Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 190–199

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

A Multi-Layer Architecture for Near Real-Time Collaboration during Distributed Modeling and Simulation of Cyberphysical Systems[1]

Paul Lonauer[a], David Holzmann[a], Christina Leitner[a], Alexander Probst[a], Stefan Pöchhacker[b], Stefan Oberpeilsteiner[b], Johannes Schönböck[a,*], Hans-Christian Jetter[a,c]

[a] *Faculty of Informatics, Communications and Media, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria*
[b] *Faculty of Engineering and Environmental Sciences, University of Applied Sciences Upper Austria, Stelzhamerstraße 23, 4600 Wels, Austria*
[c] *University of Lübeck, Ratzeburger Allee 160, 23562 Lübeck, Germany*

Abstract

We present a Web-based multi-layer architecture and implementation to enable near real-time collaboration within partially-distributed engineering teams in this paper. Our architecture named Distributed Modeling and Simulation (DisMoSim) enables collaborative 3D modeling and simulation of complex cyberphysical systems (CPS) across different servers, offices, lab spaces, or organizations. More specifically, DisMoSim creates networks of DisMoSim nodes that communicate using Web protocols with little or no perceptible delay. Each node provides different CPS services, e.g., user interface nodes provide collaborative 3D modeling; computational nodes provide simulations of multi-body dynamics or kinematics; storage nodes allow persisting and sharing models and simulation results; hardware-in-the-loop nodes connect physical testing workbenches to provide real-world sensor measurements as simulation input; software-in-the-loop nodes provide control signals to simulate hardware controllers.

In the following, we derive a set of design goals and propose our DisMoSim architecture. We illustrate how we used our multi-layer conceptual architecture to implement a variety of CPS services for an example scenario. By this, we demonstrate the practicability and versatility of our approach and conclude by discussing limitations and future work.

© 2021 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the International Conference on Industry 4.0 and Smart Manufacturing

Keywords: Distributed Modeling; Distributed Simulation; Cyberphysical Systems; Hybrid Collaboration; Software Architecture

1. Introduction

A key part of the vision of "Industry 4.0" is the decomposition of complex processes into smaller tasks and dynamically distributing them across digitally connected machines and workers from different teams or suppliers [19]. In fact, already before this was termed "Industry 4.0", manufacturers had started to cope with the growing complexity of prod-

[1] This work has been funded by FFG COIN 866851.
* Corresponding author. Tel +43 50804-22625
E-mail address: johannes.schoenboeck@fh-hagenberg.at

1877-0509 © 2021 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the International Conference on Industry 4.0 and Smart Manufacturing
10.1016/j.procs.2021.01.156

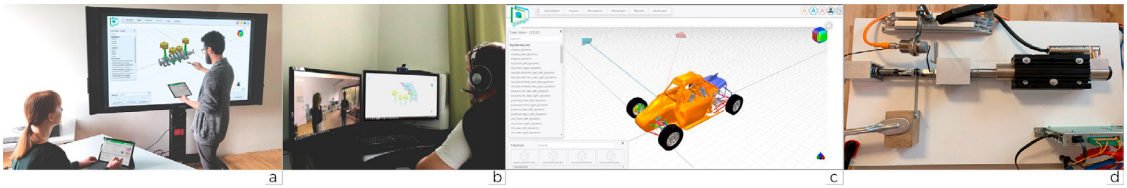

Fig. 1. Engineering teams from two companies (a,b) remotely collaborate in a virtual workshop (c) to model a new ATV. Simulation results are based on computational models but also physical prototypes connected via the internet, e.g., a HIL testing bench for wheel suspensions (d).

uct development (e.g., in vehicle, machine, or plant construction) by increased collaboration across companies and breaking down products into modules, each of which are developed and delivered in its entirety from a supplier [6]. This general trend towards modularization and horizontal integration across the entire value-creation network [20] has resulted in a growing demand for tools for virtual engineering that enable better remote collaboration between engineers. This demand is growing further due to globalized supply chains, especially when composing subsystems from international suppliers, and, very recently, due to COVID-19 social distancing. In our research project *Distributed Modeling and Simulation* (DisMoSim), we develop such collaboration tools for virtual engineering to enable the distributed modeling and simulation of a *cyberphysical system* (CPS) [11] across different teams and suppliers.

A typical scenario involves a team of engineers of *company A* (see Figure 1a) who compose a new product such as a new all-terrain vehicle (ATV) inside a *virtual workshop* (see Figure 1c) from models of mechanical parts (e.g., suspension systems, gearboxes) of supplier *company B*. These subsystems are partially available as computational models for simulation on the servers of *company A* but some of them are only available as physical prototypes with automated actuators and sensors that provide measurements as a "physical" simulation or *hardware-in-the-loop* (HIL) [3] in *company B*'s testing lab (see Figure 1d). Furthermore, digital control systems (e.g. stability control, motor control) from a second supplier (*company C*) need to be integrated. Some are available as physical controller boards in a HIL setup and some only as software simulations in a *software-in-the-loop* (SIL) setup [3].

The sum of these computational and physical subsystems at different locations can be considered a CPS that enables the distributed modeling and simulation of the ATV. The involved engineers need shared access to this CPS to collaboratively model, simulate, visualize results, and optimize the ATV and the relevant interactions between all physical or computational subsystems before further prototyping or production planning.

