, an object has a state that can be changed, either via the artefact, or by direct manipulation.

The entity structure in MiWSICx follows a combination of both *composition* and *aggregation* style, as shown in Fig. 8. At the top of the hierarchy sits the activity context entity which aggregates different activities for a user. One user is associated with one activity context per MiWSICx node. An activity context is a persistent entity that can be saved and reloaded when a user logs out and logs back in on a MiWSICx node.

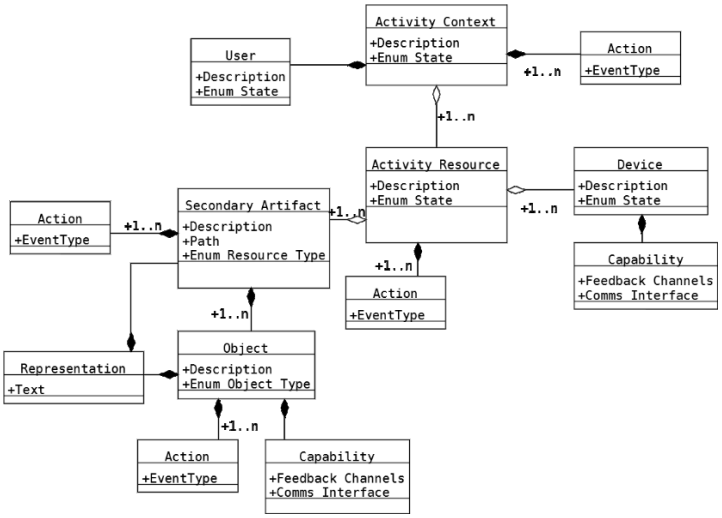

Figure 8. Core entities in MiWSICx

An *activity* consists of resources currently under use along with the devices a user is interacting with. Devices are not directly saved with the activity or the activity context but are added to the activity each time a new device connects. The intention behind this is to stay adaptive to the contextual constraints in an environment.

A *resource* contains one or more objects. An object can, further be represented by a resource, leading to a recursive data structure. Whether or not an object is able to respond to changes to its state is made explicit in its capabilities, which determines if a feedback is available.

The *action* entity appears along with almost every entity, for just like the activity hierarchy in activity theory, different entities support corresponding actions.

An *activity context* supports actions to process user login and switching between different activities. An activity entity supports switching between different resources and their corresponding assignment to devices. Resources support actions that are geared towards providing a service as a change in the resource itself or its underlying objects states, for which the objects also need to support actions themselves.

4.2 Basic Entities and Types

An entity is fully described by a combination of basic entities, as described below:

- *when:* describes a point in time, in the form of a timestamp in combination with a timeout value.
- *where:* describes a location in space, consisting of a reference point, and a relative coordinate. A where object also contains a representation for this point in space, for example in the form of an image.
- *representation:* contains additional information about an entity that is helpful in representing this entity via some modality. A representation consists of either a text description, or a resource.
- *identity:* consists of a name and/or a unique identity number.

In addition, the following types are used to categorize different entities:

- *feedback direction:* differentiates between an input and output modality.
- *feedback modality:* describes different forms of modalities that a device or object supports.
- *resource type:* common types of resources such as Images, 3D objects, files and URLs.
- *object type:* distinguishes between a physical or a virtual object.

4.3 Events

Between different activity components, the abstraction that suitably covers conveying prompts and actions is that of an event. An event is an occurrence- it signals that something has happened in the real world or in some other system [39]. Event attributes are briefly described as follows:

- *id*: represents a unique id that the message is assigned when created. This can serve as a reference for further use if the event processing is not stateless.
- *when*: each event needs a timestamp and an associated timeout interval.
- *event type*: differentiates between event types.
- *payload*: the payload in an event can be constructed out of any specification capable of transmitting objects, such as JSON or XML. MiWSICx uses JSON payloads.
- *source*: refers to the sender of the event.
- *sink*: refers to the target, either in the form of a device, or can describe a path, similar to a REST endpoint.
- *priority*: some events, for example alarms, may carry a higher priority than other messages.

4.4 Actor Model

In section 2.1, the need for adopting a better abstraction for handling concurrency and distribution was highlighted. The conventional approach for modelling such a distributed middleware is by programming multiple threads or processes in the context of an application, where *computation* is separated from *communication.* Thus, a typical distributed application consists of some sort of communication framework, either a broker-based architecture such as Data Distribution Service (DDS), MQTT or ZeroMQ, or a service-based architecture such as SOAP or REST pattern, in conjunction with the core data processing application.

