Journal of Computational Design and Engineering, 2021, 8(1), 428–451

doi: 10.1093/jcde/qwaa088
Journal homepage: www.jcde.org
Advance Access Publication Date: 26 December 2020

RESEARCH ARTICLE

Middleware for providing activity-driven assistance in cyber-physical production systems☆

Hitesh Dhiman[1,*] and Carsten Röcker[1,2]

[1]TH OWL University of Applied Sciences and Arts, Department of Electrical Engineering and Computer Science, 32657 Lemgo, Germany and [2]Fraunhofer IOSB-INA, 32657 Lemgo, Germany

☆Selected paper from the International Congress and Conferences on Computational Design and Engineering 2019, 7-10 July 2019, Penang, Malaysia.
*Corresponding author. E-mail: hitesh.dhiman@th-owl.de

Abstract

Assistance is becoming increasingly relevant in carrying out industrial work in the context of cyber-physical production systems (CPPSs) and Industry 4.0. While assistance in a single task via a single interaction modality has been explored previously, crossdevice interaction could improve the quality of assistance, especially given the concurrent and distributed nature of work in CPPSs. In this paper, we present the theoretical foundations and implementation of MiWSICx (*Middleware for Work Support in Industrial Contexts*), a middleware that showcases how multiple interactive computing devices such as tablets, smartphones, augmented/virtual reality glasses, and wearables could be combined to provide crossdevice industrial assistance. Based on *activity theory*, MiWSICx models human work as *activities* combining multiple *users*, *artifacts*, and cyber-physical *objects*. MiWSICx is developed using the *actor model* for deployment on a variety of hardware alongside a CPPS to provide multiuser, crossdevice, multiactivity assistance.

Keywords: human–technology interaction; human–computer interaction; crossdevice interaction; cyber-physical systems; assistance; smart factory; middleware; actor model; information system design; industry 4.0

1 Introduction

The production environment of the future is marked by an increasing use of cyber-physical systems (CPSs), in which physical processes and the means of their control (sensors, actuators, and processors) are both interconnected and distributed. In manufacturing, the term cyber-physical production systems (CPPSs) or Industry 4.0 (Broy & Schmidt, 2014) is used. It is postulated that the advancement of communication technologies and machine-learning algorithms could automate many human activities in manufacturing, resulting in a Smart-CPS (Tepjit, Horváth, & Rusák, 2019). Nonetheless, the history of automation shows that while introducing new technologies may compensate for human weakness, these often give rise to new strengths and weaknesses in often unanticipated ways (Strauch, 2018), rendering human involvement unavoidable, even necessary. Expectedly, Becker and Stern (2016) predict the following five distinct qualities of future production work:

- Humans will be absolutely necessary in the factories of the future.
- New tasks will be more complex.
- New tasks will be intensely connected to computational devices.
- Easy and repetitive tasks will be automated.
- Unique human abilities will play a more significant role in human task design.

Received: 31 January 2020; **Revised:** 20 November 2020; **Accepted:** 20 November 2020

Computational devices can be used in a dual manner – as control and communication technologies, and also as multiple interactive devices and modes of interaction to provide humans the tools to operate, maintain, and optimize complex CPPSs. In other words, the vision is to elevate the human operator's role to that of an *augmented operator*, or *operator 4.0* (Romero, Stahre, & Taisch, 2020), combining the best of both humans and machines and achieving a human–machine symbiosis (Scafá, Marconi, & Germani, 2020). One approach to achieve this vision is by assisting operators in their activities.

1.1 The context of assistance – two scenarios

Assistance, understood as an act of aiding or helping someone achieve a goal, consists of two entities, that which receives assistance, and that which provides it. In everyday life, when assisting someone, one has to be not only aware of their desired goal, but also mindful of their current context. Similarly, to develop system that assists an operator in a cyber-physical context, an understanding of the operator's need as a combination of their context of work and the underlying CPS is required; the contemporary approach to assistance ignores this fact, as illustrated via two scenarios below.

1.1.1 Scenario 1: assembly assistance

Employee X works at a projection-based assembly station attached to a manufacturing line. X receives a new assembly order, at which point the assembly system guides X through the assembly process. Suddenly, the assembly line stops, and X sees the status indicated on a stack light. X wishes to report the incident to the supervisor, but the mounted screen on the assembly system offers no such option. X then uses a tablet that has a maintenance application installed on it and follows a standard maintenance procedure that offers step-by-step guidance. X goes through the checklist and finds no faults. X waits until the incident escalates to the supervisor.

Scenario 1 portrays the contemporary state of assistance. Most assistance applications today favor only a single activity, providing preconfigured instructions on dedicated devices. A review of industry 4.0 literature in the field of human–computer interaction (HCI) reveals that development of assistance applications has until now focused on single device interaction, usually coupled to a single workstation or a task (Ong & Nee, 2013; Sand, Büttner, Paelke, & Röcker, 2016; Dhiman, Martinez, Paelke, & Röcker, 2018; Dhiman & Röcker, 2019; Dhiman, Büttner, & Röcker, 2019). Crossdevice and activity-driven assistance is a field yet to be explored in the industrial domain.

1.1.2 Scenario 2: activity assistance

Employee Y is assembling an order at a projection based assembly station, where the tablet on his station is at the moment running an assembly application. Y is alerted to a fault on the manufacturing line on the smartwatch Y is wearing, which occurs only when events of certain severity are reported. The smartwatch also reports the location of the line and a QR code to allow access to the machine where the fault may have occurred, which Y transfers to the tablet. Y goes to the machine and scans the QR code that activates a troubleshooting activity for the machine, and uses a pair of mixed reality (MR) glasses that show him the location of various sensors on the machine and their current readings. Besides, Y can also see "augmented" notes made by other operators about points to look out for in the form of textual information on the tablet and as associated 3D locations on the MR glasses, local minutiae that cannot be found in the generic troubleshooting guide. Y fixes the issue as it occurs when this particular product configuration is manufactured, and writes an augmented note for future knowledge. After resolving the issue, Y takes off the glasses, goes back to the assembly station with the tablet, switches back to the assembly application, and resumes work on assembling the product.

Scenario 2 is reimagined as a crossdevice, multiuser scenario where assistance adapts to the users' information needs and respects the situational view of problem solving. It takes an activity-centric view of work, where activities overlap, and sometimes, override one another. Further, it also acknowledges the need to capture the evolving nature of human understanding and knowledge at work and the fact that activities on the shop floor can span across many machines. The devices here play a pivotal role in helping the user switch between different modes of work, each involving inputs from different devices and possibly, other users as well. In short, scenario 2 describes a situation in which assistance is *activity-driven*:

- it presents relevant information through a combination of interactive interfaces and devices most suited to the nature of underlying cyber-physical entities, and
- it can be adapted to the flow of work at the workplace.

In order to realize the second scenario of activity-driven assistance in a CPPS context, an intermediary is needed, a system that can

- communicate with the different, interconnected low-level system components, and simultaneously
- manage crossdevice interaction as operators initiate, stop, and switch between activities.

The most commonly used term to define such a solution is *middleware*. In this paper, we elaborate on the design and prototype implementation of such an intermediary – **MiWSICx** (*Middleware for Work Support in Industrial Contexts*). MiWSICx is a middleware solution that mediates the interaction across industrial systems and multiple devices and users to support them in both informationally and spatially fluid manner.

The rest of the paper proceeds as follows: Section 2 describes how the nature of work in a CPPS also affects the design of interactive solutions for operators, leading to requirement formulation. Section 3 discusses the role of activity theory (AT) in providing a basis for comprehending these requirements, and presents the fundamentals of AT to derive a framework that applies it in the CPPS domain. Further, in Sections 4 and 5 this framework is applied to create an ontology and metamodel for MiWSICx, and the design and deployment of MiWSICx to support CPPS activities are demonstrated. Section 6 elaborates on some of the unresolved issues and sketches themes for further exploration, and Section 7 concludes this paper.

2 Nature of Work in a CPPS

Industrial control systems, until now, have been developed to realize distributed, real-time system control, and long-term stability. Manufacturing and process control applications utilize several independent industrial communication technologies (Modbus, Ethernet, and Profinet to name a few) and the possibilities of human–machine interaction have also been traditionally tightly coupled to these systems, in the form of dedicated industrial human–machine interfaces (HMIs) located on site or in control rooms that provide an overview of the state of the system. As automation increases, the idea of industry 4.0 envisions a future consisting of *smart factories* as opposed to *deserted factories* (Spath *et al.*, 2013). According to Gorecky, Schmitt, Loskyll, and Zühlke (2014), in smart factories human workers are seen “as the most flexible entity in CPPSs,” as they will be “faced with a large variety of jobs ranging from specification and monitoring to verification of production strategies.” In the following subsections, the link between technical qualities of future production and its impact on human work is explored.

2.1 Scalability, concurrency, and operational flexibility

Moving beyond the principles of *cellular manufacturing* and *group technology*, models of manufacturing in industry 4.0 propose distributed, flexible, scalable, and hence complex production environments. In such smart factories (Lucke, Constantinescu, & Westkämper, 2008), on-demand and completely customizable smart products (Abramovici, 2015) can be manufactured on the same manufacturing lines. Based on their capabilities, components can be arranged in many configurations: sequential, parallel, or hierarchical in the form of *distributed services* (Bettenhausen & Kowalewski, 2013) or *agent-based* systems (Vogel-Heuser, Lee, & Leitão, 2015).

While technical flexibility may imply that technology can be arranged and made to operate in different modes and across various hierarchical levels, it does not naturally follow that technology itself is capable of handling the variability – it will be passed on to the human operators in the form of task complexity.

2.2 Increase in human task complexity

ElMaraghy, ElMaraghy, Tomiyama, and Monostori (2012) provide a comprehensive overview of the term and its manifestation in manufacturing, assembly, and enterprise. According to the authors, a system could be complex in *functional*, *structural*, *spatial*, and *temporal* domains. For instance, the structurally complexity in the design of a product is guided by the functional complexity required. Spatial complexity refers to the distribution of components in different locations, while temporal complexity refers to how events in time affect system behavior and lead to complex behavior. In the future workplace, human operators will have to confront all four aspects of complexity in a CPPS. Tasks will become increasingly distributed and concurrent, requiring information about the structure and function of the underlying system from a multitude of sources, increasing cognitive demands and the need for both explicit and tacit knowledge (Rasmussen & Lind, 1981; Byström & Järvelin, 1995; Vakkari, 1999; Becker & Stern, 2016).

The use of HMIs in the manufacturing industry has been shown to reduce the negative effect of system complexity for human operators (Guimaraes, Martensson, Stahre, & Igbaria, 1999). A combination of various interaction possibilities such as tablets, head-mounted displays, AR systems, and wearable technology could reduce the cognitive complexity of tasks in an Industry 4.0 workplace (Lucke *et al.*, 2008; Valdez, Brauner, Schaar, Holzinger, & Ziefle, 2015), but how this combination in an industrial context could be achieved remains unexplored.

2.3 Multidevice versus crossdevice interaction

We could use multiple interactive devices to carry out tasks in two ways: simultaneous or sequential use (Brudy *et al.*, 2019). In simultaneous use, single/multiple activities are distributed on multiple devices and executed in parallel. In sequential use, single/multiple activities are carried out in sequence on different devices. Whereas *multidevice* interaction could refer to either scenario, *crossdevice* interaction concerns the coordinated, fluid use of multiple devices by simultaneously or sequentially distributing interactive elements across these devices. The distribution strategy can be logical, spatial, or temporal (Brudy *et al.*, 2019).

Since multitasking on multiple devices has been shown to reduce performance and increase strain, indiscriminate use of multiple interactive devices could be detrimental to occupational health and safety (Paridon & Kaufmann, 2010). Consequently, a lot of research effort in recent years has been devoted to constructing crossdevice workspaces, which comes with several challenges (Dearman & Pierce, 2008; Santosa & Wigdor, 2013; Jokela, Ojala, & Olsson, 2015). First, devices vary in their modalities, and while different modality combinations could be used to reduce cognitive overload (Elting, Zwickel, & Malaka, 2002), the best suitable device for each task element has to be chosen. Second, since most interactive devices nowadays support multiple applications, a related issue involves switching among activities using the same device ecosystem depending on user needs, and maintaining this configuration over sessions in the background. Third, with use, data tend to spread across different devices and cloud platforms, resulting in fragmentation. Further, any multidevice ecosystem also needs to cater for combinations of individual or collaborative use of devices (Srensen, Raptis, Kjeldskov, & Skov, 2014): *one-user-one-artifact*, *one-user-many-artifacts*, *many-users-one-artifact*, and *many-users-many-artifacts*.

