ICST Transactions on Collaborative Computing Research Article ICST.ORG

A Collaborative Virtual Workspace for Factory Configuration and Evaluation*

Ingo Zinnikus[1,*], Sergiy Byelozyorov[2], Xiaoqi Cao[1], Matthias Klusch[1], Christopher Krauss[1], Andreas Nonnengart[1], Torsten Spieldenner[1], Stefan Warwas[1], Philipp Slusallek[1]

[1]German Research Center for Artificial Intelligence GmbH, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
[2]Saarland University, Campus E1 1, 66123 Saarbrücken, Germany

Abstract

The convergence of information technologies (IT) has enabled the Digital Enterprise in which engineering, production planning, manufacturing and sales processes are supported by IT-based collaboration, simulation and enactment. As a result, borders between reality and its virtual representations become increasingly blurred. Advanced tools need to support flexibility, specialization and collaborative evolution of the design where the exchange of knowledge between domain experts helps to improve informed decision making. In this paper, we present a collaborative, synchronized web-based framework to create 3D scenarios for product design, simulation and training assisted by animated avatars.

Received on 01 February 2014; accepted on 23 April 2014; published on 27 May 2014

Keywords: Collaborative computing architectures and networks, Computer supported collaborative work, Web-based collaboration, Visualization techniques for collaborative networks and applications, Service-oriented architectures for collaborative networking and applications

doi:10.4108/cc.1.1.e5

1. Introduction

In recent years the trend towards networked organizations where businesses and companies are working (often remotely) together has been intensified [2]. Activities in networked organizations consist of distributed processes which include communication, exchange of resources and joint production of business artefacts. These artefacts now include assets such as product designs which in the past were kept under control of one company. The co-innovation and co-design of product families and production plants with a large number of part suppliers requires precise adjustment between contributing partners.

Supporting this precise adjustment of products and production facilities requires IT Systems for integrated product development. Engineering these integrated and interactive systems where interaction involves cooperation and collaboration of a potentially large number of contributors with complementary skills e.g in product assembly design. This is an inherently collaborative process where multidisciplinary experts from different areas and geographically remote locations contribute to one product or resource.

Regarding collaborative engineering, Booch and Brown [3] highlighted the importance of a Collaborative Development Environment defined as: 'a virtual space wherein all the stakeholders of a project - even if distributed by time or distance - may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to create an executable deliverable and its supporting artifacts' (see especially [4]). Empirical investigations confirm this importance [5], [6].

Another recent trend is the usage of virtual techniques for product and production facility development which allows designing 3D representations that capture selected traits of products and production plants. 3D prototypes can be used to evaluate products in advance e.g. in order to reduce costs and time to market. Together with this trend towards virtual prototyping based on 3D design and engineering, a combined research area arises: collaborative virtual prototyping.

*Extended version of [1]
*Corresponding author. Email: ingo.zinnikus@dfki.de

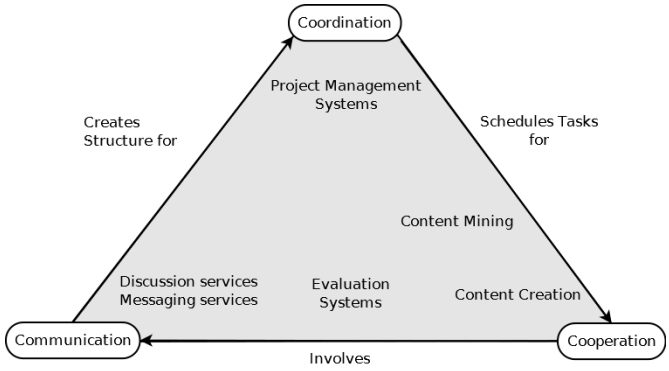

Figure 1. Iteration cycle of collaborative work based on the 3C model as stated by Fuks et al. [11] and required services for each step as stated by [12]

As cooperation, coordination and collaboration are often used interchangeably, a more fine-grained distinction between these concepts is required. There are two major lines of research which differ in the details of defining the relation of these concepts (contrasting vs. encompassing). One (older) approach stemming from organisation science defines the relation as continuum where cooperation falls on the low end and collaboration on the high, with coordination in-between [7], [8]. Cooperation is defined as the least formal interaction, based e.g. on a simple verbal agreement. In cooperative work, the division of labour leads to processes where each person is responsible for a portion of a problem solving [9]. Cooperation becomes coordinated when the informal problem solving process is following an explicit or implicit joint planning where responsibilities and roles are assigned. Collaboration in a contrasting and narrow sense is the most formal interorganizational relationship involving shared authority and responsibility for planning, implementation, and evaluation of a joint effort and involves the "mutual engagement of participants in a coordinated effort to solve the problem together" [9].

In contrast, the other main approach on which we base our work defines collaboration as the encompassing process with cooperation, coordination and communication as its ingredients. Ellis et al. describe collaborative work as an iterative process of *communication*, *coordination* and *cooperation* (3C model [10], see Fig. 1). Based on this model, Fuks et al. describe collaborative work as an iterative, cyclic process [11]: Communication describes spontaneous exchange of information between team members, like for example in the planning phase of a project. Based on the communication, future work is coordinated in tasks which are then cooperatively accomplished. During the cooperation step new issues will arise. Those need to be discussed and therefore lead to a new iteration cycle until the project work is finished.

In general, design tasks are often *ill-structured problems* [13]. In contrast to well-structured problems, where options and goals are clearly defined, in ill-structured problems the options available and possibly even the goals are unclear and vague. For many ill-structured problems, generic routines and procedures for problem solving do not exist. Although considerable amount of research has been devoted to identify recurrent collaboration patterns (e.g. [14]), for ill-structured problems, a normative approach prescribing sequences of steps to solve the problem is not feasible. A collaborative engineering system for design tasks should be open to support different problem-solving activities and strategies.

Nevertheless, a common feature of design tasks is that they involve an iterative process of interactive decision making and model building [13], thus confirming the adequacy of the 3C model. To support this iterative process, a "wide range of sophisticated communication and content services" [12] are necessary. Examples include "customized **discussion services**, **workflow and knowledge management systems**, and content **creation, mining, and retrieval services**. [..] To create useful collaborative virtual working environments for communities, collaboration requirements need to be linked to services satisfying those requirements" [12].

Collaboration in 3D design and virtual environments has additional and specific requirements. Traditional collaboration systems already provide the possibility to jointly work on designing artefacts. In 3D environments, the central artefact is a *visible virtual model* of reality. As *visible* 3D artefact, the model is intended to be (possibly jointly) looked at. As *virtual* (in contrast to a physical mock-up) *model*, it is an archetype which represents selected and supposedly relevant features of the real resource(s) to be produced. In virtual prototyping of products and factories, these features include functional interdependencies and other aspects such as ergonomy which can be simulated or even aesthetic qualities. As in using physical mock-ups, these functional and non-functional features and aspects in virtual models can be evaluated before the resource is actually produced. The joint and simultaneous evaluation of these visible virtual models is a possibility which needs to be supported in a much more refined way, especially when the visual experience should be shared in realtime. Hence *efficient synchronisation* of many clients in a collaborative setting is a key issue.

Whereas many 3D systems present the final view of a product, a collaborative virtual prototyping system allows jointly creating, viewing, reviewing, modifying 3D artefacts and discussing alternative designs at all stages of the design process. The possibility to make changes in realtime as well as support for content creation and distributed storage retrieval are further requirements. In contrast to many commercial systems

which often are monolithic and oversized, the system should be easily extendable and quickly adaptable to different use cases and scenarios.

2. Objectives

We developed *Collaborate3D*, a web-based collaborative development environment for virtual prototyping that supports and realizes the three iteration steps based on the 3C model by Ellis [10]. Collaborate3D provides collaborative workspaces with native support for communication, cooperation and coordination and enables a shared visual experience of the creation, modification and evaluation of a virtual 3D design.

Since in different contexts and scenarios, a variety of components and modules are needed, the system is intended to be extendable and customizable. Instead of a one size fits all approach (which is prevalent especially in the commercial tools), we developed a service-oriented platform which supports customizing and adding services as plugins when required. The result is a modularized and configurable collaboration architecture for 3D scene editing, evaluation and simulation.