In reality, multi-threading puts serious limitations in the way of achieving its goal [40], the most significant being the need for data invariance, which has three implications for concurrent and/or distributed systems. First, locking threads makes them wait for access to shared resources. Second, such locking is hard to achieve on distributed systems. Third, parent threads delegating work to multiple child threads cannot handle exceptions in child threads, since any exceptions that occur are local to a thread, and the parent is unaware of issues on the child thread. In addition, thread synchronization on multi-core processors is a surprisingly expensive operation, and it becomes worse as a problem is broken down into multiple, smaller, synchronized tasks [41].

For managing dynamic activity contexts, using multiple applications in the form of micro-services and service-oriented architectures is a second option, however, it does not come without its drawbacks. Maintaining interoperability of several different services can be a challenging task, further, the more distributed the services are, the more complex they are to develop and maintain owing to their distributed nature [42].

The *actor model* is proposed as a replacement that mitigates the drawbacks of threads and service-oriented architectures, while at the same time staying distributed, concurrent and resilient. It was proposed in 1973 as a mathematical theory of computation that treats actors as the universal primitives of concurrent digital computation [43], and was developed further by Agha [44]. An *actor* is a computing abstraction, an entity that contains a local, immutable state; does not share this local state with other actors; and is responsible for updating is local state.

The word *actor* emphasizes the design principle of separation of concerns. Each *actor* only performs a single job. Structurally, in an actor model, systems comprise of concurrent, autonomous entities, called *actors* and *messages*. An *actor* requires an immutable *name* or an *address* to send messages to it, and actors communicate exclusively by sending *asynchronous messages* to one another [44], as shown in Fig. 9.

The structural and functional constraints on actors result in the following behavioral properties [45]:

- *Encapsulation*: An Actor cannot access the state of other actors, nor can actors share states. The possibility of race conditions or shared data state corruption is avoided.
- *Fair Scheduling*: Messages are guaranteed to be eventually delivered to destination actors, and consequently, no actors are starved of resources.
- *Location Transparency*: The actual location of an actor has no bearing on its name. An actor could be on the same core, on the same CPU, or on a different node on a network.
- *Mobility*: Actors can be updated and moved across different nodes individually, and an Actor Model can be reconfigured for load balancing and fault tolerance. This is different from typical OOP applications where the entire application must be compiled and deployed.

These properties consequently facilitate the processing of requests and events coming from multiple devices, all of which may belong to the same activity and may compete for resources. Actors handling communication with devices can react to messages, while instantiating and delegating problems to other actors.

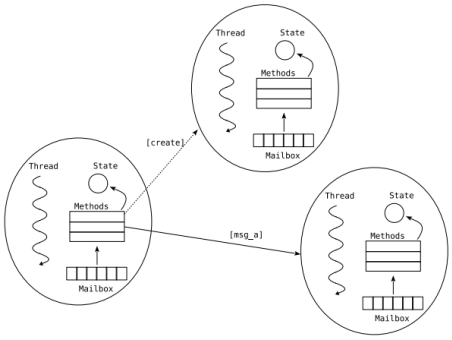

Figure 9. Actor model, taken from [42].

4.5 Actors in MiWSICx

Designing with actors usually means the software must be constructed in accordance with the principle of separation of concerns, thus each actor performs a single task. The following sections elaborate the various actors created in MiWSICx, shown in Fig. 10.

4.5.1 Core Actors

The root or top-level actor in MiWSICx is referred by the same name. Its job is to startup the MiWSICx base system, which consists of a *comms actor*, the *resource manager actor*, the *activity context manager actor* and the *discovery actor*. All the actors are informed of each

other's actor addresses at startup. Upon shutdown, the MiWSICx actor sends an exit message to all the child actors, who are then responsible for shutting down their respective child actors.

A device can join MiWSICx depending on its capability. Camera enabled devices can use QR codes to scan and obtain the IP address, and devices without this capability can communicate with a Bluetooth or NFC sensor. Another way is to use UDP broadcast for exchanging the address of the MiWSICx endpoint, but with several nodes broadcasting in the vicinity the user will have to choose from a list of nodes. Once a device knows the IP Address, it can communicate further by establishing a connection to the TCP server on the Comms Actor. Access to CPPS objects and other MiWSICx nodes in the network is maintained by the *extern comms actor*. Supporting user activities in the form of activity templates, plans and resources is handled by the *resource manager actor*.