To provide shared access to CPS modeling and simulation, we propose a multi-layer software architecture that enables teams to collaborate in near real-time across different locations by creating a network of connected DisMoSim nodes (see Figure 2). Each node provides different services including user interfaces for collaborative 3D modeling

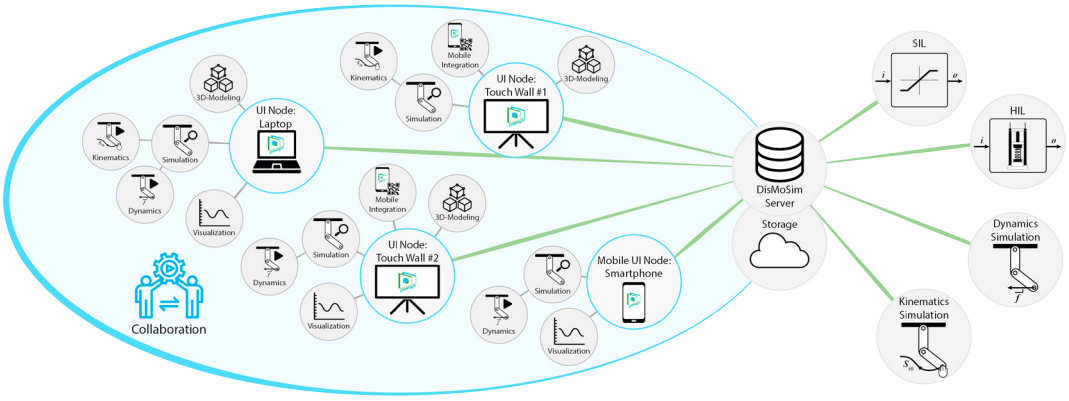

Fig. 2. Exemplary DisMoSim network with nodes (larger bubbles) and their services (smaller bubbles). ***Left:*** Three distributed teams access a shared virtual workshop on the DisMoSim server using three types of UI nodes, two large touch screens (Touch Wall #1 & #2) and a laptop, for 3D modeling, starting simulations, and visualizing simulation results. Smartphones can act as standalone UI nodes for managing longer-term simulation tasks or can be paired as external controllers with Touch Walls. Communication is provided via the DisMoSim server in near real-time. ***Right:*** Nodes with computational and physical resources, e.g., simulation nodes provide kinematic or dynamic multi-body simulations, HIL and SIL nodes contribute physical measurements or controller outputs to simulation processes.

and visualization of simulation results, computational services for distributed simulation or storage, and integration of SIL and HIL. For easy integration into existing IT infrastructures, all services use common Web technologies for data storage and network communication with little or no perceptible delay, i.e., "near real-time".

Our main contribution is the proposed DisMoSim architecture and a detailed description of how it can be used to implement nodes and services for the distributed modeling and simulation of CPS. We also demonstrate the versatility of our proposal by describing how we implemented nodes and services for the above-mentioned DisMoSim example scenario of collaborative 3D modeling for kinematic and dynamic multi-body simulation. To achieve this, we first discuss related work and derive a set of design goals for the DisMoSim architecture. We then introduce our conceptual DisMoSim multi-layer architecture, and describe how we used it to implement nodes and services using standard Web technologies to meet these requirements. We conclude by discussing limitations of our architecture and future work.

2. Related Work

DisMoSim builds upon a wide range of related work from Human-Computer Interaction (HCI), Computer-Supported Cooperative Work (CSCW), and software engineering. We here present selected related work in three broad categories: (i) human-centered perspectives on virtual engineering tools from HCI, (ii) research on CSCW for engineering, and (iii) technologies for implementing collaborative workspaces. This previous and related work influenced our formulation of requirements for our architecture and our proposed design solutions.

(i) HCI Research on Virtual Engineering. Research on digital tools for virtual engineering has always been a source of innovation for HCI and computer science in general. A famous example is Sutherland's Sketchpad [21] from 1963 which introduced the first GUI with pen input and can be seen as the forefather of CAD software. In the 1990s, the URP system by Underkoffler and Ishii's introduced tangible UIs on tabletops to support co-located teams during urban planning by letting them arrange tangible buildings and visualizing simulated shadows or wind effects in real-time [22]. In the 2000s, Fitzmaurice et al. further explored the roles that different I/O-technologies and form factors of devices can play in design and engineering. Their vision of "sentient data access" [5] includes large screen PowerWalls for presentation of car designs, augmented reality for displaying 3D models, and non-UI devices such as connected milling machines to produce foam models. Similarly, our DisMoSim Web-based multi-layer architecture also supports a wide variety of devices and I/O technologies, e.g., multi-touch walls with pen input. This also includes non-UI devices that enable engineering services, e.g., the integration of simulation servers and HIL or SIL setups.

(ii) CSCW Research on Collaborative Engineering. Traditional CSCW tools are classified by time and space [10]: E-mail is asynchronous and remote (*different time, different place*), a video-call is synchronous and remote (*same time, different place*), and large shareable devices such as vertical or horizontal screens for group work support *same time, same place* collaboration [8, 17]. More recently, researchers have been moving towards *hybrid collaboration*, i.e., collaboration that involves simultaneous co-located and remote collaboration with phases of both synchronous and asynchronous work across multiple groupware applications and devices [15]. DisMoSim aims at supporting such hybrid collaboration by supporting partially-distributed teams with both co-located and remote members using either shareable devices such as touch walls or non-shareable devices such as laptops or smartphones.