Synchronisation of plugin services can be achieved directly in the Web-client, if the services provide an appropriate interface, like WebSocket or HTTP REST. But there are cases in which client-side integration is not a suitable approach. This may be the case when one or more of the attached services have to send confidential data to the integration point. In the case of client-side integration, that means that users may get access to sensible data. Apart from that, processing data from services on client side puts additional workload on the client application. This can be an issue when the client Web-site is accessed from a mobile device. Those usually have significantly less computation power than a work station. In order to support mobile devices with limited capacities and ensure security of data, we introduce a server-based synchronisation approach which allows distributing a scene over a large number of clients in realtime.

Taking into account that design is an ill-structured problem, a specific configuration of system modules and services is open for different problem-solving strategies. The system is designed to be adaptable to a changing collaboration and task model and to support the whole continuum of cooperation and collaboration.

Apart from these general objectives, for implementing an effective collaborative workspace for virtual 3D prototyping, further goals include in detail:

- Ubiquitous accessibility via Web Page like implementation (central point of access)
- Simple inclusion of new 3D content and fast attachment of new code
- Flexible adaption to different use-case scenarios by plugin system for service integration
- Attach established (open) project management including issue tracker tools for asynchronous collaboration.

The application scenario is based on a factory production line where collaborators design and evaluate a factory module. Evaluation in this case consists of checking functional features such as production capacities, velocities and safety properties. Additionally, activities of workers can be modelled and their performance simulated and evaluated according to different criteria.

The structure of the paper is as following. In section 3 we give an overview of the architecture of the web-based system for collaborative prototyping, the XML3D-based layer for visualisation and content creation, as well as the distributed search technologies for content retrieval. The verification service and agent technologies for evaluation are presented in section 4. In section 5, we refine the Collaborate3D architecture for enabling efficient synchronization of clients in a collaborative setting. In section 6, we describe technical details of the implementation of the workspace components. We analyze and measure the performance of the synchronisation architecture in section 7. Related work is discussed in section 8. We conclude and describe future work in section 9.

3. 3D Collaboration Framework: Architecture

Collaborative virtual prototyping calls for a fusion of several technologies in order to provide a shared visual experience of the design artefacts. Web and Web service technology is the up-to-date solution to enable distributed access and real-time sharing of contents and resources.

The Collaborate3D architecture (see Fig. 2) reflects the idea of introducing service orientation into the virtual environment. Platform components needed for the realisation of the virtual environment are encapsulated as services hosted in the Collaborate3D service cloud. The architecture consists of three layers: a project related collaborative workspaces with a 3D editor for creating and modifying content, a service cloud with application specific services and a storage layer. The 3D editor is based on an web-based framework for rendering 3D graphics. In the current implementation the service cloud consists of a verification component for evaluation of functional properties and an agent platform for evaluating dependencies when e.g. human workers are interacting with devices in production plants. Other components could be a physics engine, kinematics, etc. A distributed repository for content storage and retrieval provides

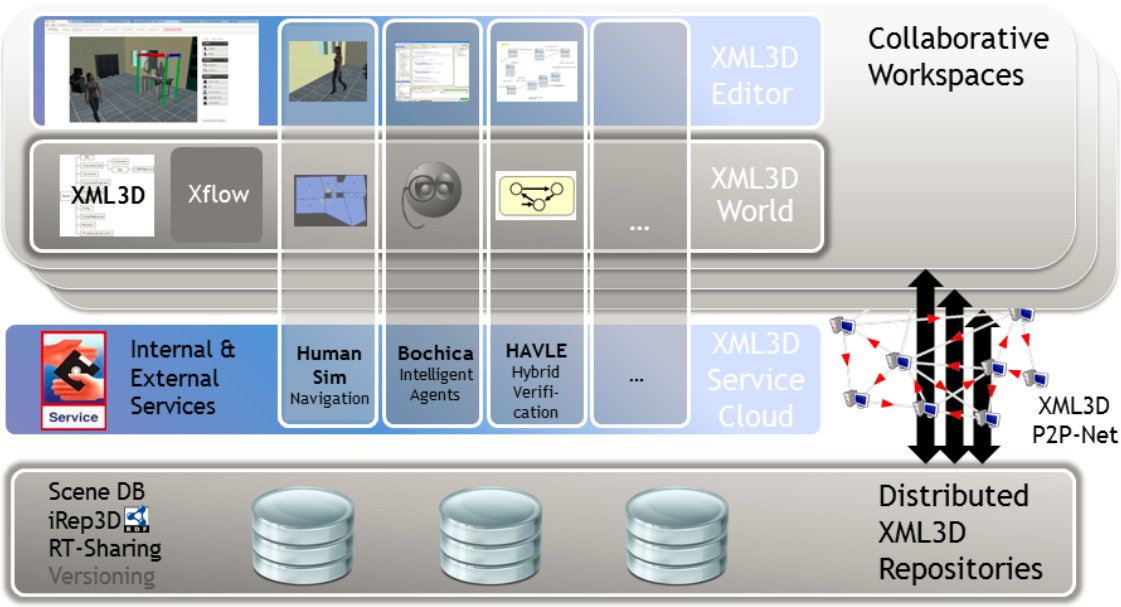

Figure 2. Collaborate 3D architecture

access to semantically annotated 3D models which can be inserted into a scene.

3.1. XML3D World

The collaboratively created 3D artefacts are contained in the data layer which is based on the XML3D specification. XML3D is an open declarative XML format that extends the set of HTML web page elements by additional nodes to represent 3D graphics in a scene graph like structure [15]. All nodes within this graph are also nodes in the web sites DOM tree representation (*Document Object Model*, [16]) and can be accessed and changed via JavaScript like any other common DOM elements as well. On these DOM nodes, HTML events can be registered similar to known HTML elements. Resources for mesh data can be stored externally in either JSON or XML format and referenced by their URL. The renderer that is used by XML3D is based on WebGL.

In addition to XML3D, *Xflow* allows to combine the scene graph with dataflows [17]. Xflow is a declarative data flow representation that was designed for complex computations on XML3D elements. These computations include for example skinned meshes and key frame animations. In these cases, the current key frame takes the role of a data source in the graph, whereas mesh transformations are sinks in the dataflow. By this, changing the value of a key frame leads to a change in the posture of a mesh, and thus a continuous change of the key frame over time results in an animated mesh.

By using XML3D with Xflow as foundation for data layer, we achieve both ubiquitous accessibility and high customizability: all scripts needed to display and use the 3D editor are automatically loaded by the browser as soon as the user opens the page in any web browser that supports WebGL. Furthermore, 3D models used for the scenario can be referenced by their URL from remote storage locations.

For evaluation services to be able to operate directly on the artefacts created in the editor, they must be able to access the data layer that contains the 3D artefacts. This access is directly provided by having both Xflow parameters and the 3D scene graph represented as part of the DOM tree by XML3D. External services can e.g. animate virtual characters by changing key frame values of Xflow graphs (in the case of the agent platform service), to generate the navigation mesh (using the geometry information contained in the DOM tree) or verifying the functional correctness of a module and display the trace witness via Xflow animation. The guiding principle for services such as the verification is that the functional specification of a composite module is based on the functional specifications of the parts contained which build up the composite module. The formal specifications of the parts are contained in the scene graph as annotations of objects.