Second tier actors may decide to start *child actors*. For example, the *resource manager* can create *service provider actors* that provide data analytics, and the *external comms actor* may start an actor that communicates with the CPPS hardware through ModBus or OPC-UA. etc, allowing the system to configure itself based on the desired needs and capabilities of the background CPPS.

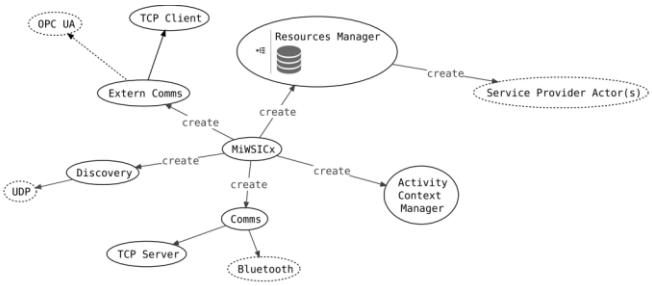

Figure 10. Actors in a MiWSICx node.

4.5.2 Device Handlers

When a device connects to a MiWSICx node, it needs to announce its capabilities to the corresponding *comms actor*, which then creates a *device handler actor* that takes care of all further communication with this device. Once the device disconnects, the corresponding device handler actor is also destroyed.

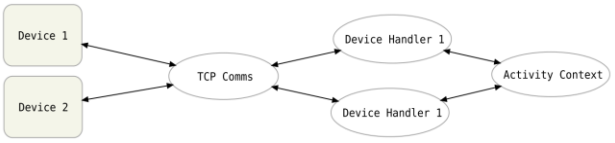

Figure 11. Device handler actors.

Device handlers also act as event filters. For example, when a feedback event is sent to a device handler, it can decide based on its capability if it will forward the message to the device, and if yes, how to best represent this event.

4.5.3 Activity Context Actor

An *activity context* is started when a user logs in on a particular device. First, the activity context manager directs the request to the resource manager, which searches for an existing context for this user. In case no context is found, a new activity context handler is created for the user. Any subsequent device connections with the same user name will be directly added to the same activity context handler.

The *activity context actor* itself is responsible for routing messages within the context of a user activity and contains a composite entity. The state of devices, for example, in an activity context is handled by the device handler actor, and the resources themselves are handled by the resource handler actor.

Once an activity context is created, the user has access to the services and resources available for either restoring previously paused activities or creating new ones. As new users connect, each user is assigned a new activity context handler, and upon logging out, the activity context is saved by the resources manager. Sharing activities means assigning more than one user to a context. Any changes made by one user are visible to the other user as well.

4.5.4 MiWSICx Messaging Protocol

Within MiWSICx, events are transmitted as actor messages, but for transmitting events to and from devices, a messaging protocol is needed. The MiWSICx messaging protocol is adapted from HTTP and is divided into a header and payload. The header contains the event attributes as presented in section 4.3, while the payload contains information regarding the activity at hand.

4.5.5 Deployment

MiWSICx is written in Python and uses "The Thespian Actor Framework" [46]. Python was chosen because of its cross-platform compatibility, its dynamic typing system, and its rich repertoire of stable libraries.

Nonetheless, there are several actor frameworks available in various languages which can be chosen based on technical and functional requirements. MiWSICx as such does not rely on one language or library, allowing it to run on a variety of hardware platforms.

5 Conclusions

The traditional approach to HCI in industrial contexts

as an interface between man and machine needs to be rethought in the form of human activity and how it flows within different places and devices at various times on the manufacturing floor. Problem solving as such cannot be modelled as a predetermined division of roles between the man and the machine; rather, the context of work needs to be rethought as that of activities composed of the human worker, his/her artefacts as devices and informational resources, and the objects that are the recipients of this activity. This shift in perspective allows for design of systems that are adaptable and extensible by the human workers themselves as their own creative problem-solving tools. One approach to system design is in the form of a middleware, and MiWSICx is the first such middleware that implements such a design for supporting work in industrial contexts.

A CPPS consists of a mixture of both networked and non-networked components distributed in a spatial and temporal manner, hence the framework for work support also needs to be distributed. A high level of responsiveness in handling multiple users, activities and devices simultaneously means that an efficient concurrency framework is needed, one that preferably avoids the drawbacks of thread-based approaches. The Actor Model is hence used as the foundation for the development of MiWSICx, which is a generic framework that can be deployed using any underlying Actor implementation. By modelling the constituents of human activity - users, devices, resources and objects, MiWSICx manages activities across various device configurations, which are established via operational and instrumental properties of devices.