A success factor for hybrid collaboration is *workspace awareness* [7], i.e., the ability of users to determine who else is in the workspace, where they are working, and what they are doing. It is insufficient to simply provide audiovisual live streams from remote users because they can be perceived as distracting or disturbing and can result in inefficient or failing teamwork [15]. *Workspace awareness* in 3D spaces is particularly challenging since video or audio feeds are unable to provide enough *spatial awareness*. Collaborators lack understanding where their remote counterparts are currently looking and working due to spatial discontinuities between the remote and local camera perspective on the 3D world [2], and also between the video cameras of collaborators and the screen locations of their video streams [1]. Recent work has therefore experimented with a wide range of awareness cues for better 3D collaboration. This ranges from visualizing the remote users' view frustums or lines of sight including hands or avatars [2] to attempts of detailed "holographic" representations of remote users [16]. In DisMoSim, we are using Web technologies to experiment with different combinations of video and audio feeds and 3D visualization to optimize our awareness cues.

(iii) Software Architectures for Collaborative Systems. Over the decades, there have been many different approaches for the implementation of near real-time collaboration experiences. The core ideas of these approaches have remained relatively similar but their implementations have undergone many iterations or paradigm shifts with new programming languages, database and network technologies, or operating systems. Roseman & Greenberg introduced

GroupKit in 1996 that provides shared data using synchronized key-value-dictionaries, remote procedure calls, and shared widgets to compose UIs in C++ and Tcl/Tk for UNIX workstations [18]. Jetter et al.'s ZOIL software framework from 2012 uses C# and .NET / Windows Presentation Foundation for providing synchronized object-oriented data models using db4o, OSC commands for remote procedure calls, and interactive domain objects defined by C# and XAML [9]. Klokmose et al.'s Webstrates from 2015 was implemented in CoffeeScript on Node.js and shares data by synchronizing DOMs of web pages with all clients via ShareDB, triggering attached functions with events on DOM elements, and defining interactions and visuals in the form of substrates using HTML5/JS [12].

Apart from composing UIs from different building blocks (widgets [18] vs. objects [9] vs. instruments [12]), these approaches also differ in their management of concurrent, and potentially conflicting, user actions. Simple solutions are locking or non-locking optimistic synchronization [9, 18]. More advanced algorithms for concurrency control such as operational transformation [4] have been popularized by Google Docs and are also used in Webstrates [12].

The aforementioned approaches have influenced the DisMoSim architecture. Like ZOIL [9], DisMoSim is based on a NoSQL database for synchronization of a shared object-oriented data model. However, this shared model is provided by the Meteor framework that uses JavaScript and WebSockets like Webstrates [12]. Also, all interactions and visuals are implemented using HTML5/JS and not XAML/C#.

3. Design Goals and Architecture

Based on the aforementioned related work and sociotechnical and business needs, we formulate four design goals for DisMoSim to summarize the most important requirements.

3.1. Design Goals

DG1 - Virtual Workshop. The overall design goal of DisMoSim is to provide a *virtual workshop* as a visual workspace where multiple engineers can collaborate in near real-time. The virtual work environment should allow for easy sharing of work processes, e.g., jointly modeling parts of a CPS and their discussion in a 3D environment (see Figure 1c), including means for video conferencing and joint 3D interaction (see Figure 1b).

DG2 - Hybrid Collaboration. Hybrid collaboration [15] by partially-distributed teams enables group decision making and can reduce an individual's cognitive load and error rate. DisMoSim UIs must support hybrid collaboration, so that teams can work simultaneously co-located or remote and across different devices (see Figure 1a & b). Awareness as well as efficient and equitable interaction styles are a key design goal.

DG3 - Multi- and Cross-Device Interaction. The device-specific advantages/disadvantages shall be exploited. For example, virtual workshops can be accessed via desktop/laptop but also via large multi-touch and pen-enabled displays for co-located collaborative modeling (see Figure 1a). For specific tasks, smartphones can be used, e.g., for managing simulation tasks while on the road, or as 6-DoF-controllers for mid-air interactions during 3D modeling.

DG4 - Web-based Near Real-Time Communication. DisMoSim requires near real-time communication between different services and nodes. For example, fast communication is required for group awareness by seemingly immediate synchronization of the 3D workspace, virtual cameras, and user views as well as flawless audio & video conferencing. Also, computationally-intensive simulation tasks need to be distributed between several high-performance computational nodes. Especially time critical are synchronous simulation tasks, e.g., interactive kinematics simulations to show how a mechanical system (e.g., an engine) would move in the real world in reaction to user-generated movements of its parts (e.g., moving a piston by touch or mouse).

3.2. Conceptual Architecture

To realize the DisMoSim network, we propose a three-layer architecture (see Figure 3) that enables the implementation of diverse CPS services (see Section 4) using a shared technological base. The three layers are (i) the *network communication layer*, (ii) the *model layer*, and (iii) the *service layer*.