3.2. XML3D Editor

The 3D artefact itself is created in the interactive 3D editor which is a web-based tool for creating and editing 3D scenarios. Those scenarios consist of a static base geometry (e.g. an empty factory hall) and several sets

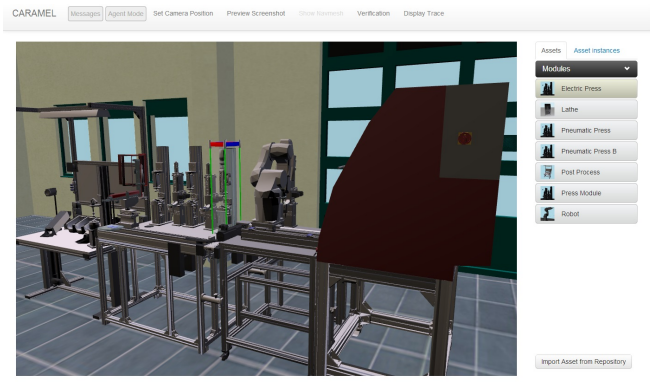

Figure 3. Editor tool

of 3D *assets*. Assets can be added, moved within or removed from the scene with simple drag and drop operations. Figure 3 shows the user interface of the editor: The interactive 3D scene is displayed on the left-hand side. Next to it is the sidebar which contains the selection of 3D assets that can be added, as well as already placed assets in a second tab.

Scenarios that share a common static geometry and the same set of assets are grouped in *projects*. In terms of the factory example above, this allows to create and store different configurations (*scenarios*) for the same factory hall (with the factory setting being the *project* in this case). The editor itself is part of a collaborative work space which allows a group of designers and domain experts to jointly construct and visit a 3D scene.

To provide seamless communication with collaborators during content creation, a basic personal messaging system is integrated into the editor. Using this system, team members can send each others short pieces of text. New messages are indicated as soon as a user enters the scenario in the scope of which the message was sent.

In addition, third party ticket and bug tracker systems that provide a REST API [18] for HTTP access (as for example *Mantis*[1] or *Trac*[2]) can be integrated into the work space. The step of actually choosing and integrating the tracker system in the overall application is not possible from the graphical user interface of the work space yet. However, the editor code is easily extendable via a provided API for third party services. As the editor is entirely written in JavaScript, and both 2D user interface as well as 3D elements are represented as DOM elements within this website, this extension can also be done by web designers with minor experience in website scripting. Information from third party system is them displayed directly in the work space's user interface.

[1] http://www.mantisbt.org
[2] http://trac.edgewall.org/

Whenever more than one collaborator logs into the same scenario, the process of content creation turns into a synchronous editing session. Changes done to the scene by one user (including adding objects, moving existing objects or deleting them) triggers an update of the scene state in all connected clients. We employ two approaches to avoid update conflicts during concurrent editing. On client side, we use a locking mechanism: An object that is selected by one user is locked for editing for all his collaborators. This lock is indicated in both the 3D view of the scene by rendering the object semi transparent, and in the list of placed assets next to the editing window: list entries of locked objects carry a lock icon and are highlighted in red. In addition, the database performs a version control based on revision numbers for each object. If an update is performed on an object with outdated revision, the update request is declined and the sending client is informed about the conflict.

External services from the XML3D Service Cloud (see also section 4) can be accessed directly from the editor's user interface.

3.3. Storage Layer

A collaborative work space is composed of different types of data (see Figure 4): first, an abstract representation of a scene that was created in the editor as described in the previous section. Second, 3D assets that contain the actual geometry and texture information of 3D objects from which the resulting 3D artefact is composed. Third, information for coordinated collaboration like for example open tasks or deadlines. For each type of data, we need a repository that provides the respective data to the work space.

Scene Database. Data about projects and scenarios that are created with the scenario editor are stored in a database that is directly connected to the editor. This project data includes projects and related scenarios, 3D asset sets and abstract representations of placed 3D objects, including an object's position and the URI that references its 3D asset data in XML3D format on the resource repository. Each object may moreover carry configuration parameters for attached services. Changes in data are automatically transmitted to all clients that are currently connected to a scenario that is affected by these changes to provide synchronous, collaborative editing of a scenario.

Semantic 3D Asset Repository. 3D artefacts are assembled from 3D assets, available in *iRep3D* [19]. iRep3D is a repository for hybrid semantic indexing and retrieval of annotated 3D scenes in X3D, XML3D and COLLADA at any level of granularity, in near real-time and with high precision. iRep3D comes with

a web-based user interface which supports the user in annotating 3D scenes with plain text as well as appropriate ontology-based concepts and services in order to describe the functionality of scenes and their objects [20].

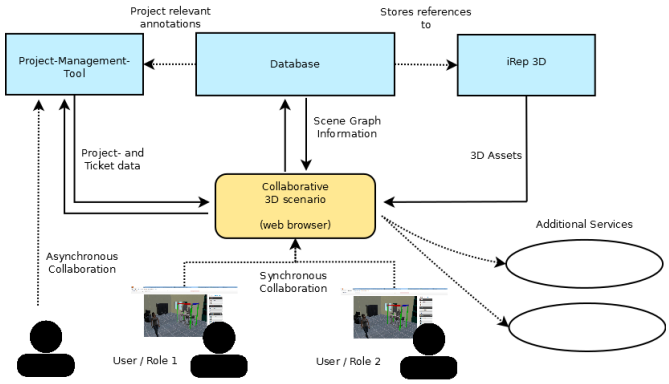

Figure 4. Storage and Service Assembly serving the XML3D editor

iRep3D performs off-line semantic indexing and on-line query processing. The indexing process determines the relevant score of a stored 3D scene based on its semantic annotation and geometric features for the (a) scene concept index, (b) semantic service index, and (c) geometric index of iRep3D. The semantic similarity between scene concepts in standard OWL2 bases on their approximated logical subsumption relation and its information-theoretic valuation. The matching score for pairs of semantic services in OWL-S each of which describing the functionality of scene objects are computed with the currently most precise service selection tool iSeM [21]. The geometric feature index of a scene is a set of B+ trees each of which represents a standard feature-attribute pair which is instantiated by the scene according to the X3D, XML3D and COLLADA specifications, while geometric feature matching by iRep3D relies on classical approaches for this purpose. The repository applies a breadth-first-traverse-based pruning heuristic to efficiently maintain its scene indices in case the set of indexed scenes or their annotations change.

A query is a scene (in X3D, XML3D) that is annotated with the desired semantic scene concept, semantic services, and geometric features. iRep3D answers such queries by means of a parallel index-based subquery processing and final aggregation of resulting rank lists with a classical threshold algorithm. Besides, its search for relevant scene objects is performed even within indexed 3D scenes at any level of granularity while positive results are extracted and indexed as new scenes for further re-use, while the top most relevant scenes are then displayed to the user together with provenance information.

Indexed 3D scenes in XML3D, X3D or COLLADA are stored internally in a native XML database while the iRep3D repository itself provides a web-based user interface including a semantic annotation toolkit, and is also accessible via a REST API. Our experimental evaluation based on a test collection 3DS-TC 1.0 with more than 600 annotated XML-based 3D scenes revealed that iRep3D is significantly more precise and with the same average response time than its relevant and open-source competitors like Trimble3D, ADL and the Princeton3D search engine.

External project management systems. Data for work coordination is provided by external project management systems. This meta data includes user groups, messages that were sent in the scope of a project, issues and bug trackers. Those systems can also take the role of user authentication, if provided by the tool. Project management or bug trackers are connected to the work space via REST API. All data is displayed directly in the editor GUI and thus available during synchronous editing sessions. The benefit of using an external tool instead of using the scene database to store the meta data directly with the scene data is the asynchronous accessibility of the data from outside the editor. Existing tools like Trac, Mantis or Redmine come with a front-end GUI that can be operated from a Web-browser. That allows to contribute to project coordination steps without having to use the actual 3D design tool.

4. XML3D Service Cloud

The XML3D service cloud contains several services which provide scenario-related simulation and evaluation functionality. The service cloud can be extended and new services can be added as plugins. In the current implementation, a verification service for functional evaluation of factory modules and an agent platform for controlling avatars are included.

4.1. Collaborative Evaluation through Formal Analysis

Physical systems that are controlled by embedded software like flight control systems, automatic breaking systems, and production lines in factory environments are called hybrid systems as they involve both discrete and continuous behavior. A major goal in the design and implementation of hybrid systems is the ability to reliably verify functional properties of the system at hand. Flaws in the design of such complex systems occur regularly, especially when created in a collaboration of many different specialists. If they remain undetected and manifest themselves in the final implementation of the system they may lead to

severe malfunctioning resulting in loss of money and reputation or even worse, injury or loss of life. A solution to this problem is the use of formal methods for a semantically unambiguous modeling of systems and their verification.