Therefore, designing applications for crossdevice interaction requires an approach to handle challenges at different levels. First, this approach needs to tackle the aspects of *distributed interaction*: parallel versus sequential use, specialized versus redundant use, and individual versus collaborative use. Second, it needs to address the *maintenance of such a configuration* of devices based on spatial, logical, and temporal strategies, and lastly, it has to *manage sources of information* in the background. In the next section, we describe how AT could inform this approach.

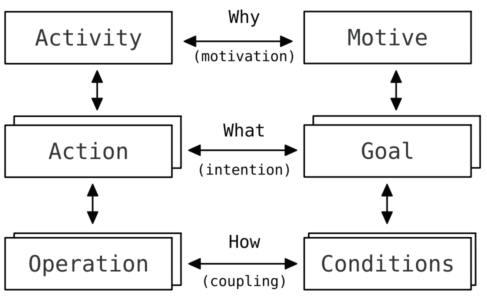


Figure 1: Activity hierarchy, taken from Leont'ev (1978).

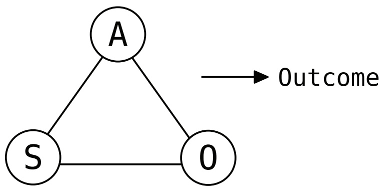


Figure 2: The mediated action triad, as conceptualized by Vygotsky (1980). S, A, and O denote the *subject*, *artifact*, and *object*, respectively.

3 Comprehending Human Work Practice in a CPPS

3.1 Activity theory

Initially developed as a theory of psychology, AT shares some similarities with other goal-oriented approaches to human psychology, such as *action regulation theory* (Zacher & Frese, 2018). While the latter has been applied in the field of *job and work design*, AT has informed research and design in the HCI domain.

AT is a socio-cultural framework that offers an analysis of human behavior as the realization of human motives and goals as *activities*. In the model developed by Leontiev (Leont'ev, 1978), an activity is what links any subject, human or nonhuman, to artifacts in the world in which this subject exists. Leontiev proposes a three-level hierarchy to describe human activity, shown in Fig. 1. At the first level, an activity accomplishes a *motive* by reflecting on an *object*. For instance, a meeting at the office could be a motive for going to the office. At the second level, *actions* are carried out to realize conscious *goals*. In our example, that corresponds to the action of travelling to work, and in some cases, consciously deciding on a means of transport to do so. The object at this level would be the means of transport. At the third and final level, actions are accomplished by means of *operations* that are *internalized* patterns of behavior acquired through learning or social interactions (Baerentsen & Trettvik, 2002). For example, if driving a car, an experienced driver is able to operate the steering wheel and shift gears without them being the focus of attention. Operations can exist only within the structure and condition of actions; for example, the car's condition itself determines how well a driver is able to drive it.

An activity can be differentiated from another only when it is intended toward a different object, even though some actions may be common to both activities. In the previous example, the motive for *going to the supermarket* differs from *going to work*; however, as long as the second level object (car) and the conditions (route, weather, etc.) stay the same, similar actions and operations would be involved.

An essential distinction between AT and other HCI theories is that AT tries to explain how humans transfer between the different levels of activity via *mediation*. Vygotsky (1980) introduced a tool or an artifact as a *mediator* of the interaction between the subject and the object in an asymmetrical relationship (Fig. 2). Most human interaction with objects in the environment rarely takes place without the use of physical and cognitive tools – the process of learning itself involves mastering the use of tools to achieve goals, be it solving a mathematical problem or preparing a meal.

In a complex activity system, numerous varieties of mediating artifacts may be involved. Wartofsky (2012) proposes a three-tiered hierarchy of artifacts. *Primary artifacts* are the most apparent in everyday operations, for example, pens, keyboards, etc. *Secondary artifacts* are representations of tools, as well as plans, explanatory models, and hypotheses. *Tertiary tools* are mobilized at the most abstract level of an activity system to comprehend it and to shape its course. In Fig. 3, we use the terms *why*, *how/what*, and *which* to denote the categories of mediating artifacts. "Why" artifacts represent externalized reasons to engage in activities, for example maintenance schedules. "What" artifacts represent the statement of goals and "how" artifacts show how to achieve them. The final category is "which," pointing to the physical artifacts used to achieve goals. The use of the term "how" at this level is not needed, since operations are internalized in the subject.

Nowadays, the process of digitalization is transferring many forms of physical artifacts, such as documents, into the digital domain; hence, all digitalized why/how/what artifacts are represented by an interactive "which" artifact (for example, desktops, tablets, wearable devices, etc.). This relationship is also shown in Fig. 3.

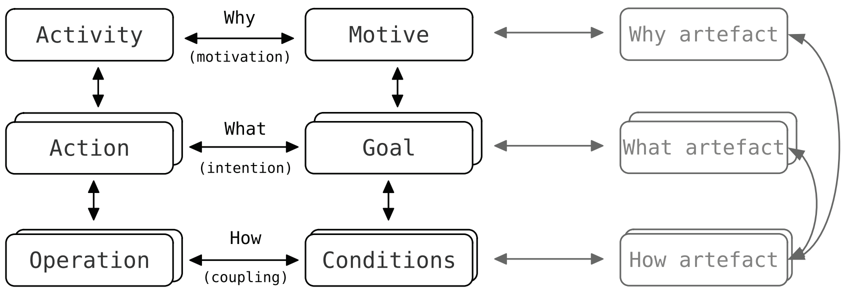


Figure 3: The mediating artifacts *why*, *what*, and *how*, and *which* correspond to their activity counterparts.

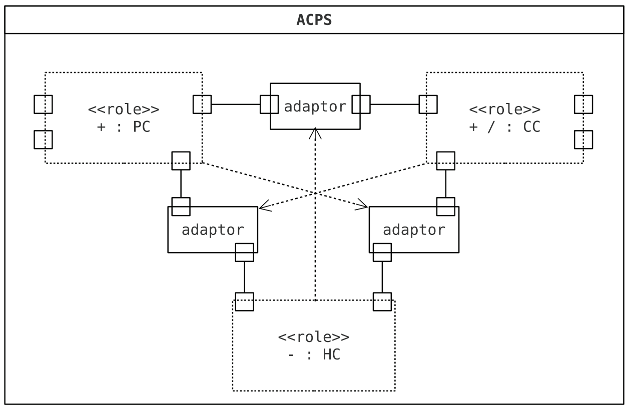


Figure 4: Anthropocentric CPS (Zamfirescu *et al.*, 2013).

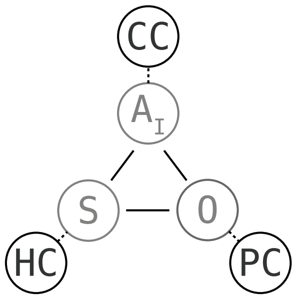


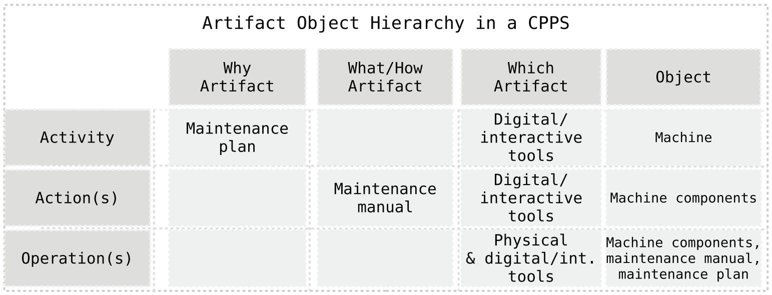
Figure 5: Parallels between the entities in a CPPS and AT.

3.2 An activity-centric CPPS

While the notion of activity-based computing (Norman, 1986) and activity-centric computing (Bardram, 2011) has long entered the desktop computing environment (KDE.org, 2018), the potential of AT remains relatively unexplored in the industrial domain. While we are unsure of the reason behind this state of affairs, we postulate that this may have something to do with the socioeconomic dynamics of research in the HCI domain that has traditionally focused more on the consumer segment. In this section, we establish the link between AT and CPPS.

We start with a human-centric (or anthropomorphic) CPS architecture for a smart factory proposed by Zamfirescu, PâRvu, Schlick, and ZûHlke (2013), shown in Fig. 4. In this architecture, a CPS is divided into three components: the *physical component* (PC), the *cyber/computational component* (CC), and the *human component* (HC). Each of these components is connected outside the CPS to a specific physical, computational, and social dimension. *Adaptors* transfer information between pairs of these components. The model identifies the components of a CPPS and is appreciative of the relationships between them, but does not commit to any particular computational or interactive model for putting the framework to use. In this model, both the PC and CC can support interactions, the former through "special displays" and the latter via "classic HCI devices," both of which are forms of adaptors.

On a closer look, the parallels between the models shown in Figs 4 and 2 are observable. Borrowing the terminology from AT, the *intentional* object is the CPS, while the *coupling*, or mediation, is supported by the adapters. This mediating nature of the *adaptors* becomes apparent 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. 5. In an activity-centric CPPS, cyber-PCs are the objects of all control applications in a CPPS; they are



Artifact Object Hierarchy in a CPPS

	Why Artifact	What/How Artifact	Which Artifact	Object
Activity	Maintenance plan		Digital/ interactive tools	Machine
Action(s)		Maintenance manual	Digital/ interactive tools	Machine components
Operation(s)			Physical & digital/int. tools	Machine components, maintenance manual, maintenance plan

Figure 6: Example of an artifact object hierarchy.

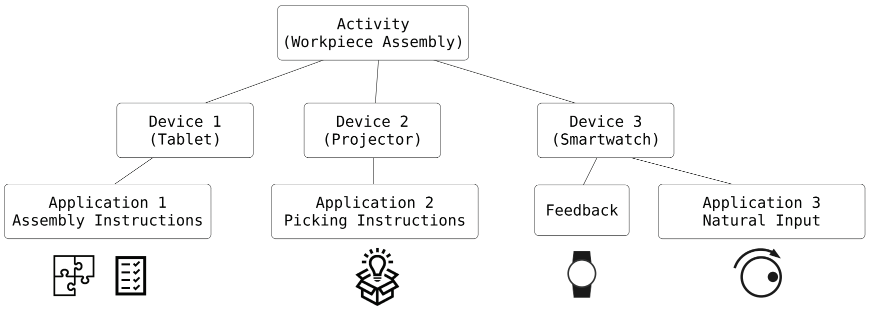

Activity (Workpiece Assembly)
Device 1 (Tablet)
Device 2 (Projector)
Device 3 (Smartwatch)
Application 1 Assembly Instructions
Application 2 Picking Instructions
Feedback
Application 3 Natural Input

Figure 7: Example of a concrete activity.

information producers and carriers that support interaction, mediated by interactive artifacts. Using this model as an entry point, the physical and informational relationships between these entities can be better understood.

As the granularity of the activity changes from activity to actions and operations, so does the granularity of the objects and artifacts. For example, to carry out the maintenance of machine, at an activity level, the object is the machine itself. To have successfully carried out the maintenance means that the machine is in running order. The “why” artifact at this level is the maintenance plan that helps initiate the activity. At the second level, actions represent the work to be done, for example cleaning a part, checking the physical or electrical condition of parts or replacing them, etc. The “what and how” artifact under use here is the maintenance manual, along with cognitive tools such as interactive devices. At the final level, no “how” artifact exists, since operations are internalized; they are not described but assumed to be carried out through a “which” artifact, for example unscrewing a part or a subpart. The table in Fig. 6 summarizes these levels.