3.2.1. DisMoSim Network Communication Layer

The DisMoSim Network Communication Layer provides the core functionality to exchange data between nodes and services and is represented by green edges in Figure 2. The current implementation supports two protocols.

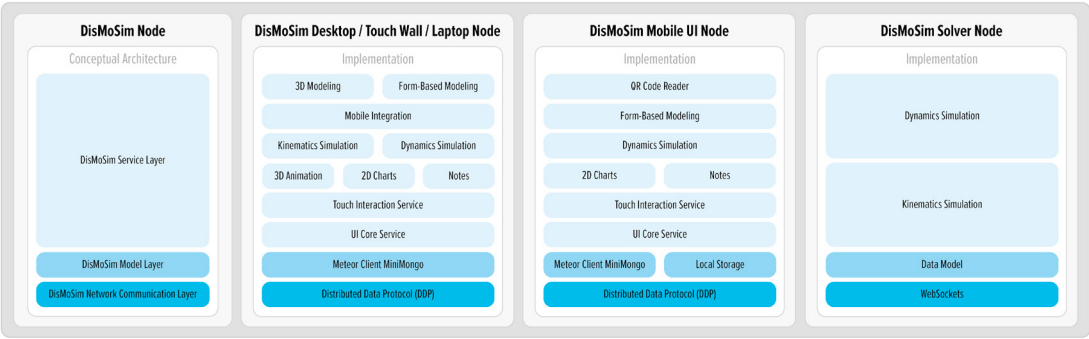

Fig. 3. Conceptual architecture of a DisMoSim node (1st column). Different DisMoSim nodes and how they implement and provide different services in the DisMoSim network (2nd to 3rd column).

WebSockets. The WebSocket[2] protocol is based on TCP and provides a bi-directional connection between two endpoints, e.g., to send a model to a simulation node and to receive a stream of simulation results. WebSockets support a persistent connection between client and server and both parties can start sending data at any time. WebSockets are characterized by a very low latency allowing near real-time communication.

Distributed Data Protocol (DDP). DDP uses WebSockets to enable (i) a bi-directional remote procedure calls between clients and servers, and (ii) client subscriptions to a set of server-side documents, with the server keeping clients informed about changes. Thus DDP allows to synchronize data across multiple clients by enabling clients to call any functions defined on the server, enabling near real-time collaboration scenarios. DDP is used in the Meteor[3] framework, which we use to implement shared data models for the DisMoSim Model Layer.

3.2.2. DisMoSim Model Layer

The DisMoSim Model Layer provides means for persistently storing data and for synchronizing it between different clients using Meteor's DDP for near real-time data synchronization. More specifically, Meteor relies on the NoSQL stores MongoDB[4] and Minimongo[5], which are accessed via so-called `Collection` objects. Collections are the primary persistence mechanism in Meteor and can be accessed from both, the server and the client. After changes, they update themselves automatically, i.e., a view component backed by a collection will automatically display the most up-to-date data. Every data change is first automatically stored in the client-side Minimongo which is then synchronized via collections and the underlying DDP with the Mongo database on the server. If the server accepts these changes, every client that listens to the changing collection is notified, its client-side database gets synchronized and the user interface gets updated. Therefore, this mechanism is especially useful for implementing collaborative workspaces and to control concurrent, and potentially conflicting user actions.

To illustrate how the DisMoSim Model Layer supports distributed simulation, we use the example of multibody simulation of mechanical systems. This is a method of numerical simulation in which a mechanical system, e.g., the aforementioned ATV, is represented as various bodies and their physical connections are modeled by kinematic constraints or force elements [14]. This enables a numerical simulation of future states of the system by repeated calculation of its changes after small time steps at a very high temporal resolution.

The object-oriented DisMoSim Data Model in Figure 4 contains the necessary entities to represent multibody systems for distributed modeling and simulation. `DisMoSimObjects` are divided into objects for modeling bodies (see class `RigidBody`), `Forces` (e.g., the rotational force on a crankshaft), `Constraints` (e.g., to limit degrees of freedom between the crankshaft and the con rod), or `Measures` (e.g., to measure some distance or angle between bodies). To give objects a 3D shape, `Visuals` may be attached to a `DisMoSimObject`, either by predefined visualizations (e.g., `BasicMeshBox`, primitives such as spheres or arrowheads) or by providing a 3D mesh in STL format (see `DisMoSimMesh`) that can be imported from CAD data. A `Workspace` acts as entry point for `Users` and repre-

[2] https://tools.ietf.org/html/rfc6455 (last accessed 17.06.2020)
[3] https://www.meteor.com/ (last accessed 17.06.2020)
[4] https://www.mongodb.com/ (last accessed 17.06.2020)
[5] https://github.com/slacy/minimongo (last accessed 17.06.2020)

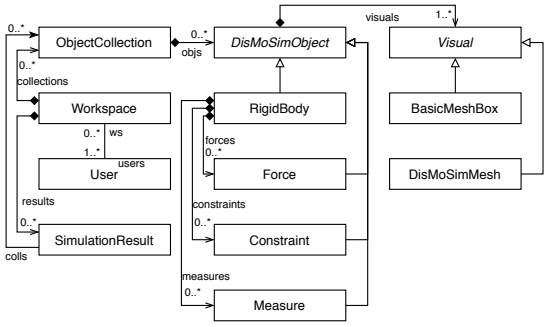

Fig. 4. Simplified DisMoSim Data Model in UML Notation

sents our concept of a *virtual workshop*. Only users within a workspace may edit the model. Within a `Workspace`, `ObjectCollections` of `DisMoSimObjects` can be used to form meaningful object groups (e.g., group "crankshaft"). During collaboration, the data model is kept synchronized between all clients via the DisMoSim server (see Figure 2) and is uploaded or updated before starting simulation processes on simulation nodes.