References

[1] Broy, M., and Schmidt, A. Challenges in Engineering Cyber-Physical Systems. Computer, 47, 2 (Feb 2014), 70–72.

[2] Pfeiffer, S. (2016). Robots, Industry 4.0 and Humans, or Why Assembly Work Is More than Routine Work. Societies, 6, 16 (2016).

[3] Bainbridge, L. Ironies of Automation. Automatica, 19 (1983), 775–779.

[4] Lucke, D., Constantinescu, C., and Westkämper, E. Smart Factory - A Step Towards the Next Generation of Manufacturing. In Manufacturing Systems and Technologies for the New Frontier (London, 2008), M. Mitsuishi, K. Ueda, and F. Kimura, Eds., Springer London, pp. 115–118.

[5] Valdez, A. C., Brauner, P., Schaar, A. K., Holzinger, A., and Ziefle, M. Reducing Complexity with Simplicity - Usability Methods for Industry 4.0. Proceedings 19th Triennial Congress of the IEA, pp. 9–14.

[6] Spath, D., Ganschar, O., Gerlach, S., Moritz, H., Krause, T., and Schlund, S. Produktionsarbeit der Zukunft - Industrie 4.0. Fraunhofer, 2013.

[7] Gorecky, D., Schmitt, M., Loskyll, M., and Zühlke, D. Human-Machine Interaction in the Industry 4.0 Era. In 2014 12th IEEE International Conference on Industrial Informatics (INDIN) (July 2014), pp. 289–294.

[8] Bettenhausen, K. D., and Kowalewski, S. Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation. VDI/VDE-Gesellschaft Mess-und Automatisierungstechnik (2013), 9–10.

[9] Vogel-Heuser, B., Lee, J., and Leitão, P. Agents Enabling Cyber-Physical Production Systems. at-Automatisierungstechnik 63, 10 (2015), 777–789.

[10] Lucke, D., Constantinescu, C., and Westkämper, E. Smart Factory - A Step towards the Next Generation of Manufacturing. In Manufacturing Systems and Technologies for the New Frontier (London, 2008), M. Mitsuishi, K. Ueda, and F. Kimura, Eds., Springer London, pp. 115–118.

[11] Abramovici, M. Smart Products. In CIRP Encyclopedia of Production Engineering. Springer, 2015, pp. 1–5.

[12] ElMaraghy, W., ElMaraghy, H., Tomiyama, T., and Monostori, L. Complexity in Engineering Design and Manufacturing. CIRP Annals 61, 2 (2012), 793–814.

[13] Li, K., and Wieringa, P. A. Understanding Perceived Complexity in Human Supervisory Control. Cognition, Technology & Work, 2, 2 (May 2000), 75–88.

[14] Guimaraes, T., Martensson, N., Stahre, J., and Igbaria, M. Empirically Testing the Impact of Manufacturing System Complexity on Performance. International Journal of Operations & Production Management 19, 12 (Dec. 1999), 1254–1269.

[15] Rasmussen, J., and Lind, M. Coping with Complexity. HG Stassen (Ed.) (1981).

[16] Liu, P., and Li, Z. Task Complexity: A Review and Conceptualization Framework. International Journal of Industrial Ergonomics, 42, 6 (Nov. 2012), 553–568.

[17] Vakkari, P. Task Complexity, Problem Structure and Information Actions. Information Processing & Management, 35, 6 (Nov. 1999), 819–837.

[18] Sellen, A., and Nicol, A. Building User-Centered On-Line Help. In Readings in Human–Computer Interaction. Elsevier, 1995, pp. 718–723.

[19] Grayling, T. Fear and Loathing of the Help Menu: A Usability Test of Online Help. Technical Communication 45, 2 (1998), 168–179.

[20] Chilana, P. K. Supporting Users After Software Deployment through Selection-Based Crowdsourced Contextual Help. PhD Thesis, 2013.

[21] Brettel, M & Friederichsen, Niklas & Keller, M &

Rosenberg, N. (2014). How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective. International Journal of Science, Engineering and Technology. 8. 37–44.

[22] Ong, S. K., and Nee, A. Y. C. Virtual and Augmented Reality Applications in Manufacturing. Springer Science & Business Media, 2013.