In a CPPS scenario, we use the word *resource* to denote the digital representation of *why/what/how* artifacts in the form of uniquely identifiable and locatable object representations as well as the plans to manipulate, maintain, and modify them. A *what* resource could be a *digital twin* (Uhlemann, Lehmann, & Steinhilper, 2017) to a CPPS object in the industry 4.0 context, whose related *how* artifacts provide additional information to the operator to carry out the actions supported by this object via an interactive artifact. A change in the state and properties of the associated CPPS object would be reflected in the what and how resources. *Why* resources are represented by plans and events that form the motivation behind carrying out an activity. The actions that are allowed depend on a combination of the *capability* of the mediating device and the properties of the resource – for instance, a tablet can allow text manipulation; doing so with a smartwatch would be cumbersome.

3.3 Activity contexts

In a crossdevice scenario, each interactive artifact may be delegated a different role, that is, to act as a mediator for different artifacts, feedback, or action, depending on its instrumental (what it helps the user achieve) and operational (how it helps the user achieve it) capabilities (Bødker & Klokmose, 2011).

Figure 7 shows a more practical way of looking at an activity. The motivation of performing actions here is the completion of a workpiece assembly. Interactive artifacts represent how/what artifacts, or mediate actions, or both, each playing to its own strength – the tablet is best suited for representing a how/what artifact, i.e., instructions, while *in situ* projection draws worker’s attention to the objects of these instructions, that is, bins on the workstation. A smartwatch provides immediate, haptic feedback, suited to a noisy environment to relay *why* events, along with natural interaction capabilities.

Viewed from an abstract perspective, an activity context could be viewed as the distribution of interaction with resources among different devices as illustrated by Fig. 8. Each device, therefore, due to its instrumental and operational capabilities, affords different actions under the same activity. Individual what/why resources are assigned to a device, but could also be shared between some of the devices. The first approach prefers *specialization*, while the second is more *redundant*, both in physical (Vernier & Nigay, 2000) and informational aspects (Moore, Chrysanthakopoulos, & Nielsen, 2007).

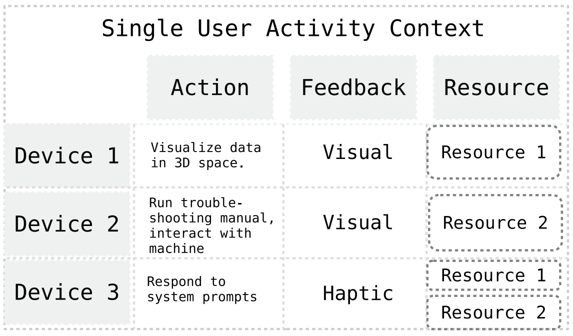

Single User Activity Context
Action
Feedback
Resource
Device 1
Visualize data in 3D space.
Visual
Resource 1
Device 2
Run trouble-shooting manual, interact with machine
Visual
Resource 2
Device 3
Respond to system prompts
Haptic
Resource 1
Resource 2

Figure 8: Single-user, crossdevice configuration.

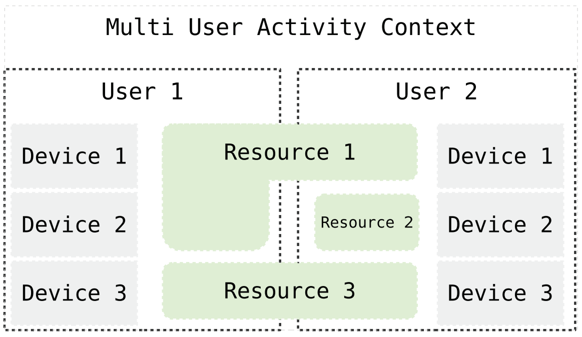


Figure 9: Multiuser, crossdevice configuration.

The same concept could be extended to collaborative activities, where users, by definition, work together toward a common objective. It does not imply that the user needs to be involved in the same activity, only that the object of their activities is common. For example, two workers may carry out the maintenance of the same machine, but carry out different tasks. In the simplest case, the users only share the object, and have their own individual interactive and resource artifacts. In the second, more complex case, they may share the resource artifacts, as shown in Fig. 9.

Until this point, AT has been utilized to create an abstract representation of an activity context consisting of a combination of *users*, *artifacts*, *activities*, *goals*, and *actions*. In the following section, an ontology of the activity context is derived to implement the data model for the middleware (MiWSICx), following which a suitable technical framework is chosen to create a software architecture that can meet the technical requirements.

3.4 Summary and requirement formulation

Based on the discussions in the previous sections, the following technical and functional requirements were derived for MiWSICx. Technically, the middleware needs to

- be distributed;
- support concurrency;
- be scalable;
- reduce implementation complexity.

As for functional requirements, an MiWSICx will have to

- communicate with various artifacts and objects over different channels;
- provide access to resources and services;
- support activity-centric computing for multiple users and on various devices.

Based on these requirements, the following sections elaborate on the design of MiWSICx.

4 MiWSICx: Design

4.1 Ontology

Figure 10 shows the activity ontology in MiWSICx. (Bardram, 2011) and Moran, Cozzi, and Farrell (2005) have also developed similar ontologies, but our ontology differs in two aspects. First, the concept of an *object* is realized as a networked CPPS object, and second, the concept of a *resource* is introduced as the facilitator of this interaction.

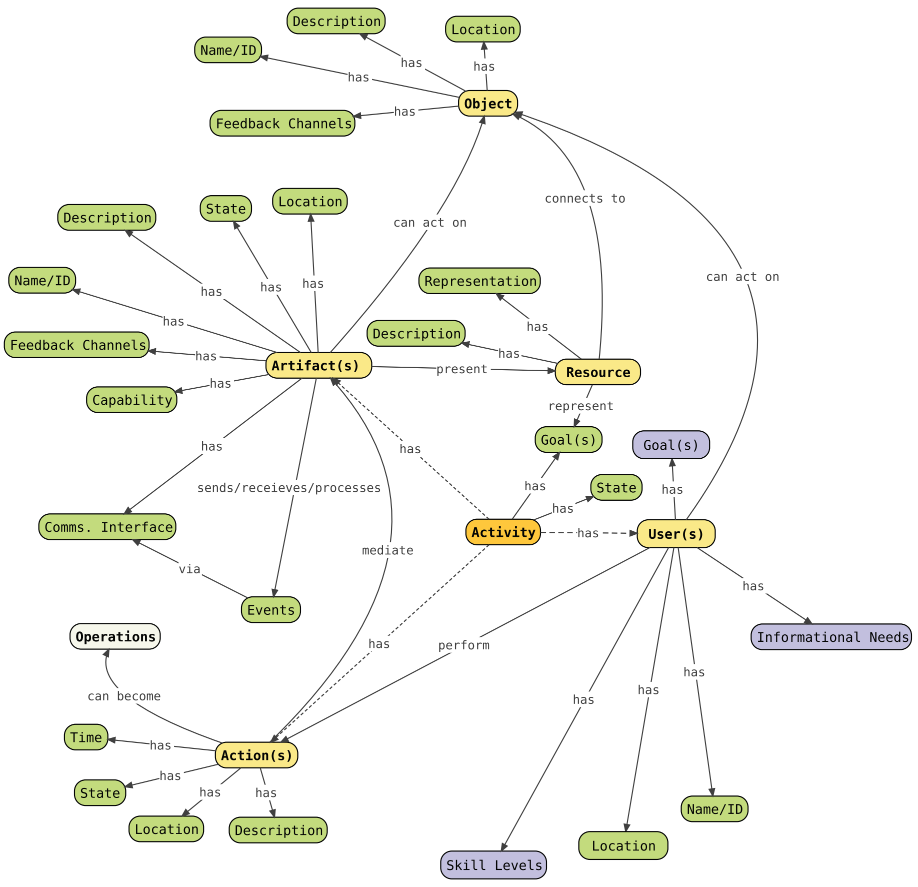


Figure 10: Entities in an activity-centric CPPS. Green elements indicate quantitative components, while blue elements indicate qualitative components.

Figure 11 shows the derived entity structure for MiWSICx. An *activity context* entity aggregates different activities for a user, whereby a *user* is associated with an activity context, defined as a persistent entity that can be saved and reloaded when a user leaves and reenters an activity context.

An *artifact*, or a *device*, is what mediates user action, and is uniquely identifiable by its description, name, and location. Most digital devices support various modalities and communication interfaces, termed *capabilities* through which they can exchange information. An activity can contain a combination of such devices.

An *activity* consists of *resources* currently under use along with the *devices* a user is interacting with. To adapt to the contextual constraints in an environment, devices are not persistently saved with the activity or the activity context, but added to the activity each time a new device connects.

Further, the term *action* encapsulates the corresponding changes of state that an abstract entity affords. For instance, an activity context supports actions to process user login and switching between different activities, whereas an activity entity supports change in activity states and corresponding assignment to devices. Resources support actions that provide a service as a change in the resource itself or its underlying objects states, for which the objects also need to support actions themselves. Therefore, in conjunction with the action entity, a *state* too is contained in almost every entity. The state here signifies the semantic representation of an entity's state brought about by an action.

4.2 Description entities and types

The description object encapsulates basic entities through whose combination an entity can be fully described, as shown in Fig. 12:

- *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.

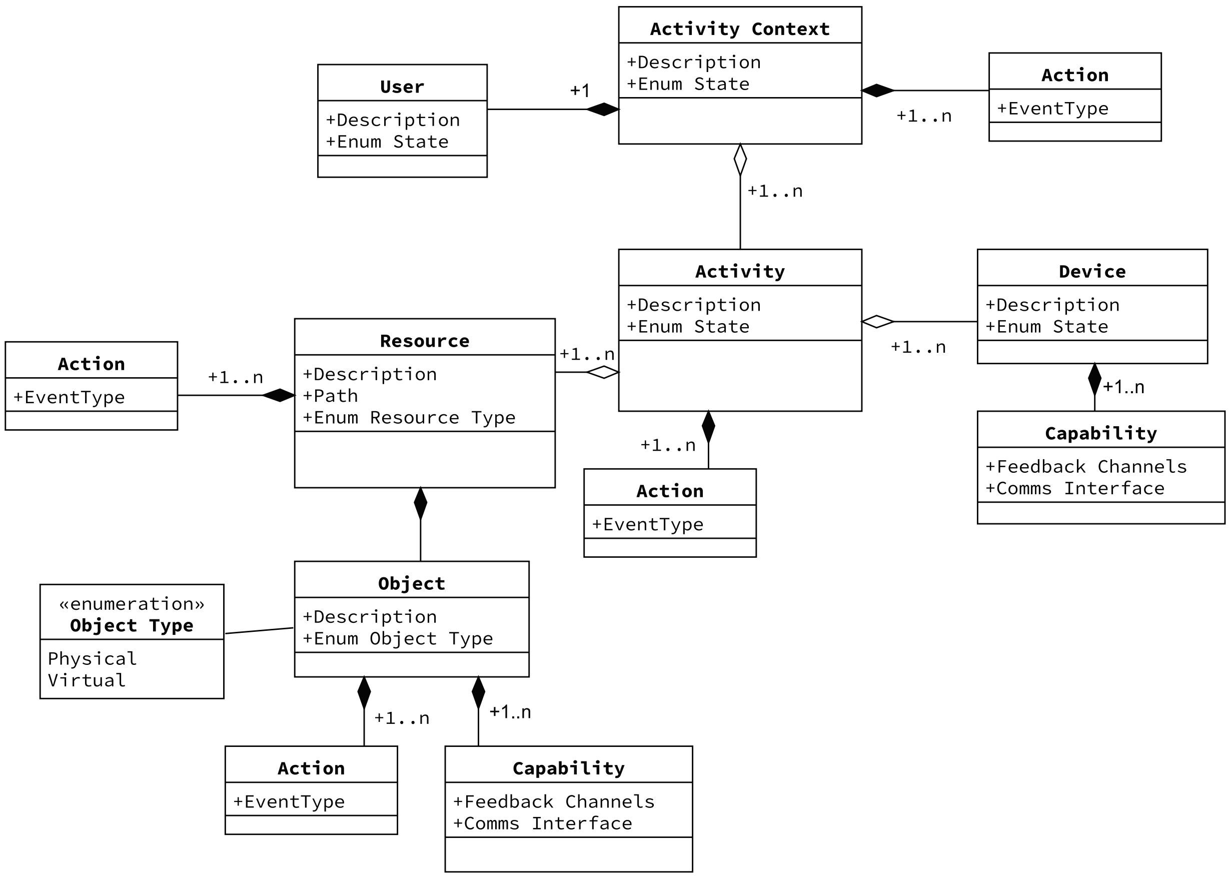


Figure 11: Core entities in MiWSICx.