After completing a simulation run, a `SimulationResult` contains the state of all `DisMoSimObjects` at the time the simulation was started and for each subsequent time step of the simulation run. This enables the 3D visualization and time navigation of results inside a separate window. Additionally, it is possible to transfer simulated future states from `SimulationResult` into the current workspace as if its state had traveled forward in time. This also allows for live updates of the workspace by external nodes, e.g., real-time kinematics simulations during user manipulations or letting HIL and SIL nodes alter objects according to simulated outputs.

3.2.3. DisMoSim Service Layer

DisMoSim *services* are pieces of software that provide users with the necessary functionality to solve one of the tasks from the *service domain*. In our case, this domain is concerned with the distributed and collaborative modeling and simulation of a CPS, therefore requiring a great variety of different functionalities for solving very different tasks.

Some services strongly rely on user interactions and thus require a complex UI, e.g., the 3D modeling or visualization services that users can access on the UI nodes (see Figure 2). Other services are purely computational, e.g., simulations that are calculated on the Kinematics Simulation and Dynamics Simulation nodes. To provide such "invisible" services to users in a usable and controllable form, DisMoSim requires UI services that serve as frontends. However, behind the scenes, a user's interaction might transparently trigger distributed services on multiple physical nodes. This frees users from needing to understand on which physical machines (or nodes) a service is executed.

In general, DisMoSim services use the internet to efficiently share resources. For example, instead of installing and maintaining redundant UI code locally on all UI nodes, the DisMoSim Server node contains a single Web application that is downloaded to and executed by a web browser on UI nodes (see Figure 2). Besides, this functionality is loaded from the server asynchronously and lazily after additional functionality has been requested within the UI and only if the hardware and/or browser is suitable for it (principle of progressive enhancement known from web programming).

To ease the implementation of specific services, the service layer provides common utility methods to access the data model and common core services (e.g., authentication/authorization of users, handling of workspaces, or general methods to put data into or to read data from the corresponding Meteor collections). The details of the implementation and integration of specific services are discussed in the following section.

4. Implementation of Services

DisMoSim provides services for the core functionalities of (i) modeling and visualizing a CPS and (ii) integration of the simulations that are at the heart of any multibody simulation software. Additionally, (iii) collaboration and interaction services are provided to support means of hybrid collaboration.

4.1. Modeling and Visualization Services

Services in this category provide components for an easy-to-use UI for modeling mechanical systems. They are meant to run on any UI node, including desktop PCs, laptops, large interactive displays, as well as smartphones.

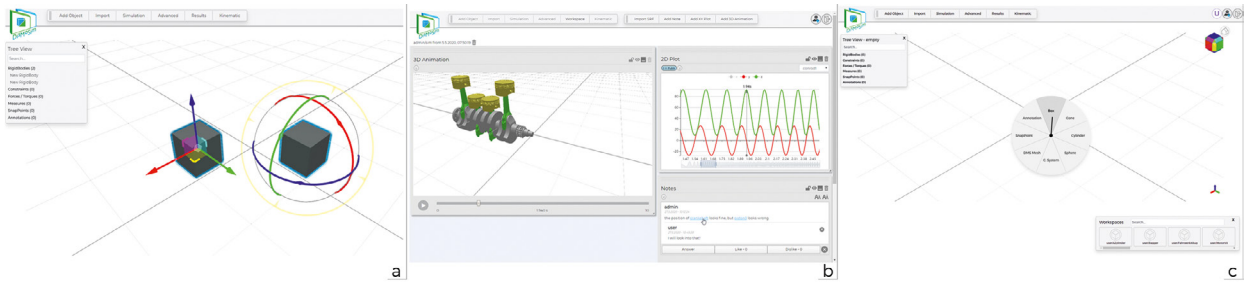

Fig. 5. Transformation controls for easing rotation and translation of 3D objects (a), Simulation results from the visualization service (b), Marking menus on a large interactive display (c)

UI Core Service. The UI Core service sets up core parts of Meteor needed on clients (e.g., the Meteor client database MiniMongo) as well as the minimal, responsive UI required on each UI node, into which other UI services can be integrated. Since collaboration is central to DisMoSim, it also provides UI components that deal with user management in general (e.g., login, authorization), handling of workspaces, and the assignment of users to specific workspaces. As a UI framework, we currently rely on React[6] because it supports our idea of separating parts of the UI into reusable components and because React can be easily integrated into the underlying Meteor framework. The UI is browser-based only and can thus run on any node capable of running a browser. In case of a mobile node, e.g., a smartphone, the UI core service includes means to deliver the application in terms of a Progressive Web App, i.e., a manifest, a corresponding service worker implementation, and implementations to receive push notifications. PWAs enhance the browser's feature-set by i.e., offering an icon on the smartphone's home screen to access the app fast and push notifications to receive updates, even when the app is closed. The user's personal preferences, i.e., if a push notification should be received, are stored directly on the phone via the `WebStorage` API[7].