In our implementation we included a verification module that allows the collaborative evaluation of the system modeled in the 3D Editor (see section 6.1). In communication with a collaborating specialist for formal methods and verification the system designer can formulate and verify the requirements for the system at hand. A tight integration of the formal and the 3D model allows a (partially) automatic generation of the formal model. On the other hand, the tight integration also enables the results from the formal analysis performed by the verification specialist to be presented to the designer in a generally understandable manner. The verification itself is performed by the verification tool HAVLE[3] provided through a REST service.

Hybrid Automata. HAVLE uses Hybrid Automata as they are a language particularly well suited to formally model hybrid systems in that they allow to specify both the continuous and discrete behavior parts of the system in one model. Our version of hybrid automata is very similar to the language of rectangular hybrid automata as they are known from [24], however with some extensions to provide an easy to use and extensive approach to model hybrid systems. Their main contribution is the support of a high degree of modularity. All possible system behaviors are defined by the composition of the different components. Since the language of hybrid automata is a formal one with formal semantics, verification is possible on systems specified in this language. For a detailed introduction into hybrid automata we refer to [23].

The Verification Module. Figure 5 depicts the general workflow of the evaluation of a system. Models of the system can be constructed in the 3D Editor by putting together component parts coming from a library of components. To allow verification and at the same time provide a seamless integration, additionally to the 3D description of their looks and physical dimension, they come with a formal description of their possible behavior. Adding a component to the model in the 3D Editor implicitly also adds the hybrid automaton assigned to this component to the formal model. A mapping that also comes with the component describes how the parameters of the 3D object correlate with the parameters and the initial states of the formal model. It translates position and rotation of the 3D object into values for the template parameters and initial locations

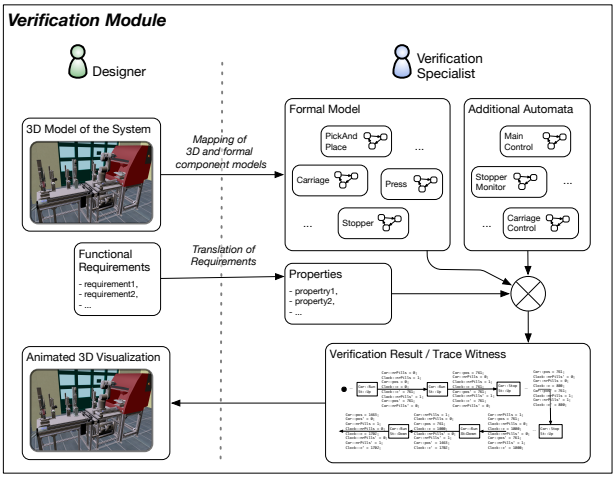

Figure 5. Collaborative evaluation

of the automaton template. Applying this for all objects added to the system allows us to automatically generate a projection of the formal model from the system modeled in the 3D editor onto its visual parts.

After constructing the system the designer can ask the verification specialist using the available communication mechanisms, e.g. the messaging system presented in section 3.2, to verify certain requirements given for the designed system. Due to the shared workspace and the included seamless synchronization the verification specialist can immediately access the current model of the system. If required (or desired) she can adapt the automatically generated formal model by, e.g. adding automata describing the non-visual parts like controllers or automata for the interrelation between the physical parts. She then formalizes the required properties submitted by the designer and verifies them.

Depending on the verification result (in case of the rejection of a safety property or the approval of a reachability property) HAVLE provides a *trace witness* (out of a constructive proof). Essentially a trace is a formal description of the behavior that the system has to show to reach a certain desired or undesired state. However a formal trace is very hard to read and can only be understood by specialists.

Due to the tight integration of the formal and the 3D model through the verification module we are able to map such traces to animated visualizations of the behavior[4] described by the trace in the 3D model of the system. Using basic modularized *Xflow* keyframe animations we perform a stepwise interpretation of the formal trace as pictures in an animation sequence. As

[3] Hybrid Automata Verification by Location Elimination [22, 23].

[4] In fact the projection of the trace onto the physical components occurring in the 3D system.

soon as the verification has been performed by the verification specialist the visualization is also available for the designer and provides it with a tangible feedback of the performed verification.

Apart from collaborating with a specialist the designer can also formulate the properties to be proven by himself or choose from an existing list of required properties[5]. This is especially useful in case a previous version of the system that had already been verified has been modified. Similar to the idea of unit testing, by so called unit verification one can verify after every modification whether the system is still correct. Modifications like adding, removing, or moving of components are directly reflected in the formal model and need no further adaptations.

A more detailed description of the 3D visualization of verification results, the automatic generation of formal models from 3D models can be found in [23].

4.2. Agent-based avatars

In order to simulate interactions between workers and the production assets, the editor allows positioning animated avatars into the scene (see Figure 6). The behaviour of avatars is controlled by an agent platform which is provided as a service. Agents in our context are the abstract entities representing the avatars in the scene. Agent models containing the behaviour can be modelled in advance and assigned to virtual characters. For modeling agent behavior, we use the Jadex agent platform[6]. The agent platform is based on the BDI (belief - desire - intention) [25, 26] approach for describing agent behavior. The BDI approach with its incorporation of reactive and goal-based behavior is especially appropriate for controlling avatars, because avatars in a 3D scene need to react quickly to a changing environment while exposing goal-directed behavior at the same time.

When a user wants to place an avatar into the scene, first the avatar asset is selected. After placing the avatar into the scene, one or more behavior capabilities behavior can be assigned to the avatar. The available capabilities are based on the agent behaviors which are provided by the agent platform. By asssigning concrete goals (the intention) to an agent, the corresponding avatar tries to achieve this goal according to the behavior description. Agent-controlled avatars can be used to simulate workers, e.g. for evaluating the reachability of modules, time constraints for production processes, etc.

Since the objects in the scene can be moved using the editor, they constitute possible obstacles for avatars.

[5]Possibly formulated before the system was even constructed resulting from a requirements analysis.
[6]http://jadex-agents.informatik.uni-hamburg.de

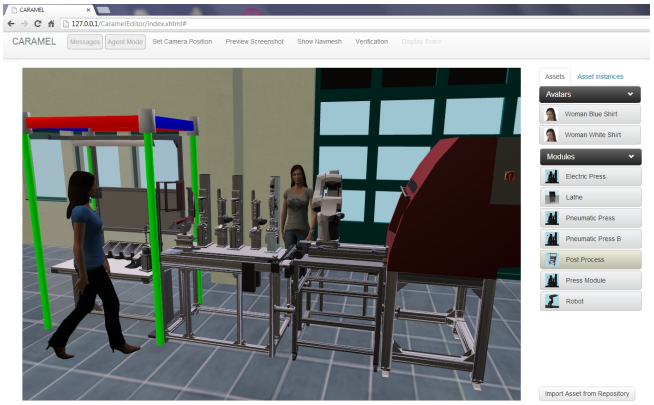

Figure 6. Editor for agent-controlled avatars

Therefore, in the framework, a navigation service is associated to the agent platform which generates a navigation mesh on demand each time the scene layout is changed and prevents collisions of the avatars with each other and objects in the scene. The editing and simulation of agent activities can be done collaboratively in the shared workspace.

5. Synchronization and Integration Server

As mentioned in section 2, the integration of the presented services can be achieved directly in the Web-client, if the services provide an appropriate interface. Since there are cases in which client-side integration is no suitable approach, e.g. when data to be sent to a client is confidential or the capacity of client devices are restricted (in the case of mobile devices), we have developed and implemented a server-side synchronisation solution to address these issues.

The integration server is designed in a way that allows for easy extension by third party services. The design opts at the possibility to not only have different services run in parallel, but also make data from one service accessible to another.

This also makes integrating the services into the clients easier by wrapping the different interfaces that are provided by the different services into one homogeneous interface.

A server-based solution moreover enables shared synchronized simulations of service data, for example visualization of verification results, for several connected clients.