5 MiWSICx: Implementation

Section 3.4 listed the following technical requirements for MiWSICx: concurrent behavior, scalability, and ease of deployment. In this section, the different approaches available for implementing such a middleware solution are compared.

5.1 Services, agents, or actors?

In choosing a foundation for implementing MiWSICx, it was necessary not only to take into consideration the management of activities and the associated user interaction, but also the possibility that these activities may involve distributed computational tasks such as image processing, data analysis, and machine learning in the background. Therefore, both asynchronous messaging and computational concurrency would have to be managed such that a deployment could manage multiple users, activities, and associated background processes at the same time.

The conventional approach would involve a decoupling of communication from computation, where communication would be handled either by a shared-data-space/broker/publish–subscribe model such as data distribution service (Pardo-Castellote, 2003), MQTT (Banks & Gupta, 2014), and ZeroMQ (Hintjens, 2013) in conjunction with a client–server model, implemented as a service-oriented architecture (SOA).

While this decoupling would more than adequately support communication between systems and user devices, the problem of distributed computing would still remain unsolved at an individual system level. Computational concurrency would still have to be managed by each application at a local level, and applications would have to use a different interprocess communication protocol to exchange data between the local (within the application itself as well as within applications on the same system) and global (between systems) levels. This *lack of transparency* between the global and local scale means added complexity as different mechanisms for managing communication and concurrency would have to be maintained (function calls, threads, processes, interprocess communication, and messages between systems using a broker), which goes against our design aim to exert minimal effort in maintaining and deploying the system.

Searching further for a suitable foundation for MiWSICx, two approaches emerged as promising candidates – *multiagent systems* and *actor model*. At a broad level, both frameworks share some similarities. They both offer a means for concurrent computation by implementing communicating autonomous entities. The roots of both these concepts can be traced back to the 70s; however, they were both designed to address different requirements. The actor model is a mathematical theory of computation that treats *actors* as the universal primitives of concurrent digital computation (Hewitt, Bishop, & Steiger, 1973). The actor model was formulated to address some of the core issues of computing, namely, *shared, mutable state* (shared resources whose state changes with time),

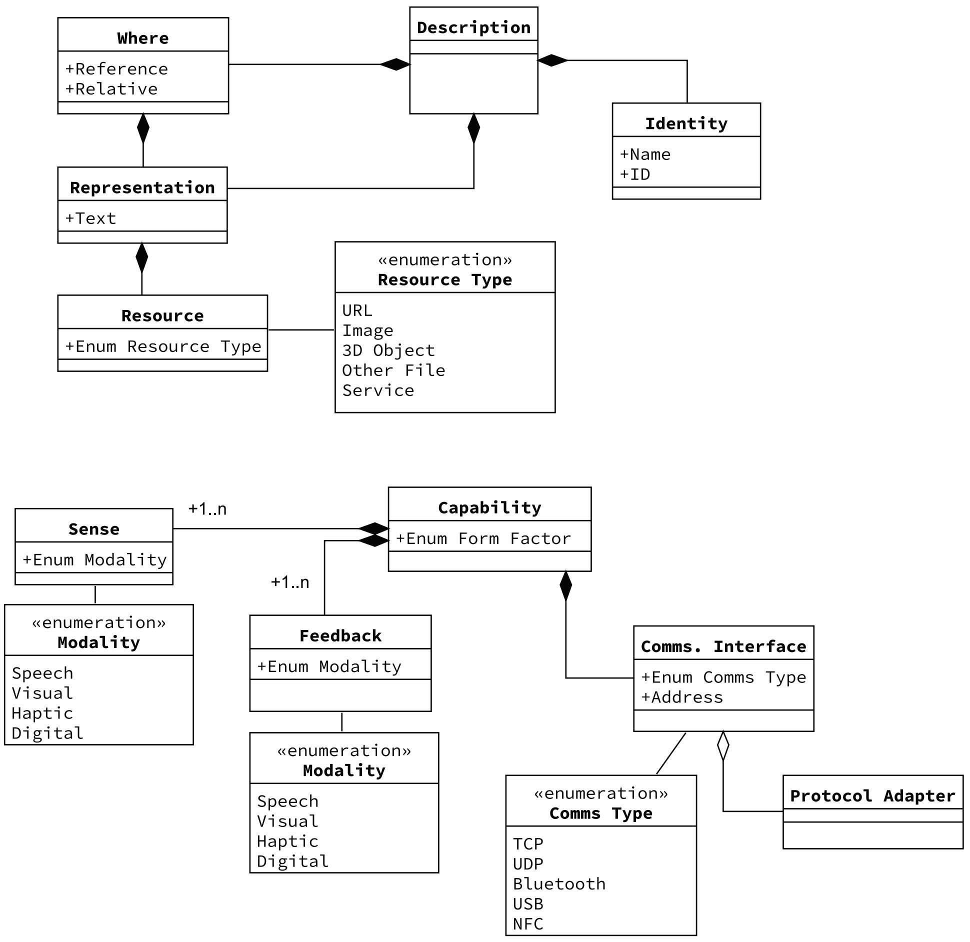


Figure 12: Description entities in MiWSICx.

concurrency (that the activities could be carried out in parallel if possible), and *reconfigurability* (new objects are created and can be communicated with after their creation; Agha, 1986). Multiagent systems, on the other hand, were conceived to harness ubiquity and interconnection to delegate tasks to intelligent agents that would be capable of *independent, autonomous action* and achieve *particular goals* (Wooldridge, 2009). Over the years, multiagent systems have enjoyed widespread adoption in the industrial sector, whereas the actor model has only gained popularity more recently in the web services sector.

In our view, the multiagent paradigm and actor model address different levels of abstraction. While the actor model provides a solution to concurrent, distributed, and reconfigurable computation, multiagent systems are designed to intelligently and autonomously solve problems. In other words, a multiagent system could use the actor model as its foundation (and many agent frameworks have incorporated actor like behavior), but the converse may not hold. Similarly, there is no inherent incompatibility between the actor model and an SOA – just that the former can replicate the latter in a simpler way without the need for an additional messaging subsystem.

Given the functional requirements, described in Section 4, whereas an agent-based model would also be feasible, the actor model offers a simpler, more elegant solution, as described in the following sections.

5.2 Actor model

The actor model itself does not prescribe any programming language. It is a programming abstraction that only specifies the requirements, which fall into three categories: the structure of an actor model, the behavior of actors, and the communication between them. Specifically, an actor is a computing abstraction that contains a *local, immutable state*, does not share this local state with other actors, and is responsible for updating its local state.

Structurally, as shown in Fig. 13, 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 (Agha, 1986).

Within an *actor system*, each actor is assigned a unique *address* upon its creation, and an actor is free to share this address with other actors. This address follows the principle of *location transparency*, which means that an actor may be based on the same core, processor, or on a different node on the network (Karmani, Shali, & Agha, 2009). Unlike typical applications, actors are also *mobile*,

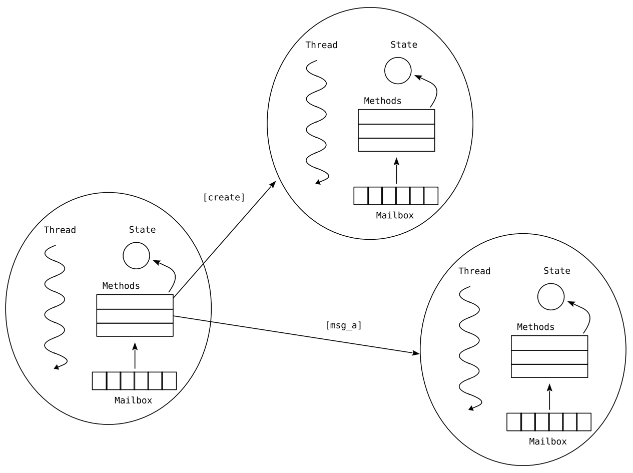


Figure 13: *Actor model*, taken from Karmani, Shali, and Agha (2009).

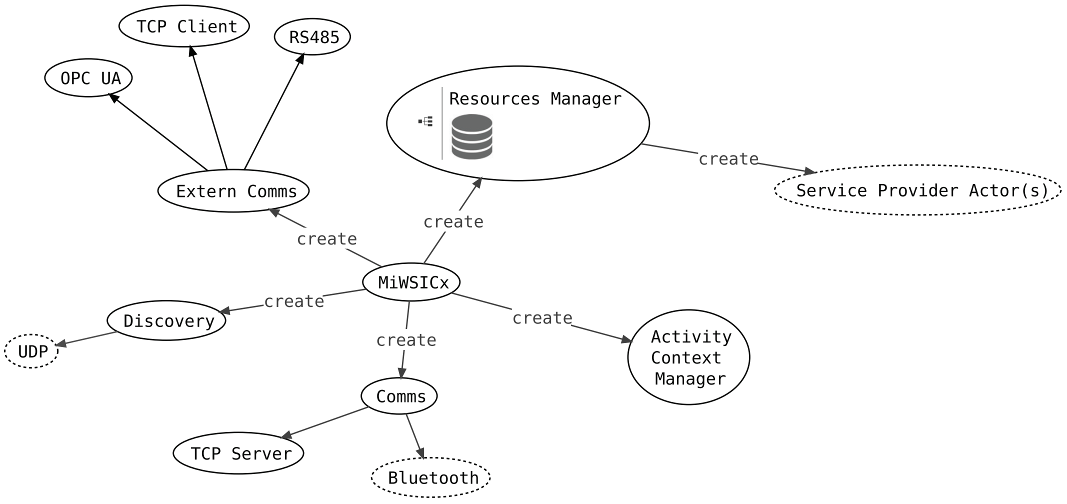


Figure 14: Implemented MiWSICx actors. Actors with dotted boundaries are possible extensions to the MiWSICx core.

which means that they can be updated and moved across nodes or reconfigured to handle varying loads without recompiling the application.

Therefore, the messaging mechanism in actors provides an abstraction that goes beyond the constraints imposed by publish–subscribe, remote-procedure call or a request–response service pattern, and yet, it provides a uniform interface on both global and local scales. Actors will communicate in the same manner, be it on the same hardware or the same network.

In summary, actors offer a level of abstraction that combines message passing and computation in a single, elegant package. In addition, actors could not only be used to deploy the middleware for managing activities as presented in this paper, but also run the associated data intensive tasks, such as image processing, data analysis, and filtering applications, many of which could constitute the computational side of activities. Depending on one's choice of programming language, several flavors of actor models are available: the most popular being the Java Virtual Machine-based *Akka* Framework (Thurau, 2012), the C++ Actor Framework (Charousset, Hiesgen, & Schmidt, 2014), the C#-based *Orleans* Framework (Bernstein & Bykov, 2016), and the Python-based Thespian Actor Framework (Quick, 2018).

5.3 MiWSICx actors

MiWSICx has been developed in Python, chosen because of its crossplatform compatibility, its dynamic typing system, and its rich repertoire of stable libraries serving both scientific and engineering needs. The Thespian Actor Framework (Quick, 2018) provides the actor model. In the following sections, the different actors in MiWSICx are explained.

5.3.1 *Core actors*

Figure 14 shows the core actors in MiWSICx. The *root* or *top level* actor in MiWSICx is called by the same name. Its job is to start up the MiWSICx base system, which consists of

- the *comms actor* that maintains communication with devices;
- the *resource manager actor*, responsible for handling persistent data storage, activity templates, and loading activities on runtime;
- the *activity context manager actor* that manages instantiating activities based on events and user requests;
- the *activity manager actor* that runs the activity at a given time for a particular user;
- the *discovery actor* that allows devices to discover the MiWSICx node;
- the *external comms* actor that creates respective actors to communicate with other nonactor-based services, such as RS485 networks or an OPC-UA server.