Modeling Service. This service provides UI components for interactive 3D modeling of mechanical systems and can be divided into the following two sub-services.

Form-Based Modeling. The basic version of the UI is provided in terms of a form-based version that can be used to create, edit, and delete elements of the DisMoSim model layer to create multibody models for later simulation. By relying on the Meteor framework and the underlying DDP, elements within the model are automatically synchronized between multiple clients, enabling collaborative modeling on multiple devices.

3D Modeling. The 3D modeling service extends the form-based modeling service with a 3D representation of the multibody simulation model (see Figure 1a & b). The geometry can either be imported from a CAD system (STL file format) or designed from scratch. The ThreeJS[8] framework is used for rendering the model data in 3D, i.e., the rigid bodies and the corresponding meshes. To display a workspace, the service collects all its attached *DisMoSimObject*s and their corresponding *Visual*s. Depending on their type information, those objects are then rendered in the ThreeJS space, which users can freely navigate. The database is monitored for any additions, changes, and removals, which are going to be applied to the scene graph, while unaffected objects are left untouched. In addition to the forms provided by the above-mentioned service, the 3D modeling service also provides means to facilitate the rotation and transformation of the model by providing intuitive visual controls, as shown in Figure 5a.

As the displays of the UI nodes in Figure 2 (Touch Wall #1 & #2 and laptop) are large enough the 3D modeling service (which depends on the form-based modeling and the UI core service) is downloaded (see Figure 3). However, smartphones may only utilize the form-based modeling service due to its limited size and processing power (see Mobile UI Node in Figure 3).

Result Visualization Service. The visualization service, which depends on the core UI service, allows easy discussion of the simulation results, i.e., how to interpret the calculations returned by a simulation node. For this purpose, the current service includes the following components.

3D Animation. This component provides a visual representation of the simulation in the form of an animated 3D model using ThreeJS (see left in Figure 5b), allowing simulations to be "played back" to quickly verify their accuracy.

[6] https://reactjs.org/ (last accessed 17.06.2020)
[7] https://www.w3.org/TR/webstorage/ (last accessed 17.06.2020)
[8] https://threejs.org/ (last accessed 17.06.2020)

2D Charts. The 2D Charts component displays simulation results as line charts (see right in Figure 5b) and allows the selection of bodies or measurements to study their evolution over time, based on Apache ECharts[9]. Apache ECharts was selected due to its full support for custom events and its performance.

Notes. To document simulations, we provide textual notes similar to threads in forums (see bottom right in Figure 5b).

In contrast to the automatically synchronized model, visualization components can be selectively set to public or private, i.e., visualizations are either shared with other users and can be edited by them, or they are only visible and editable for their creator. This offers the possibility to configure components before they are presented in a discussion of the results. The spatial arrangement of the visualizations on the UI can be individually arranged in a grid layout to make optimum use of the display area provided by the device. Additionally, components can be configured per device, e.g., 3D animations may not be suitable for small mobile devices, but only on desktop nodes or large interactive displays (see Figure 3). The data to be displayed in the visualization can either (i) be received from a dynamics simulation node via WebSockets and written to the server-side database for later synchronization, or (ii) be loaded from a file. The data is sent/read in JSON and converted into corresponding DisMoSim model classes in a pre-processing step.

If components run together on a single node, user interactions between them are automatically synchronized. For example, if a point in time is selected in the 2D graphs, the animation is placed at that point in time, or bodies mentioned in a note can be clicked to select the results for that body in the 2D graph.

4.2. Simulation Services

The core of any multibody simulation software is the implementation of computation algorithms that solve equations of motion. These services run primarily on the simulation nodes of a DisMoSim network, while consumer-facing nodes are given means to start/stop and interact with simulations (see Figure 3).

Dynamics Simulation Service. The dynamics simulation service first encapsulates UI components to start/stop a simulation run (from simple start/stop buttons on desktop nodes to push notifications on mobile nodes). Several simulation runs of the same or a different model can run simultaneously due to the asynchronous, parallel computation. Second, it provides a wrapper around the computation algorithms written in C++. The core task of the simulation service is to continuously inform this software about changes in the model to be simulated (e.g., if a new constraint is added), since it has its own data model that enables optimized calculations. The wrapper can either receive the data via WebSockets or load it directly from the DisMoSim data model. Depending on the desired scenario and use case, the calculated results can either (i) be sent back to the DisMoSim server via WebSockets or (ii) be stored in a file. For example, if the results are written to the data model of the DisMoSim server, they can immediately be shared and edited by all users within the workspace, or they can be kept "private" and the user can decide to share them later.