5.1. High-Level architecture

The overall architecture of the integration platform aims at a very slim core with no domain specific functionality. Instead, the server core maintains a generic data representation of the constructed 3D asset and means to synchronize that data to connected clients (see Figure 7). This data model as well as

the functionality of the server can be extended by a number of additional modules or *plug-ins*. Whenever a new module is added to the server runtime, both its introduced data and functions are also made available to other connected modules. In the case of our presented factory design and evaluation workspace, those modules may be plug-ins for agent behavior simulation, formal verification or messaging between connected users. Modules can implement new features from the scratch or wrap existing services to a common API. For the example of formal verification, this would mean that the respective module connects to the existing verification service. Clients in turn do not operate directly on the verification's REST API anymore, but communicate with the integration server which in turn processes the messages to the verification service via the attached module.

5.2. Entity Component Attribute World Model

We have decided for a world model that on the one hand makes as few assumptions about the intended use case domain as possible, and on the other hand lets developers and users extend it by additional data that may be very specific concerning their intended use. For this, we have employed an *Entity-Component-Attribute (ECA)* [31] model to represent assets in the 3D scene on the server.

According to this model, all objects in the world are considered as *Entities*. This includes not only placeable assets, but also abstract concepts such as agent behavior definitions or abstract scene descriptions that may for example occur from a set of project parameters.

An Entity does not contain any data, but is rather an empty container with an ID that may be filled with data of any kind. Providing data is done by attaching *Components* to an entity, which in turn are containers for sets of typed *Attributes*. Both Components and Attributes can be accessed via a string-typed name on the entity. By this, the position of a placeable asset may for example be accessed via *entity["placeable"]["position"]*. For attribute types, we currently allow basic data types (int, string, float ...) as well as structs and lists from these types.

These components and the definition of the set of attributes they introduce are provided by specific plugins. As described in Section 5.1, all components that are registered in the server core are accessible from any other plugin. That is, when running the server with both plugins for agent behavior simulation and verification loaded, verification may access the position of agents and current states in their plans to include them into the verification procedure.

5.3. Unified Service Interface

When integrating different services directly on client-side, we face the drawback that for each attached service a connection needs to be opened that follows the API specification of the respective service. In our use case presented that includes the REST services to the database and HAVLE, the WebSocket connection to HumanSim service and the reference to the web interface of IRep3D. Even though the JavaScript client code and the tight coupling of XML3D to DOM scripting simplifies inclusion of such services for client developers, we will end up with a rather unflexible client, as adapting to another application domain means removing the respective connections from the client.

Drawing the step of service integration into the server level gives us the possibility to introduce a homogeneous API provided by the synchronization server. In the current server design, we employ a Remote Procedure Call framework to expose server-side functions as *service functions*, grouped in *service* which take the role of a namespace. We refer to the complete set of services that are provided by the server as *service API*.

Plugins may expose their own functions as service by registering them to the server core. They are then accessible in the format *service.servicefunction*, e.g. *verification.runTraceVisualization*. When the client starts up, it retrieves the list of provided services from the server and may wrap those into JavaScript functions. Calling the function on the server then amounts to a simple JavaScript function call, e.g. the service for trace visualization may be invoked by the client by a call like *runTraceVisualization(assetID)* as if it was a function of the client itself. The list of parameters expected for the service call (in the last example, the ID of the asset for which the visualization should be run) is defined on the server by the native function that is registered for the service.

6. Implementation

6.1. 3D Editor

To fulfill the requirement of ubiquitous accessibility, the implementation of the editor tool is completely based on XML3D with Xflow and JavaScript. The implementation of the web editor client uses the *Backbone.js* framework [27] to render both the 2D and 3D content from the data in the database. Backbone.js implements the *Model-View-Controller* (MVC) pattern [28]. For each data object in the data base (*model*), backbone creates DOM elements (*view*) to render for example both the XML3D element of an asset instance in the 3D scene and the respective entry in the list of asset instances on the right hand side of the editor view.

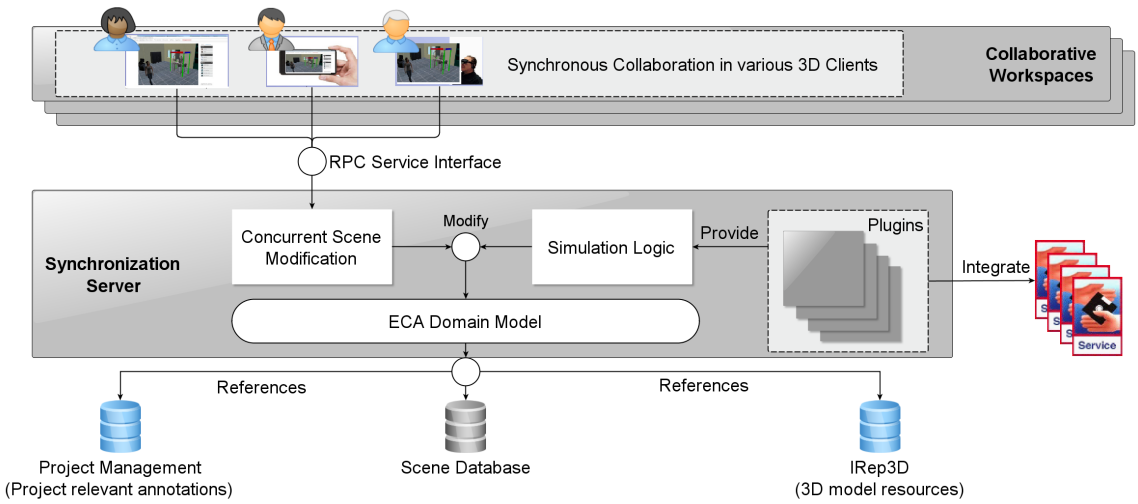

Figure 7. Collaborative Workspace architecture with employed synchronization server

We chose Apache's CouchDB[7] as storage unit for scene data. This NoSQL ("not only relational") database stores data as key-value-pairs with no fixed scheme. These values can be hierarchical objects with a number of attributes, usually accessed by a unique ID.

CouchDB adds a revision number for each object to avoid update conflicts when data is accessed by numerous users [29]. When a request to save data is sent to CouchDB, the request has to include the latest revision number of the updated data set known to the submitting client. If the revision number is different from the one stored in the database, the data update is rejected to avoid conflicts from concurrent updates.

CouchDB provides a RESTful web service to query data. It returns data as JSON (*JavaScript Object Notation*) objects. These objects can directly be used in a browser application's JavaScript code, without any overhead to parse the data. We keep the number of requests needed to keep in sync with the database low by using *long polls*. Requests that are sent as long polls do not return immediately, but remain at the receiving entity (in our case, the database), until the requested data is available. This approach reduces the work load in the browser by the reduced number of database polls. In addition, once a long poll is returned, the delivered changes can directly be used to change displayed data accordingly. This keeps the displayed system state in the application consistent with the actual data stored in the database. The database is accessed by both editor and services from the XML3D service cloud.

3D assets are stored and managed by iRep3D. In order to import a new 3D asset to a project in the editor, the user opens iRep3D from within the editor GUI and queries for the desired asset. Once he concludes the import by mouse click, iRep3D adds the reference URL of the respective 3D asset to the list of asset documents in the database. By synchronous long poll updates as described above, the imported 3D asset is immediately available for placement.

We included Redmine[8] exemplary for existing project management systems to the work space. Redmine is a Web-based project management system, implemented using the Ruby on Rails framework[9] and published under the GNU general public license. Project descriptions as well as issues and issue trackers for projects can be queried via a REST API. In the terms of the introduced 3D editor tool, we map Redmine projects to configuration projects and Redmine sub-projects to scenarios. For personal messages, we introduced a respective tracker in Redmine and model messages in the configuration tool by issues in Redmine. The link between Redmine projects and the 3D work space is realized by respective parameter attributes for objects in the scene database. Redmine provides moreover HTTP Basic Authentication [30] via REST and can therefore be used for user login. By taking this step, the set of projects provided to a user when entering the 3D work space is determined by rights granted by the Redmine configuration.