The core actors are informed of each others 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. The communication between different actors during startup, initialization, various stages of activity driven interaction and shutdown is detailed with the help of sequence diagrams in the appendix (Fig. A1 to A14).

Each MiWSICx node runs the actors in an *actor system*. Thespian combines multiple actor systems in a *convention*, with one actor system acting as the *convention leader*. Each actor system lists specific *capabilities*, which determines the kind of actors it can run. Upon starting up, an actor system registers itself with the convention leader, allowing actor systems to communicate with each other and, if needed, instantiate actors on other actor systems that can support a specific capability.

5.3.2 *Device handlers*

When a device connects to an MiWSICx node, it needs to announce its capabilities to the corresponding comms actor, which then creates a *device handler actor* that represents this device to MiWSICx. Once the device disconnects, the corresponding device handler actor is also destroyed.

5.3.3 *Activity contexts*

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.

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

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. Only one activity will be active at one time. As new users connect, each user is assigned its own activity context, and when a user logs out of the system, it is saved by the resources manager. In case of multiple users, the same activity could be assigned to two different contexts.

5.4 MiWSICx: communication

Thespian communicates within actor systems by *pickling* the Python MiWSICx event entity. In order to communicate with devices that do not support actors or use a different implementation of the actor model, the MiWSICx messaging protocol is used.

5.4.1 *Events*

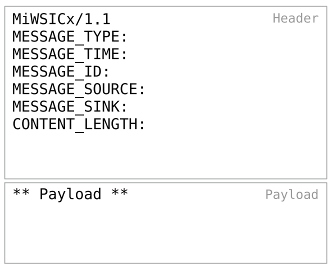
Between the different actors, artifacts, and objects, the abstraction that suitably covers conveying prompts and actions is that of an *event*.

Events can be enriched semantically by defining a subtype, adding a payload, and prioritized depending on the specific importance of the message. A prompt relating to an emergency situation, for example, is evidently more important than a prompt signaling the finishing of a process. Routing and filtering events require that events specify their producers and consumers (Etzion, Niblett, & Luckham, 2011). This leads to the definition of an *event* in MiWSICx, and it consists of the following attributes:

- *message type* that denotes the type of an event.
- *message subtype* specifies the event for a particular activity.
- *id* that represents a uniquely identifiable event.
- *priority* that signifies event importance. For example, an error event may be of a higher importance than other messages.
- *payload* refers to the content that elaborates the context of an event.
- *source* sender's address, represented as a hierarchical path to a particular task or an application on a device. For artifacts and components that implement the TCP–IP stack, the address begins with the IP address. In case of lower level communication networks such as RS485, it begins with the master or slave address.
- *sink* the receiver's address, represented in the same way as the source.

5.4.2 *Messaging protocol*

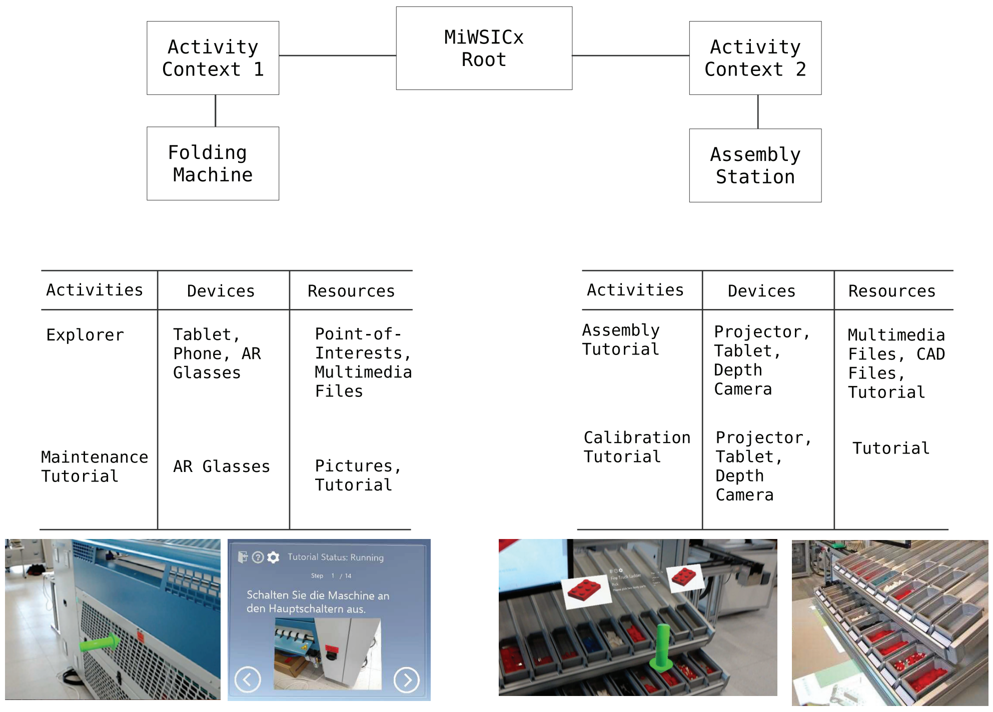
Communication both within MiWSICx actors and with external devices is maintained via a messaging protocol, divided into a header and payload, shown in Fig. 15. The header contains all the information needed to route events to their destination, while the payload contains information regarding the activity at hand. For networked devices relying on TCP/IP communications, a header format similar to HTTP and JSON for message payloads is used; for machine level communications with field bus devices such as using RS485, a specific hex code is used for each event field, followed by a value.



```
MiWSICx/1.1                       Header
MESSAGE_TYPE:
MESSAGE_TIME:
MESSAGE_ID:
MESSAGE_SOURCE:
MESSAGE_SINK:
CONTENT_LENGTH:

** Payload **                     Payload
```

Figure 15: MiWSICx messaging protocol.



Activities	Devices	Resources
Explorer	Tablet, Phone, AR Glasses	Point-of-Interests, Multimedia Files
Maintenance Tutorial	AR Glasses	Pictures, Tutorial

Activities	Devices	Resources
Assembly Tutorial	Projector, Tablet, Depth Camera	Multimedia Files, CAD Files, Tutorial
Calibration Tutorial	Projector, Tablet, Depth Camera	Tutorial

Figure 16: Deployment example.

5.5 Implementation example

Figure 16 shows the example implemented in our smart-factory demonstrator. There are two activity contexts supported at the moment: The first consists of a laundry folding machine and the second consists of an assembly station. There are three actor systems in use, each running on its own hardware (Raspberry Pi). The actor system with the “MiWSICx” capability acts as the convention leader, and serves as the entry point to the entire network of actor systems. The respective activity contexts are run on the other two actor systems. Activity context 1 runs two instructional activities – *explorer*, which shows the user the location of the various sensors on the machine in a 3D environment, and a *maintenance manual*, which guides the user through a maintenance process. Activity context 2 is an assembly instructor that delivers step-by-step instructions to the user on assembling diverse products. The change in the user’s context is detected by a positioning device linked to a portable arduino board with Wi-Fi capability, connected to the root actor system. Another possibility would be to use a smartwatch to exchange nearfield communication data with a sensor located in the specific activity context.

Since MiWSICx is implemented in Python that also supports scripting, the metamodel is instantiated directly without a domain-specific/descriptive language. The script instantiates the entities defined in the ontology (Section 4.1) as Python objects that are then serialized and stored for future use. A snippet from an activity hosted by activity context 1 is shown below. Creating an activity also creates the associated messages that the actor will act on; the behaviors that these messages activate are already supplied by the activity template.

```
template_dir = PersistentStorage.get_template_dir()
image_dir = PersistentStorage.get_image_dir()

" " "Example of a multi object, multi step activity" " "
activity_1 = MultiStepActivity()
activity_1.set_name("Cleaning_Machine_A")
```

```
# Reference Location
ref_location = Description.Representation()
ref_location.set_text("Locate this marker to start the activity.")
# Reference Resource
ref_resc = Resource("Reference", "marker.jpg", ResourceType.IMAGE)
ref_location.set_resource(ref_resc)
ref_location.need_capability(FeedbackType.VISUAL2D)
activity_1.set_reference_representation(ref_location)
# Step 1
step_1 = Resource.ResourceStep()
step_1.set_number(1)
step_1.set_text("Please turn off the switch indicated below")
step_1.set_behavior("edit_text", step_1.edit_text)
step_1.need_capability(FeedbackType.VISUAL2D)
# Object
switch = Object()
switch.create_identity("Switch", 0x1A5)
switch.add_location(Where("SwitchBox", 0.5, 1.0, 3))
switch.need_capability(FeedbackType.VISUAL2D, FeedbackType.VISUAL3D)
step_1.add_object(switch)

activity_1.set_behavior("locate", activity_1.locate)
activity_1.set_behavior("next", activity_1.next)
activity_1.set_behavior("prev", activity_1.prev)
```

These activities are inflated back to objects by an actor whenever a user joins a particular activity context through a device and wishes to work on an activity. The user's own activity context (hence the state including any changes the user made) is stored back to a serialized format after logging out, available for future use. Since the source code itself does not have to be compiled, new activities can be added as serialized objects without recompiling the rest of the system.

The actor system itself exchanges the *event* object among actors, but for communicating with artifacts outside the actor system, this object is encoded into the protocol specified in Section 5.4.2, with a JSON payload (supported natively by Python) along with the MiWSICx protocol header.

6 Challenges and Issues for Further Exploration

The scenario presented in the previous section represents our attempt at realizing the metamodel introduced in Section 4. In our view, it only presents one possible approach to the problem of activity-driven assistance, since the kind of activities to be supported will vary based on the needs of operators, the tools, and the environment on each shop floor. To this end, we propose some suggestions and themes for future research.

6.1 Activity dynamics: behavior versus content

The middleware conveys the ordering and actions supported by resources and objects in activities, but this information becomes actionable only if the frontend also replicates the ordering and affords the actions. So, for example, if a resource's text supports editing, the frontend application has to provide the user the tools to edit the text. Unlike dynamic webpages generated by servers, mobile applications have to be programmed such that a symmetrical relationship between the actions supported by the backend and the frontend can be maintained. In our prototype, we use the same frontend application to support different activities because of their similar nature. This approach will not work for activities with distinct presentation and interaction requirements, since separate mobile applications would have to be programmed for each platform/device. Future research efforts could possibly focus on reducing this effort by coupling activity instantiation to the creation of frontend for multiple devices using crossplatform frameworks.

6.2 User created, managed, and enriched activities

Over time, as the nature of activities changes or new activities are introduced, the best case scenario would allow users to compose and edit their own activity workflows and reduce their reliance on others to program and instantiate activities for them. We use the term "best case scenario" because of the following reason: The users themselves are the first hand observers of their activities in comparison to designers, who can infer user intent only during the design phase. Applications allow users to add/remove/edit content but changing their structure requires a new design iteration and compilation. Future research efforts could explore methods (such as visual programming) for users to compose and maintain their own workflows on a combination of mobile devices by modifying both content and structure. Likewise, users could also enrich these activities by creating and adding their own combinations of images, annotations, and 3D coordinates.

7 Conclusion

The only way of handling complexity is to embrace it, a theme that is recurrent in recent research on CPPS. The human operator is seen as the final stop on the path to handling complex systems; nonetheless, the vision of *operator 4.0* remains far from realized – as a first step, there is a need to think beyond static human–machine interfaces, from providing assistance to providing *activity-driven assistance* – delivering timely, adaptive, and hence, activity-centric information on a combination of multiple interactive devices. Our approach reimagines work as activities composed of the human operator, his/her artifacts as devices and informational resources, and the objects that are the recipients of this activity.

This paper presented the ontology, information model, implementation, and application of MiWSICx, a distributed middleware implemented to provide activity-driven assistance in industrial contexts. MiWSICx utilizes the actor model to maintain a high level of responsiveness in handling multiple users, activities, and crossdevice interaction simultaneously, while also avoiding the drawbacks of thread-based concurrency approaches. By modeling the constituents of human activity – users, artifacts, resources, and objects, MiWSICx manages activities across various interactive device configurations. Through the work presented in this paper, we hope to motivate the design of systems that are adaptable and extensible by the human operators themselves as their own creative problem-solving tools.