Kinematics Simulation Service. The time integration of a multibody system done by the dynamics simulation service is time-consuming, i.e., the complete simulation of a crankshaft with pistons and piston rods, flywheel and further may take several hours. However, users want to easily test a model before time-consuming simulations, e.g., by dragging an element to see if associated model elements (via appropriate constraints) behave correctly. This can be done by a so-called kinematics simulation, where no differential equations are solved as it is necessary for the dynamics simulation. Similar to the dynamics simulation, this service provides a wrapper around the C++ implementation of the simulation computation. The kinematics simulation requires the dragged model element and the coordinates of the movement (either with the mouse or by touch interaction). The updated positions are calculated by the C++ software and must be visualized. Therefore, the service depends on the 3D Modeling component of the modeling service and is not available on mobile UI nodes. To ensure smooth user interaction, network latency should be reduced to a minimum, which is why we opted to use WebSockets. Furthermore, the kinematics solver can be used to assemble multiple different DisMoSim models to a single model in one workspace, realizing the proposed idea of a *virtual workshop*.

4.3. Interaction and Collaboration Services

To enable hybrid collaboration between engineering teams, specific forms of interaction and means of raising awareness among other users are provided. These services are meant to run on any UI node which fulfills the service requirements.

[9] https://echarts.apache.org/ (last accessed 17.06.2020)

Touch Interaction Service. The touch interaction service allows the user to manipulate the camera or the objects themselves utilizing touch gestures on nodes that provide a touch interface, e.g. smartphones or touch walls (see Figure 3). For full functionality the service depends on the 3D modeling service. A specific problem with large screens is that the menus are often arranged horizontally at the top of the screen. While this is common for desktop settings, it can be quite cumbersome on touch walls to reach certain menu items, especially if there are multiple users. Therefore, it would be more convenient if the menus appeared near the area the user is currently interacting with. For this purpose, so-called marking menus [13] were proposed, which should enable the user to make a menu selection either by opening a radial (or pie) menu (see Figure 5c) or by a straight marker in the direction of the desired menu item without opening the menu. Users navigate faster if they already know the pattern and can quickly draw a marker in the area on the touchscreen where they are working.

Mobile Integration and QR Code Reader Service. In addition to remote modeling (using the form-based modeling service), simulation management (start/stop dynamics simulations), or result visualization, the mobile phone can also be used as an input device at touch wall nodes in a co-located environment. To increase added value, the gyroscope sensor built into the mobile phone can be used as a rotatable remote control, i.e. rotation changes are mapped to the rotation values of a 3D element. Since this scenario assumes that the user is in front of the touch wall node, a corresponding QR code must be displayed within the touch wall UI and scanned with the smartphone.

Annotation Service. Shared annotations between users can be placed in both, the modeling service and the result visualization service, providing annotations related to a data point in the 3D representation of the data model or a results visualization component. The service depends on the 3D Modeling modeling service and is not available on mobile UI nodes.

Collaboration in the Result Visualization Service. To enable collaborative discussion of results in both co-localized and remote scenarios, the visualizations are synchronized across users with access to the workspace. This offers the possibility to present a previously configured result visualization to other users or to collaboratively analyze the data in real-time. To facilitate discussion, we provide a synchronized mouse pointer that allows seeing the mouse movements of remote workers.

5. Findings, Discussion, and Future Work

Based on our experiences during our implementation of the example scenario, we identified two generalizable main findings and relevant topics for future work.

Finding 1: Suitability of Web-technologies (DG1,2,4). The presented DisMoSim architecture is based on web technologies. Today's modern web frameworks reduce a lot of work for developers. Especially Meteor and its underlying DDP eases the otherwise complex implementation of an according synchronization mechanism. Additionally, web frameworks enable the efficient creation of UIs, independently of the actual node and display size. However, limitations still exist concerning access to hardware and performance of web apps. For example, it is currently impossible to identify different input pens in order to distinguishing between different users and their respective actions. Performance-wise, it should be noted that WebGL is expected to handle 3D contexts very similarly to native applications. Only minimal overhead is used to enable hardware acceleration of 3D web environments. However, large STL files (> 100 MB) can cause several seconds of loading time before they can be displayed, which can hinder smooth interaction. Finally, the incompatibility between browsers is still an issue, especially when using modern features (e.g., support for VR or AR) that are often only supported in the latest browser versions.

Finding 2: Multi- and Cross-Device Interaction (DG3). A key goal is the seamless integration of different devices with various display sizes (i.e., smartphones, laptops, large displays). A first user study investigated on cross-device interactions between smartphones and large displays on the one hand and work-specific device usage on the other hand. Especially when several people work together on large displays and have to enter complex data, interacting with smartphones is much more convenient. That is since direct touch interactions on large display cause space problems and are also felt more exhausting than interactions on the smartphone. Additionally, means for positioning the 3D models via the phone ease the modeling process since the correct position of the model elements can be found much faster. Participants pointed out that the use of the smartphone can also be helpful in collaborative environments since every user can use his/her mobile phone while the large display is used as a presentation tool, e.g., when discussing simulation results in a group. Everyone can set up the layout on their smartphone individually and look at

the results at his/her pace without standing directly in front of the large display. Finally, the study has shown that the near real-time communication between the nodes is crucial for working across devices.

Future Work 1: HIL and SIL integration. For the integration of HIL and SIL nodes we envision a bidirectional mapping between the data format delivered from specific test benches into the JSON format conforming to our DisMoSim data model. As a protocol, Websockets or plain socket communication should be used to write the data into the data model on the server. To allow for a seamless integration of HIL and SIL components into the modeling environment, e.g., if certain configuration parameters need to be set, the DisMoSim data model may be extended by subclassing the `HIL` or `SIL` classes respectively. Finally, also an according (3D) representation of the HIL and SIL elements on UI nodes needs to be provided.