6.2. HAVLE verification service

The verification tool used in the implementation is HAVLE. The specification unit of HAVLE provides a graphical user interface for modeling hybrid automata in a graph-like manner. The graphical editor is a GMF[10]-based Eclipse plugin. Automata entered in the

[7] http://couchdb.apache.org

[8] http://www.redmine.org

[9] http://www.rubyonrails.org

[10] Graphical Modeling Framework, http://www.eclipse.org/modeling/gmp/

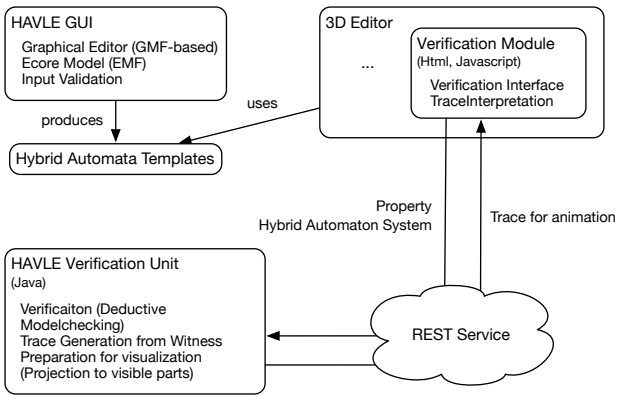

Figure 8. Components of the Verification Module

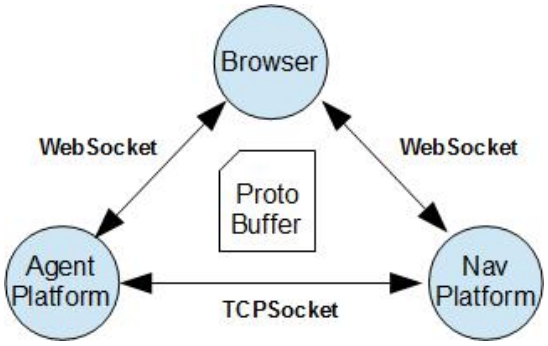

Figure 9. Agent platform and navigation service infrastructure

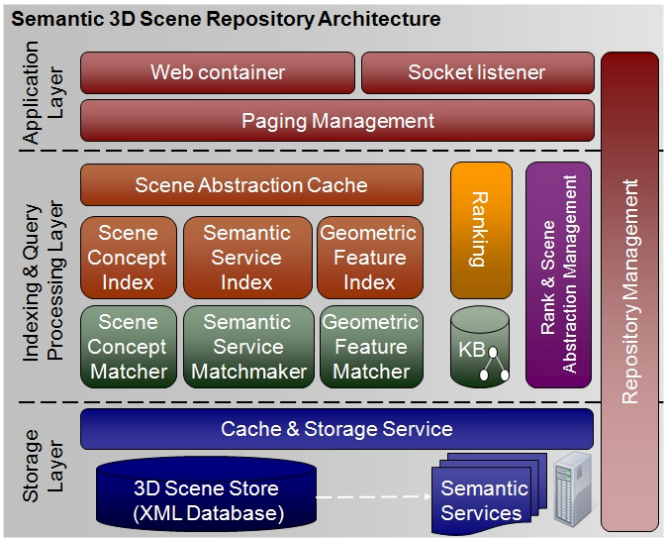

Figure 10. Architecture of the iRep3D 2.0 scene repository.

graphical editor are transformed into Ecore models using the Eclipse Modeling framework (EMF[11]).

The verification unit itself is written in plain Java. In the verification process the automata that the system consists of are composed and translated into a logic based representation. The algorithm that performs the actual verification is a combination of deductive techniques and model checking and produces additionally to the answer whether a property holds or not, a witness (if possible). From this witness it computes a formal description of behavior that leads to the desired or undesired state.

The verification module of the collaboration environment presented in this paper is implemented in Javascript with HTML as GUI front end (see Figure 8). It accesses HAVLE trough a REST service sending the formal model extracted from the system model designed in the 3D editor and the property to be evaluated to HAVLE and receiving the formal trace as a result. This trace is then processed by a trace interpreter written in Javascript: it goes through the formal trace step by step[12] and produces the according state in the 3D model using the basic (Xflow) animations and the mapping that describes how values of variables and locations in the formal model correlate to basic the animations and values of variables in the 3D Model.

6.3. Agent platform and navigation service

As already outlined in section 4, the service for agent simulation consists of an agent platform provided by Jadex and an associated navigation service for navmesh generation and agent positioning. The associated navigation service is *HumanSim*. Jadex and HumanSim communicate with each other by TCP Socket connections. HumanSim consists of two parts: a service platform that is implemented in Java, and a browser-side part written in JavaScript that interprets the messages sent from the platform (see Figure 9). The browser-side JavaScript implementation moreover provides an API to access agents managed by Jadex, create new agents in Jadex, or assign plans to existing agents directly from the browser application, in our case, the 3D editor.

When an agent-based simulation is started, the geometry of the scene that is stored in the XML3D data layer is automatically collected and sent to the HumanSim platform. Out of this geometry, a navmesh is created that is used to check for collisions between Jadex agents and obstacles in the 3D environment during agent simulation.

This setup allows us to run a BDI engine for agent simulation with taking the structure of the user generated 3D artefact into account.

[11] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/
[12] Every time a new frame is demanded by the renderer an new step is computed.

6.4. iRep3D

We have designed and implemented our first prototype of iRep3D repository in Java based on Springframework[13], Ehcache[14] and JPA 2.0[15] with Hibernate 3.0[16]. We briefly introduce, in this subsection, the implemented facilities for in particular the real-time semantic query processing. For this, the syntactic, conceptual, semantic service and geometric features of 3D scenes are extracted in advance during off-line indexing creation phase. Each ranked list of scenes is persisted as a disk file. The latter is further used for query processing by a lazy-loading mechanism, which loads the top T entries (T is configured with 50 in the prototype) into memory during system initialization and reads the next T entries when they are needed by rank merging process. Besides, the extracted features of 3D scenes are organized as scene abstracts that are further stored in a MySQL database for the purpose of showing 3D scene details (not scene selection) on demand. By means of pooling the connections to database, this performs quicker than the XML-based on-line query on 3D scene files.

Three layers of iRep3D repository are implemented in a loose-coupled fashion (see Figure 10). They can be deployed in separate servers. Each of them is able to communicate with the others via IP sockets. To increase their throughput, each layer maintains a first-in-first-out request cache. Pooled acceptor threads put their received requests from upper layer to the cache and the processing threads serve the requests out of cache. A query is processed by four joined threads in parallel. Each of them responses to the searching of scenes in syntactic, conceptual, semantic service or geometric aspect. In addition, we enabled a result caching heuristics for storing selected scene files and abstracts that would be used on demand. The list of selected 3D scene identifiers is also cached in query processing layer. When a page is displayed on demand, the contents for the next one are preloaded. For speeding up the simulation of reality, 3D scene files are (pre-) cached if they were (will be) requested (by the next page).

6.5. Server-Side Integration Approach

To evaluate the use of the server-based integration approach and measure the performance of the synchronisation approach, we model the presented use case of agent-enriched factory design and evaluation using the proposed server architecture. If we manage to

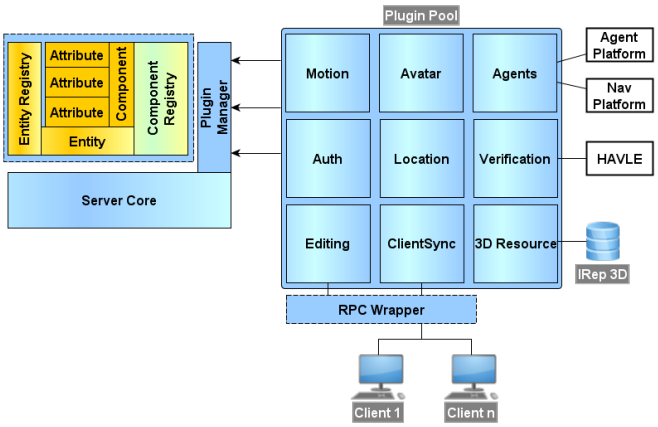

Figure 11. Factory evaluation workspace implementation based on synchronization server

map the concepts introduced by the respective services to components and network services provided by the according modules, we immediately gain a real-time synchronization of all connected clients in both design and simulation time of the application.