Conflict of interest statement

None declared.

References

Abramovici, M. (2015). Smart products. In *CIRP Encyclopedia of Production Engineering*(pp. 1–5). Berlin, Germany: Springer.

Agha, G. (1986). *Actors: A model of concurrent computation in distributed systems*. Cambridge, MA: MIT Press.

Baerentsen, K. B., & Trettvik, J. (2002). An activity theory approach to affordance. In *Proceedings of the Second Nordic Conference on Human-Computer Interaction*, NordiCHI '02(pp. 51–60). New York, NY: ACM.

Banks, A., & Gupta, R. (2014). Mqtt version 3.1. 1, *OASIS standard*, *29*.

Bardram, J. E. (2011). The activity-based computing project. In *AAAIWS'11-04: Proceedings of the Fourth AAAI Conference on Activity Context Representation: Techniques and Languages*(pp. 2–7).

Bødker, S., & Klokmose, C. N. (2011). The human-artifact model: An activity theoretical approach to artifact ecologies. *Human-Computer Interaction*, *26*, 315–371.

Becker, T., & Stern, H. (2016). Future trends in human work area design for cyber-physical production systems. *Procedia CIRP*, *57*, 404–409.

Bernstein, P. A., & Bykov, S. (2016). Developing cloud services using the orleans virtual actor model. *IEEE Internet Computing*, *2013-03* , 71–75.

Bettenhausen, K. D., & Kowalewski, S. (2013). Cyber-physical systems: Chancen und nutzen aus sicht der automation. *VDI/VDE-Gesellschaft Mess-und Automatisierungstechnik*, 9–10.

Broy, M., & Schmidt, A. (2014). Challenges in engineering cyber-physical systems. *Computer*, *47*, 70–72.

Brudy, F., Holz, C., Rädle, R., Wu, C.-J., Houben, S., Klokmose, C. N., & Marquardt, N. (2019). Cross-device taxonomy: Survey, opportunities and challenges of interactions spanning across multiple devices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19(pp. 1–28). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/3290605.3300792.

Byström, K., & Järvelin, K. (1995). Task complexity affects information seeking and use. *Information Processing & Management*, *31*, 191–213. http://linkinghub.elsevier.com/retrieve/pii/030645739580035R.

Charousset, D., Hiesgen, R., & Schmidt, T. C. (2014). Caf-the C++ actor framework for scalable and resource-efficient applications. In *Proceedings of the Fourth International Workshop on Programming based on Actors Agents & Decentralized Control*(pp. 15–28). New York, NY: ACM.

Dearman, D., & Pierce, J. S. (2008). It's on my other computer!: Computing with multiple devices. In *Proceeding of the 26th Annual CHI Conference on Human Factors in Computing Systems - CHI '08*(p. 767). Florence, Italy: ACM Press.

Dhiman, H., Martinez, S., Paelke, V., & Röcker, C. (2018). Head-mounted displays in industrial ar-applications: Ready for prime time?In *International Conference on HCI in Business, Government, and Organizations*(pp. 67–78). Berlin, Germany: Springer.

Dhiman, H., & Röcker, C. (2019). Worker assistance in smart production environments using pervasive technologies. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*(pp. 95–100).

Dhiman, H., Büttner, S., & Röcker, C. (2019). Handling work complexity with ar and deep learning – An exploratory approach to support production work. In *Proceedings of the 31st Australian Conference on Human-Computer-Interaction (ozCHI'19)*. Association for Computing Machinery, 518–522.

ElMaraghy, W., ElMaraghy, H., Tomiyama, T., & Monostori, L. (2012). Complexity in engineering design and manufacturing. *CIRP Annals*, *61*, 793–814.

Elting, C., Zwickel, J., & Malaka, R. (2002). Device-dependent modality selection for user-interfaces: an empirical study. In *Proceedings of the Seventh international Conference on Intelligent User Interfaces*(pp. 55–62). ACM.

Etzion, O., Niblett, P., & Luckham, D. C. (2011). *Event processing in action*. Greenwich, London: Manning.

Gorecky, D., Schmitt, M., Loskyll, M., & Zühlke, D. (2014). Human-machine-interaction in the industry 4.0 era. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*(pp. 289–294).

Guimaraes, T., Martensson, N., Stahre, J., & Igbaria, M. (1999). Empirically testing the impact of manufacturing system complexity on performance. *International Journal of Operations & Production Management*, *19*, 1254–1269.

Hewitt, C., Bishop, P., & Steiger, R. (1973). Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence. In *Advance Papers of the Conference*(Vol. 3, pp. 235). Menlo Park, CA: Stanford Research Institute.

Hintjens, P. (2013). *ZeroMQ: Messaging for many applications*. Newton, MA: O'Reilly Media, Inc.

Jokela, T., Ojala, J., & Olsson, T. (2015). A diary study on combining multiple information devices in everyday activities and tasks. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15(pp. 3903–3912). New York, NY: ACM.

Karmani, R. K., Shali, A., & Agha, G. (2009). Actor frameworks for the jvm platform: A comparative analysis. In *Proceedings of the Seventh International Conference on Principles and Practice of Programming in Java*, PPPJ '09(pp. 11–20). New York, NY: ACM.

KDE.org(2018). Plasma is kde's flagship product, offering the most customizable desktop environment available. https://www.kde.org/plasma-desktop/. Last accessed 20 Nov. 2020.

Leont'ev, A. N. (1978). *Activity, consciousness, and personality*. Englewood Cliffs, NJ: Prentice-Hall. http://lchc.ucsd.edu/MCA/Paper/leontev/.

Lucke, D., Constantinescu, C., & Westkämper, E. (2008). Smart Factory - A step towards the next generation of manufacturing. *Manufacturing Systems and Technologies for the New Frontier*(pp. 115–118). London: Springer London.

Moore, G. M., Chrysanthakopoulos, G., & Nielsen, H. F. (2007). Multiple redundant services with reputation. US Patent 7,310,641.

Moran, T. P., Cozzi, A., & Farrell, S. P. (2005). Unified activity management: supporting people in e-business. *Communications of the ACM*, 48, 67–70.

Norman, D. A. (1986). Cognitive engineering. *User Centered System Design*, *31*, 61.

Ong, S. K., & Nee, A. Y. C. (2013). *Virtual and augmented reality applications in manufacturing*. Berlin, Germany: Springer Science & Business Media.

Pardo-Castellote, G. (2003). Omg data-distribution service: Architectural overview. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference*(pp. 200–206). IEEE.

Paridon, H. M., & Kaufmann, M. (2010). Multitasking in work-related situations and its relevance for occupational health and safety: Effects on performance, subjective strain and physiological parameters. *Europe's Journal of Psychology*, 6, 110–124. https://ejop.psychopen.eu/index.php/ejop/article/view/226.

Quick, K. (2018). Python actors. https://thespianpy.com/doc/. Last accessed 20 Nov. 2020.

Rasmussen, J., & Lind, M. (1981). Coping with complexity. In H. G. Stassen (Ed.), *First Annual European Conference on Human Decision-Making and Manual Control*(pp. 70–91). New York: Plenum Press.

Romero, D., Stahre, J., & Taisch, M. (2020). The Operator 4.0: Towards socially sustainable factories of the future. *Computers & Industrial Engineering*, 139, 106128.

Sand, O., Büttner, S., Paelke, V., & Röcker, C. (2016). Smart.assembly – Projection-based augmented reality for supporting assembly workers. In *International Conference on Virtual, Augmented and Mixed Reality*(pp. 643–652). Springer.

Santosa, S., & Wigdor, D. (2013). A field study of multi-device workflows in distributed workspaces. In *UbiComp '13: Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*(pp. 63–72). New York: ACM Press.

Scafá, M., Marconi, M., & Germani, M. (2020). A critical review of symbiosis approaches in the context of Industry 4.0. *Journal of Computational Design and Engineering*, *7*, 269–278. https://doi.org/10.1093/jcde/qwaa022.

Spath, D., Ganschar, O., Gerlach, S., Moritz, H., Krause, T., & Schlund, S., (2013). *Produktionsarbeit der Zukunft - Industrie 4.0*, Fraunhofer Verlag.

Srensen, H., Raptis, D., Kjeldskov, J., & Skov, M. B. (2014). The 4c framework: Principles of interaction in digital ecosystems. In *UbiComp '14: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*(pp. 87–97). ACM Press.

Strauch, B. (2018). Ironies of automation: Still unresolved after all these years. *IEEE Transactions on Human-Machine Systems*, 48, 419–433.

Tepjit, S., Horváth, I., & Rusák, Z. (2019). The state of framework development for implementing reasoning mechanisms in smart cyber-physical systems: A literature review. *Journal of Computational Design and Engineering*, *6*, 527–541. https://doi.org/10.1016/j.jcde.2019.04.002.

Thurau, M. (2012). Akka framework. University of Lübeck.

Uhlemann, T., Lehmann, C., & Steinhilper, R. (2017). The digital twin: Realizing the cyber-physical production system for industry 4.0. *Procedia CIRP*, *61*, 335–340.

Vakkari, P. (1999). Task complexity, problem structure and information actions. *Information Processing & Management*, 35, 819–837. http://linkinghub.elsevier.com/retrieve/pii/S030645739900028X.

Valdez, A. C., Brauner, P., Schaar, A. K., Holzinger, A., & Ziefle, M. (2015). Reducing complexity with simplicity - Usability methods for industry 4.0. In *Conference: IEA 2015*(Vol. *19*).

Vernier, F., & Nigay, L. (2000). A framework for the combination and characterization of output modalities. In *International Workshop on Design, Specification, and Verification of Interactive Systems*(pp. 35–50). Springer.

Vogel-Heuser, B., Lee, J., & Leitão, P. (2015). Agents enabling cyber-physical production systems. *at-Automatisierungstechnik*, *63*, 777–789.

Vygotsky, L. S. (1980). *Mind in society: The development of higher psychological processes*. Cambridge, MA: Harvard University Press.

Wartofsky, M. W. (2012). *Models: Representation and the scientific understanding*(Vol. 48). Berlin, Germany: Springer Science & Business Media.

Wooldridge, M. (2009). *An introduction to multiagent systems*. (2nd ed.). Hoboken, NJ: Wiley.

Zacher, H., & Frese, M. (2018), Action regulation theory: Foundations, current knowledge, and future directions. In *The SAGE handbook of industrial, work, and organizational psychology*(pp. 80–102). Sage Project.

Zamfirescu, C.-B., PâRvu, B.-C., Schlick, J., & ZûHlke, D. (2013). Preliminary insides for an anthropocentric cyber-physical reference architecture of the smart factory. *Studies in Informatics and Control*, 22, 269–278.

Appendix 1: A UML Sequence Diagrams for Core Actors

The following pages contain sequence diagrams that illustrate the interaction between different MiWSICx actors in various stages of operation. The ordering of the diagram begins with the startup of the actor system, followed by a connection from the user device, subsequent creation/restoration of an activity context, after which the user can interact with the actor system and work on their activities. Once the user exits an activity and does not wish to switch, the user has the option to disconnect. The final diagram illustrates the shutdown sequence of the MiWSICx actor system.

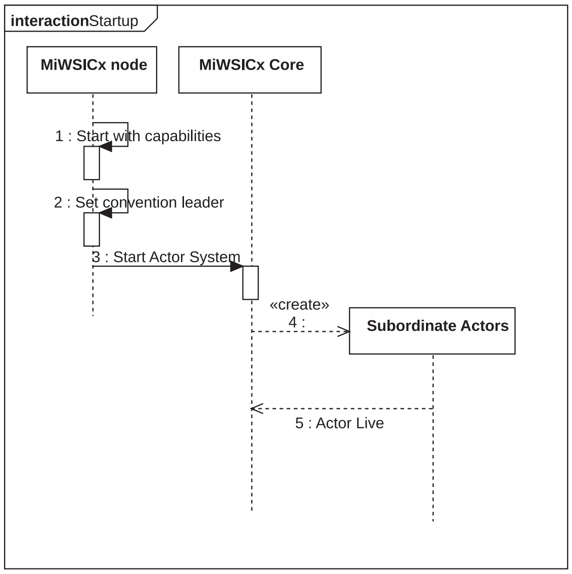


Figure A1: Actor system startup.