[23] Bardram, J., Bunde-Pedersen, J., and Soegaard, M. Support for Activity-Based Computing in a Personal Computing Operating System. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06), ACM Press, pp. 211–220..

[24] Leont'ev, A. N. Activity, Consciousness, and Personality. Prentice-Hall, Englewood Cliffs, N.J., 1978.

[25] Baerentsen, K. B., and Trettvik, J. An Activity Theory Approach to Affordance. In Proceedings of the Second Nordic Conference on Human-Computer Interaction (NordiCHI'02), ACM Press, pp. 51–60.

[26] Bødker, S. A Human Activity Approach to User Interfaces. Human Computer Interaction, 4, 3 (Sept. 1989), 171–195.

[27] Vygotsky, L. S. Mind in Society: The Development of Higher Psychological Processes. Harvard University Press, 1980.

[28] Wartofsky, M. W. Models: Representation and the Scientific Understanding, 48. Springer Science & Business Media, 2012.

[29] Zamfirescu, C.-B., Pîrvu, B.-C., Schlick, J., and Zühlke, D. Preliminary Insides for an Anthropocentric Cyber-physical Reference Architecture of the Smart Factory. Studies in Informatics and Control, 22, 3 (Sept. 2013), 269–278.

[30] Baerentsen, K. B., and Trettvik, J. An Activity Theory Approach to Affordance. In Proceedings of the Second Nordic Conference on Human-computer Interaction (New York, NY, USA, 2002), NordiCHI '02, ACM, pp. 51–60.

[31] Norman, D. The Design of Everyday Things: Revised and expanded edition. Constellation, 2013.

[32] Wensveen, S. A. G., Djajadiningrat, J. P., and Overbeeke, C. J. Interaction Frogger: A Design Framework to Couple Action and Function through Feedback and Feedforward. In Proceedings of the 2004 Conference on Designing Interactive Systems Processes, Practices, Methods, and Techniques (DIS'04), ACM Press, pp. 177–184.

[33] Bødker, S., and Klokmose, C. N. The Human-Artifact Model: An Activity Theoretical Approach to Artifact Ecologies. Human-Computer Interaction, 26 (2011), 315–371.

[34] Goumopoulos, C., and Kameas, A. Smart Objects as Components of UbiComp Applications. International Journal of Multimedia and Ubiquitous Engineering 4, 3 (2009), 1–20.

[35] Vernier, F., and Nigay, L. A Framework for the Combination and Characterization of Output Modalities. In International Workshop on Design, Specification, and Verification of Interactive Systems (2000), Springer, pp. 35–50.

[36] Bardram, J. E. The Activity-Based Computing Project. In Activity Context Representation (2011).

[37] Moran, T. P., Cozzi, A., and Farrell, S. P. Unified Activity Management: Supporting People in e-Business. Communications of the ACM 48, 12 (2005), 67–70.

[38] Rasmussen, J. Skills, Rules, and Knowledge; Signals, Signs, and Symbols, and other Distinctions in Human Performance Models. IEEE Transactions on Systems, Man, and Cybernetics, 3 (1983), 257–266.

[39] Etzion, O., Niblett, P., and Luckham, D. C. Event Processing in Action. Manning Greenwich, 2011.

[40] Sutter, H., and Larus, J. Software and the Concurrency Revolution. Queue, 3, 7 (Sept. 2005), 54–62.

[41] Latoschik, M. E., and Tramberend, H. Simulator X: A Scalable and Concurrent Architecture for Intelligent Realtime Interactive Systems. IEEE, pp. 171–174.

[42] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. Microservices: Yesterday, Today, and Tomorrow. In Present and Ulterior Software Engineering. Springer, 2017, pp. 195–216.

[43] Hewitt, C., Bishop, P., and Steiger, R. A Universal Modular Actor Formalism for Artificial Intelligence. In Proceedings of the International Joint Conference on Artificial Intelligence, 1973, pp. 235–245.

[44] Agha, G. A. Actors: A Model of Concurrent Computation in Distributed Systems. Tech. Report, Massachusetts Institute of Technology, Cambridge Artificial Intelligence Lab, 1985.

[45] Karmani, R. K., Shali, A., and Agha, G. Actor Frameworks for the JVM Platform: A Comparative Analysis. In Proceedings of the 7th International Conference on Principles and Practice of Programming in Java (PPPJ'09), ACM Press, pp. 11–20.

[46] Quick, K. Python Actors. Retrieved from https://thespianpy.com/