- *HAVLE*: The plugin that integrates HAVLE into the application introduces components to model automata configurations. Assets that constitute automata in the final design artifact are then Entities which carry information about their automata configuration in the Components. The service interface provides functions to invoke the verification procedure or start or stop the visualization. The Xflow animations from which verification visualization is composed are contained in the XML3D representation of the object, as before.

- *HumanSim*: Assets that are equipped with agent behavior in the XML3D editor are represented as entities that carry the specific agent configuration in their Components. The plug-in that integrates the agent platform communicates with the navigation component as it was done before by the client. Updates from the navigation server concerning agent positions are now applied to the entity's position and thus synchronized with connected clients. The server's service interface is extended by functions to start and stop agent simulation, create a Component with agent configuration for an entity or modify an existing agent configuration.

- *IRep3D*: IRep3D is run and accessed by its own web interface as before. 3D assets which are imported from IRep3D are now modeled as Entities which store a reference to the resource in IRep3D.

[13] http://www.springsource.org/spring-framework
[14] http://ehcache.org/
[15] http://jcp.org/en/jsr/detail?id=338
[16] http://www.hibernate.org/

	Editing Session	Visualization	3 Agents	10 agents	20 agents
Client only	30 fps	30 fps	27 fps	22 fps	14 fps
Server / 1 client					
Total message roundtrip	6 ms	6 ms	6 ms	6 ms	7 ms
Server-side incoming	< 1 ms	< 1 ms	1 ms	1 ms	1 ms
Server-side outgoing	3 ms	3 ms	4 ms	4 ms	5 ms
Server / 5 clients					
Total message roundtrip	8 ms	8 ms	9 ms	10 ms	10 ms
Server-side incoming	< 1 ms	< 1 ms	1 ms	1 ms	1 ms
Server-side outgoing	3 ms	5 ms	5 ms	8 ms	8 ms
Server / 20 clients					
Total message roundtrip	12 ms	13 ms	14 ms	15 ms	22 ms
Server-side incoming	< 1 ms	< 1 ms	1 ms	1 ms	4 ms
Server-side outgoing	5 ms	10 ms	12 ms	13 ms	15 ms

Table 1. Render frame rate and message processing times for increasing scene complexity

- *Client*: Implementation of the client becomes simpler when applying the integration server approach. The client just needs to connect to the server and retrieve the list of provided services. When Component updates are received that affect objects in the XML3D scene, the updates are applied to the respective XML3D elements' attributes. An update concerning a *position* Component of an Entity, for example, changes the *translation* attribute of the respective XML3D <*transform*> element. Key frame updates for verification visualization are applied to the respective Xflow <*data*> element.

7. Synchronisation: Empirical Performance Analysis

Our goal is to achieve a real-time interaction for both client-side and server-side integration approaches. For the client-side approach, network load is less of an issue, as all that needs to be transmitted here are atomic scene modifications. Those are directly synchronized via the chosen database. Experienced latencies occur solely from the quality of the network signal. Thus, for the client case, we are only measuring performance in terms of rendered frames per second during client-side simulations.

By introducing the server as integration and synchronization layer, we also introduce a possible bottle neck concerning network performance. The server has to translate updates from attached services to messages for its own service interface, apply service updates to the internal scene model and transmit these changes to all connected clients for shared real time visualization.

For the server, we have also included the following values for measurements:

- **Total message roundtrip**: The time in milliseconds a message needs to re-appear at the client that sent it. This measures the time any client will observe modifications done by others.
- **Server-side incoming processing**: The time it takes for an incoming message to be processed on server side, i.e. until a scene modification by a client is applied to the server's world representation.
- **Server-side outgoing processing**: The time it takes for an update of the server's internal data to be sent as serialized message to the clients.

The performance analysis process is the following: First, we navigate through the scene and add an object to it (*Editing Session*). For the client-side implementation, we measure the render frame rate. For the server-side implementation, we measure the time from adding the object in one client until the respective update is received by another client. We repeat this procedure while a visualization of the trace produced by verification service is executed. Then we add agent avatars to the scene and measure the performance with a number of 3, 10 and 20 avatars respectively. Note that both trace visualization and avatar animation perform frequent Xflow animation key updates, whereas agent updates also include updates for both position and orientation. Those create both a higher number of update messages sent by the server and a higher number of DOM manipulations to update the XML3D scene representation.

To achieve a precise measurement of the message roundtrip times, we modify the messages such that they contain the time stamp at the time when they were sent as payload. During the processing steps on the server, the timestamps at which the different processing steps (incoming processing and outgoing processing) are completed are added to the payload as well. When the message arrives back at the client, these stored

timestamps are compared to the current time in the client. Client and server are run on the same computer to rule out discrepancies by inexact time clocks on either server or client machine. In each test, we send an update message every 100ms for each client, which corresponds to 10 scene updates per client per second. The tests reported in Table 1 were run for 120 seconds each.

We observe that we achieve reasonable frame rates in the client for up to 10 agents. For more than 10 agents, frame rates drop below the desired rate of 25 fps. For the presented use case, however, we consider a number of 10 to 20 agents sufficient, and therefore the client performs well in the given setup.

For the server-implementation, the message transfer rates are entirely satisfying. Even with 20 clients observing the same scene, message updates occur faster than the actual frame rate, and thus frequent enough to not cause any observable delay in the processing. Our approach of the homogeneous service interface constitutes only minimal processing times. Most of the transfer delay is caused by actually sending the serialized message over the network.

8. Related Work

8.1. Collaborative Environments: Research and Prototypes

A project that thoroughly dealt with collaboration in virtual environments is the DiFac project as described by Sacco et al [32]. They developed a factory design tool in VR, which integrates additional services for communication and project coordination. In addition, visualization of evaluation processes on a factory setup helps to find drawbacks in a specific factory setup. Information from all external services is directly visualized in the virtual environment. In contrast to the work presented in this paper, DiFac is run in a standalone VR application and not integrated in any web browser. Moreover, the set of additional services is fixed, whereas we aimed at providing a flexible API to append services of users' choice.

Pappas et al presented *DiCoDev* [33], a web-based collaborative editing tool for 3D data. It includes user management with user roles, access management and provides file exchange for documents and 3D data. The workspace is embedded into a virtual reality application based on the commercial PTC Division MockUp platform[17]. Within this VR environment, multiple users can create 3D content in real-time in a collaborative session.

Based on DiCoDev, Smparounis et al. created a *Collaborative Prototype designer* that adds functions for collaborative review sessions to the original application [34]. In these sessions, users can inspect created models by navigating freely around them in 3D space, share their current view points with others or navigate to previously defined fixed view points. The review and evaluation process is supported by a decision support module.

Menck et al. introduced a system for collaborative factory planning in a virtual environment [35], implemented using VRUI, a framework to implement VR applications in C++. The system provides an immersive virtual factory environment for several users. Users can add objects to the virtual factory, change their positions and remove them again. Objects can be annotated to point out open issues or as documentation of the design process. Moreover, virtual meetings can be held in the virtual world, while the annotated 3D factory is explored.

While the latter examples make use of third party platforms to display 3D content, we provide a web-based 3D scenario editor that does not need any software or libraries installed but the web browser itself.

Wan et. al presented WebMWorks, a browser-based tool for collaborative design, modeling, simulation and knowledge sharing [36], building upon Modelica, an object-oriented language for modeling physical systems [18]. While models in this system are usually expressed by an equation based language, there also exists a schematic, diagram-like visualization of respective models. The WebMWorks system introduced by Wan et. al allows to create these models collaboratively in a Web-browser, and run simulations based on created systems. The actual models are served from a cloud storage. In contrast to the approach presented in this paper, WebMWorks does not provide any 3D visualization of created systems. Moreover, the system is built to work on Modelica, and does not provide any possibility to extend the existing platform by additional external services.