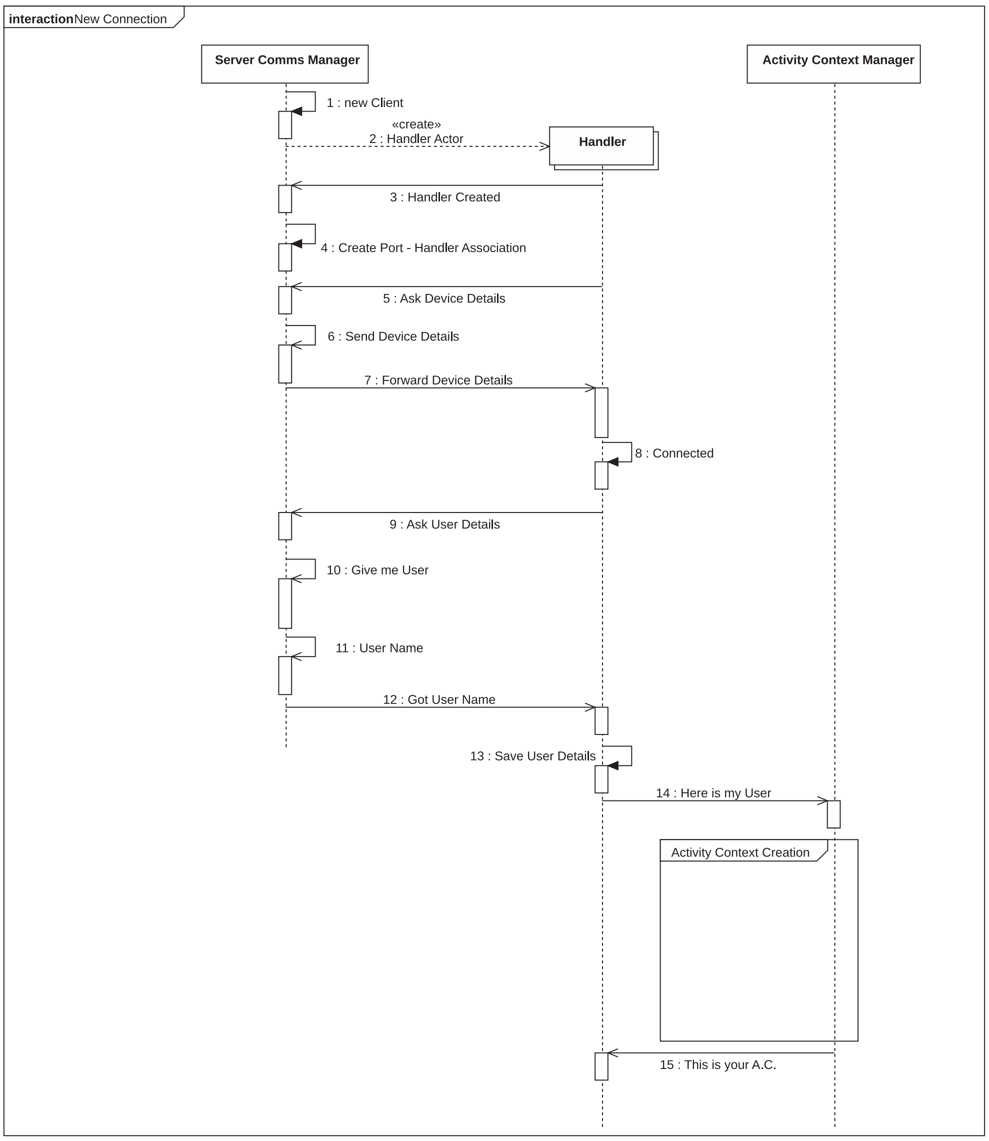


Figure A2: New connection from device.

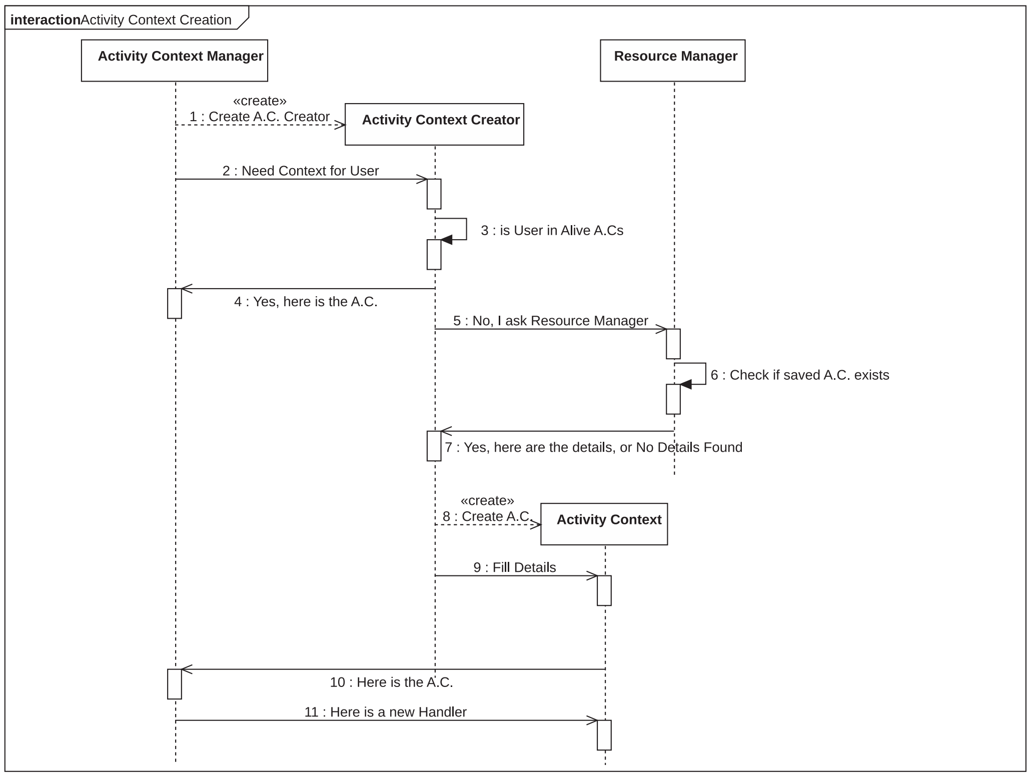

interactionActivity Context Creation
Activity Context Manager
Resource Manager
«create»
1 : Create A.C. Creator
Activity Context Creator
2 : Need Context for User
3 : is User in Alive A.Cs
4 : Yes, here is the A.C.
5 : No, I ask Resource Manager
6 : Check if saved A.C. exists
7 : Yes, here are the details, or No Details Found
«create»
8 : Create A.C.
Activity Context
9 : Fill Details
10 : Here is the A.C.
11 : Here is a new Handler

Figure A3: Activity context creation.

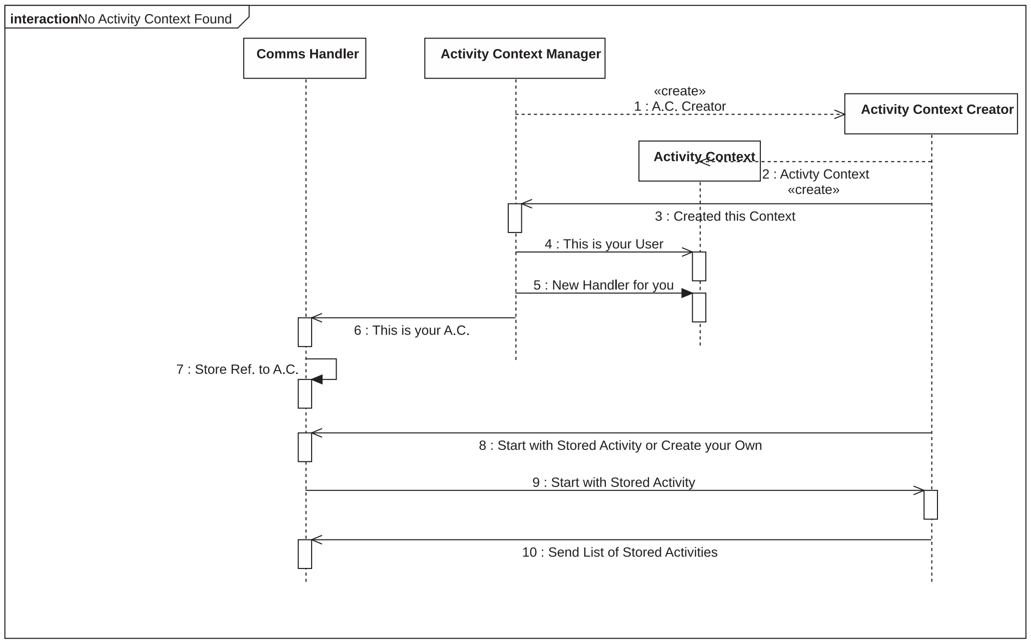


Figure A4: Creating a new activity context.

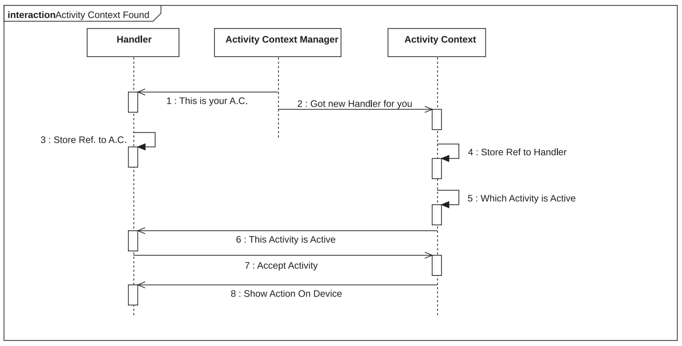

interactionActivity Context Found
Handler
Activity Context Manager
Activity Context
1 : This is your A.C.
2 : Got new Handler for you
3 : Store Ref. to A.C.
4 : Store Ref to Handler
5 : Which Activity is Active
6 : This Activity is Active
7 : Accept Activity
8 : Show Action On Device

Figure A5: Restoring an existing activity context.

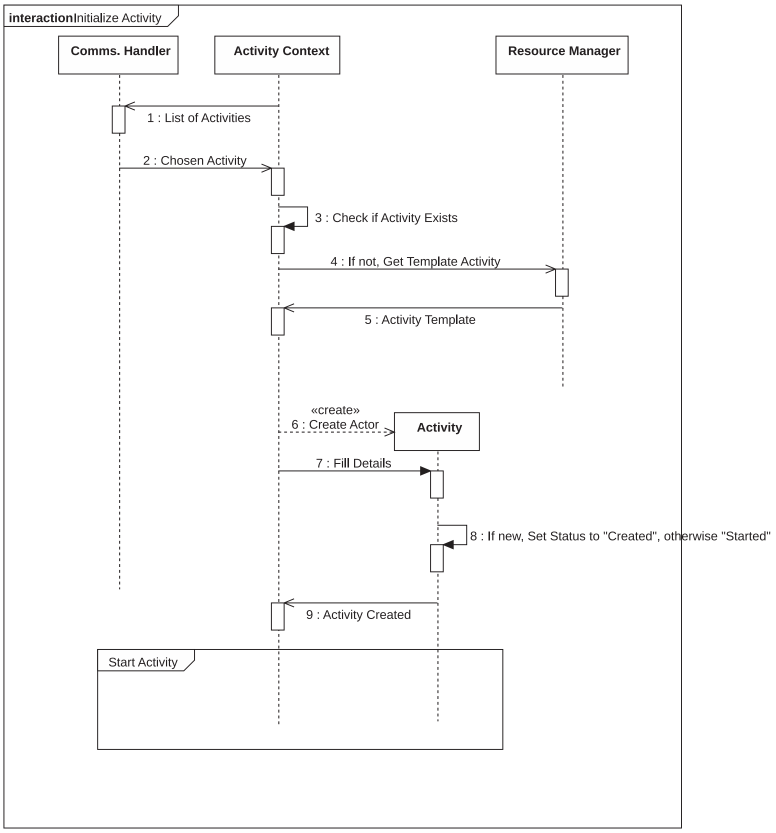


Figure A6: Initializing a user activity.