Future Work 2: Augmented Reality (AR) Modeling Service. In order to further develop the idea of the virtual workshop, especially for co-located settings, we want to include an AR service which allows a much more realistic representation of the 3D model. In addition to the visualization itself, the research focus in a first step will be on the identification of according gestures and interaction patterns that allow a seamless modeling and rapid prototyping (by using the kinematics service) of the CPS. In a second step, we will focus on collaboration in AR settings to investigate if current research on collaboration may also be applied for collaboration in AR environments.

References

[1] Avellino, I., Fleury, C., Mackay, W.E., Beaudouin-Lafon, M., 2017. CamRay: Camera Arrays Support Remote Collaboration on Wall-Sized Displays, in: Proc. CHI '17, ACM. p. 6718–6729.

[2] Bai, H., Sasikumar, P., Yang, J., Billinghurst, M., 2020. A User Study on Mixed Reality Remote Collaboration with Eye Gaze and Hand Gesture Sharing, in: Proc. CHI '20, ACM. p. 1–13.

[3] Diaconescu, M., Wagner, G., 2015. Modeling and simulation of web-of-things systems part 1: Sensor nodes, in: Proceedings of the 2015 Winter Simulation Conference, IEEE Press. p. 3061–3072.

[4] Ellis, C.A., Gibbs, S.J., 1989. Concurrency Control in Groupware Systems. SIGMOD Rec. 18, 399–407.

[5] Fitzmaurice, G.W., Khan, A., Buxton, W., Kurtenbach, G., Balakrishnan, R., 2003. Sentient Data Access via a Diverse Society of Devices. Queue 1, 52–62.

[6] Gaul, H.D., 2001. Verteilte Produktentwicklung: Perspektiven und Modell zur Optimierung. Ph.D. thesis. Technical University of Munich.

[7] Gutwin, C., Greenberg, S., 2002. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. CSCW 11, 411–446.

[8] Jetter, H.C., Gerken, J., Zöllner, M., Reiterer, H., Milic-Frayling, N., 2011. Materializing the query with facet-streams: a hybrid surface for collaborative search on tabletops, in: Proc. CHI' 11, p. 3013.

[9] Jetter, H.C., Zöllner, M., Gerken, J., Reiterer, H., 2012. Design and Implementation of Post-WIMP Distributed User Interfaces with ZOIL. IJHCI 28, 737–747.

[10] Johansen, R., 1988. GroupWare: Computer Support for Business Teams. The Free Press, New York, NY, USA.

[11] Khaitan, S.K., McCalley, J.D., 2015. Design techniques and applications of cyberphysical systems: A survey. IEEE Systems Journal 9, 350–365.

[12] Klokmose, C.N., Eagan, J.R., Baader, S., Mackay, W., Beaudouin-Lafon, M., 2015. Webstrates: Shareable Dynamic Media, in: Proc. UIST '15, ACM. p. 280–290.

[13] Kurtenbach, G., Buxton, W., 1994. User learning and performance with marking menus, in: Proc. CHI '94, pp. 258–264.

[14] Nachbagauer, K., Sherif, K., Witteveen, W., 2015. Freedyn–a multibody simulation research code, in: Proc. 11th WCCM, 5th ECCM and 6th ECFD, p. 51.

[15] Neumayr, T., Jetter, H.C., Augstein, M., Friedl, J., Luger, T., 2018. Domino: A descriptive framework for hybrid collaboration and coupling styles in partially distributed teams. PACM 2, 1–24.

[16] Orts-Escolano, S., Rhemann, C., Fanello, S., Chang, W., Kowdle, A., Degtyarev, Y., Kim, D., Davidson, P.L., Khamis, S., Dou, M., et al., 2016. Holoportation: Virtual 3D Teleportation in Real-Time, in: Proc. UIST '16, ACM. p. 741–754.

[17] Rogers, Y., Lim, Y., Hazlewood, W.R., Marshall, P., 2009. Equal Opportunities: Do Shareable Interfaces Promote More Group Participation Than Single User Displays? Human–Computer Interaction 24, 79–116.

[18] Roseman, M., Greenberg, S., 1996. Building real-time groupware with groupkit, a groupware toolkit. ACM ToCHI 3, 66–106.

[19] Schuh, G., Potente, T., Wesch-Potente, C., Weber, A.R., Prote, J.P., 2014. Collaboration Mechanisms to Increase Productivity in the Context of Industrie 4.0. Procedia CIRP 19, 51 – 56.

[20] Stock, T., Seliger, G., 2016. Opportunities of Sustainable Manufacturing in Industry 4.0. Procedia CIRP 40, 536 – 541. 13th Global Conference on Sustainable Manufacturing – Decoupling Growth from Resource Use.

[21] Sutherland, I.E., 1963. Sketchpad: A Man-Machine Graphical Communication System, in: Proceedings of the May 21-23, 1963, Spring Joint Computer Conference, ACM. p. 329–346.

[22] Underkoffler, J., Ishii, H., 1999. Urp: A Luminous-Tangible Workbench for Urban Planning and Design, in: Proc. CHI '99, ACM. p. 386–393.