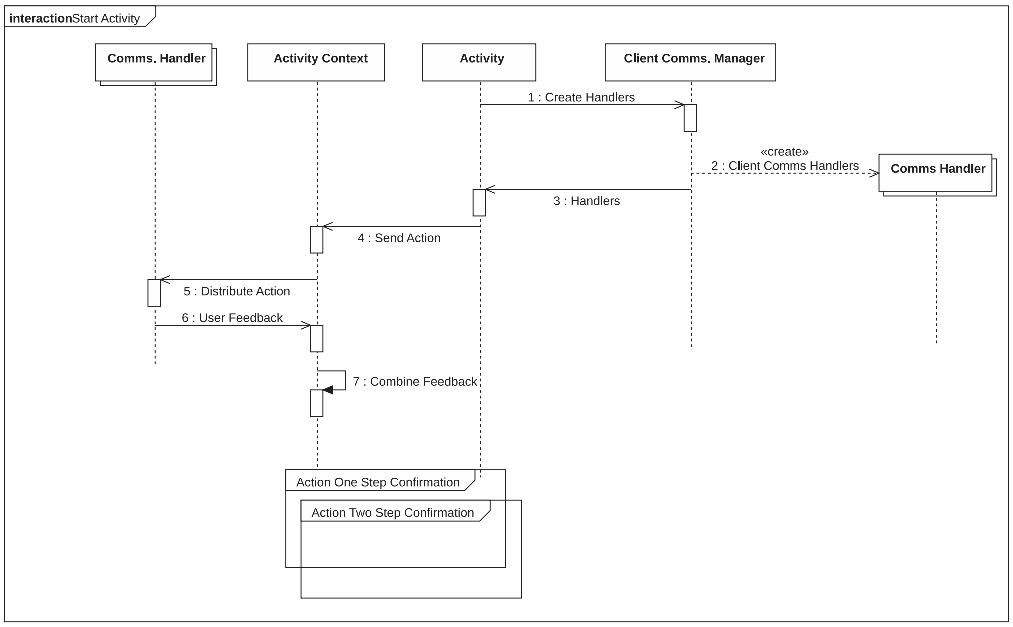


Figure A7: Starting user activity.

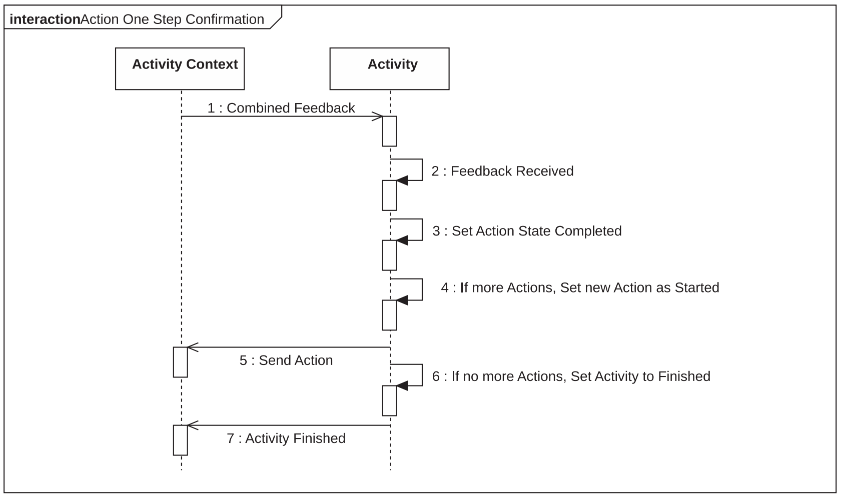

interactionAction One Step Confirmation
Activity Context
Activity
1 : Combined Feedback
2 : Feedback Received
3 : Set Action State Completed
4 : If more Actions, Set new Action as Started
5 : Send Action
6 : If no more Actions, Set Activity to Finished
7 : Activity Finished

Figure A8: Running user activity with interaction with the user device.

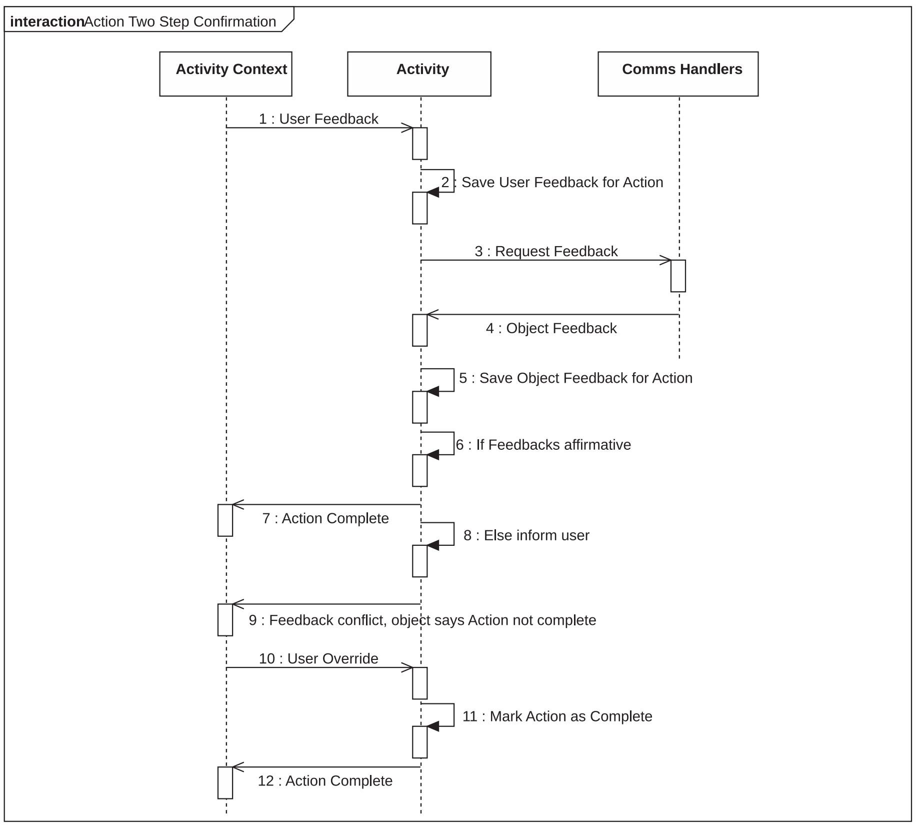

interactionAction Two Step Confirmation
Activity Context
Activity
Comms Handlers
1 : User Feedback
2 : Save User Feedback for Action
3 : Request Feedback
4 : Object Feedback
5 : Save Object Feedback for Action
6 : If Feedbacks affirmative
7 : Action Complete
8 : Else inform user
9 : Feedback conflict, object says Action not complete
10 : User Override
11 : Mark Action as Complete
12 : Action Complete

Figure A9: Running user activity with interaction with the user and CPP objects.

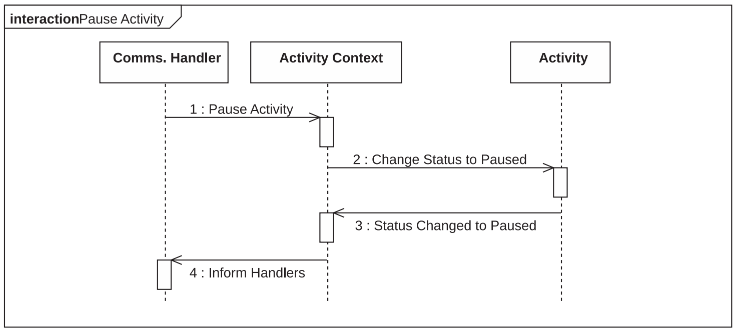

interactionPause Activity
Comms. Handler
Activity Context
Activity
1 : Pause Activity
2 : Change Status to Paused
3 : Status Changed to Paused
4 : Inform Handlers

Figure A10: Pausing user activity.

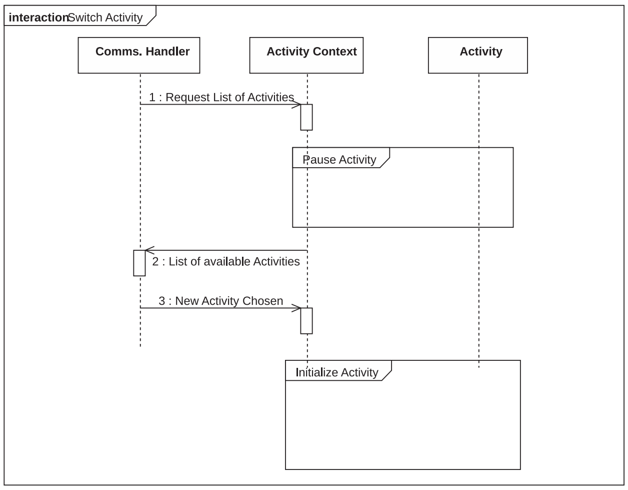


Figure A11: Switching between activities.

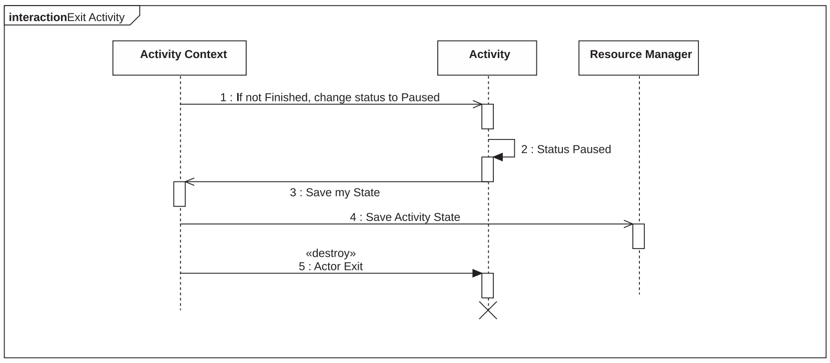


Figure A12: Exiting an activity.

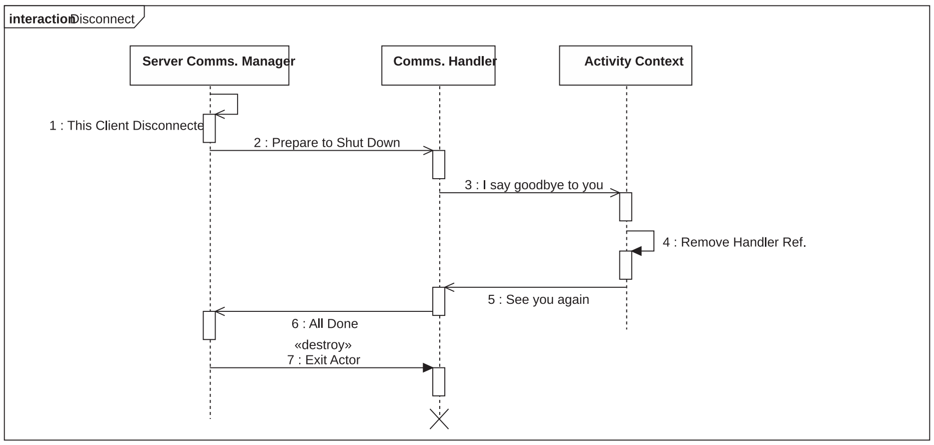


Figure A13: Client disconnection from actor system.

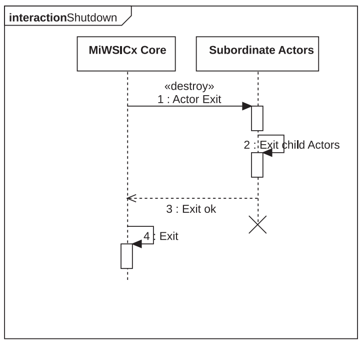


Figure A14: Actor system shutdown.