8.2. Synchronized Virtual Worlds

OpenSim is a widely known open source implementation of the *Second Life* server. Existing Second Life clients can be used to connect to an OpenSim server. OpenSim is designed to be modular and offers several extension mechanisms. However, even though only a subset of features of Second Life may be used, both new client applications and server extensions have to implement the Second Life synchronization protocol.

[17] http://www.ptc.com/product/division

[18] https://www.modelica.org/

RealXtend Playsing association introduced the *realXtend* project [19] that also uses the Entity- Component-Attribute (ECA) model to represent objects in the virtual environment [31]. Similar to our approach, entities in the world carry no information from the beginning and domain-specific information is added via attributes. RealXtend provides 3D assets in Ogre3D format and uses its own protocol *kNet* for synchronization, which puts a higher effort into development of purely browser-based clients compared to our approach. Moreover, extension of the realXtend system is based on scripted world objects and does not provide means to extend the server-side code.

Apart from the fact that data provided to the client both for 3D rendering and during synchronization is not optimized towards a purely browser-based client, the presented approaches do not consider the case of composing applications from existing external Web services.

8.3. Commercial Applications

The topic of collaborative work on 3D visualizations is not only a subject in research. Many commercial CAD (*Computer Aided Design*) and 3D modelling tools nowadays include collaborative features: Autodesk's *AutoCAD 360*[20] provides a collaborative 2D-workspace for CAD design. Work can be saved to a cloud storage and shared with co-workers. There exist versions for desktop PCs, mobile devices and a prototypical Web-browser-based frontend to allow ubiquitous access of the data. By decoupling access of the data from a specific machine and providing a browser-based implementation, designers are no longer bound to a licensed machine. However, AutoCAD 360 does not provide 3D visualization of data, but is limited to 2D sketches.

A popular tool for 3D CAD design is *CATIA* by Dassault Systèmes[21]. With *Instant Collaborative Design 1 (CD1)*[22], *CATIA* is extended by capabilities of sharing a workspace interactively with co-workers. Data transfer is realized either via peer-to-peer or client-server connection. Communication features are included by a real-time chat and personal messages for asynchronous collaboration. Whereas CD1 offers sophisticated solutions for concurrent product design, its use is limited by the fact that in order to participate in the work, software for each participant has to be licensed and installed on a specific machine. This obviously slows down the process of introducing a new member to the team or to allow a remote expert to temporarily participate in design decisions.

9. Conclusion and Future Work

In this paper, we presented a web-based collaborative development environment for virtual prototyping. Whereas commercial systems often are monolithic, we developed a Web-based, distributed, service-oriented and extendable collaboration framework which provides a shared visual experience for collaborators. We presented a design for a server-side integration approach that allows to extend the framework easily by plugins and adapt it to specific use-case scenarios. As design problems are often ill-structured, the open character of the service-oriented approach allows domain experts from different areas to discuss design issues without precluding possible problem solving steps. We described the services included in the framework that can be used for evaluation of the designed artefact. We have presented implementations for both a purely client-based and the integration server-based approach and applied both to a factory design scenario. The performance analysis of the synchronization approach shows that for the given class of use cases a sufficient update rate can be achieved.

As future work, we will extend the framework to enable mobile access to the collaborative workspaces, using tablets and smartphones as devices. Furthermore, we plan to include virtual reality devices, e.g. head-mounted displays in order to provide a more immersive experience of the 3D scenes.

For the integration server, we are planning to include a service bus component that allows configurable orchestration of attached plugins. This shall reduce concurrency conflicts that can occur when several plugins access the data maintained by the server core at the same time.

We are also investigating of how to improve the service interface of the server in such a way that not only Web-clients but also other server instances can connect to each other to form a cluster. Balancing the computational workload between several server nodes will increase the scalability of the system, concerning both the complexity of computations and the number of connected clients.

9.1. Copyright

<To be decided>

9.2. Rules of Use

<To be decided>

[19] http://www.realxtend.org
[20] https://www.autocad360.com/
[21] http://www.3ds.com/products/catia/
[22] http://www.3ds.com/products/catia/portfolio/catia-v5/all-products/domain/Infrastructure/product/CD1/

References

[1] I. Zinnikus, X. Cao, M. Klusch, Christopher Krauss, Andreas Nonnengart, Torsten Spieldenner, and Philipp Slusallek, "A collaborative virtual workspace for factory configuration and evaluation". In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 9th International Conference*, pages 353–362, 2013.

[2] L. M. Camarinha-Matos, "Collaborative networked organizations: Status and trends in manufacturing," *Annual Reviews in Control*, vol. 33, no. 2, pp. 199 – 208, 2009.

[3] G. Booch and A. W. Brown, "Collaborative development environments," *Advances in Computers*, vol. 59, pp. 1–27, 2003.

[4] K. Dullemond, B. van Gameren, and R. van Solingen, "Collaboration should become a first-class citizen in support environments for software engineers." in *CollaborateCom*. IEEE, 2012, pp. 398–405.

[5] T. DeMarco and L. Timothy, *Peopleware - productive projects and teams*, 1987.

[6] D. Perry, N. Staudenmayer, and L. Votta, "People, organizations, and process improvement," *Software, IEEE*, vol. 11, no. 4, pp. 36–45, Jul. 1994.

[7] P. W. Mattessich, B. R. Monsey, and M. Amherst H. Wilder Foundation, St. Paul, *Collaboration [microform] : What Makes It Work. A Review of Research Literature on Factors Influencing Successful Collaboration / Paul W. Mattessich and Barbara R. Monsey*. Distributed by ERIC Clearinghouse [Washington, D.C.], 1992.

[8] J. M. Czajkowski, "Leading successful interinstitutional collaborations using the collaboration success measurement model," 2007. [Online]. Available: http://www.mc.maricopa.edu/community/chair/conference/2007/papers/leading_successful_interinstitutional_collaborations.pdf

[9] J. Roschelle and S. D. Teasley, "The construction of shared knowledge in collaborative problem solving," in *Computer-Supported Collaborative Learning*, C. O'Malley, Ed. Berlin: Springer, 1995, pp. 69–97.

[10] C. A. Ellis, S. J. Gibbs, and G. Rein, "Groupware: some issues and experiences," *Commun. ACM*, vol. 34, no. 1, pp. 39–58, Jan. 1991.

[11] H. Fuks, A. B. Raposo, M. A. Gerosa, and C. J. P. Lucena, "Applying the 3C model to groupware development," *International Journal of Cooperative Information Systems*, vol. 14, no. 2 - 3, pp. 299–328, 2005.

[12] A. de Moor, "Towards more effective collaborative workspaces: From collaboration technologies to patterns," in *4th Collaboration at Work Experts Group Meeting*, 2006.

[13] D. Jonassen, *Learning to Solve Problems: A Handbook for Designing Problem-Solving Learning Environments*. Taylor & Francis, 2010.

[14] A. Schmeil and M. J. Eppler, "Knowledge sharing and collaborative learning in second life: A classification of virtual 3d group interaction scripts."

[15] K. Sons, F. Klein, D. Rubinstein, S. Byelozyorov, and P. Slusallek, "XML3D: interactive 3D graphics for the web," in *Web3D '10: Proceedings of the 15th International Conference on Web 3D Technology*. New York, NY, USA: ACM, 2010, pp. 175–184.

[16] W3C, "Document Object Model definition," http://www.w3.org/DOM/, 2005.

[17] F. Klein, K. Sons, D. Rubinstein, S. Byelozyorov, S. John, and P. Slusallek, "Xflow - declarative data processing for the web," in *Proceedings of the 17th International Conference on Web 3D Technology*, Los Angeles, California, 2012.

[18] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[19] X. Cao and M. Klusch, "irep3d: Efficient semantic 3d scene retrieval," in *VISAPP (2)*, S. Battiato and J. Braz, Eds. SciTePress, 2013, pp. 19–28.

[20] P. Kapahnke, P. Liedtke, S. Nesbigall, S. Warwas, and M. Klusch, "Isreal: an open platform for semantic-based 3d simulations in the 3d internet," in *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part II*, ser. ISWC'10, 2010, pp. 161–176.

[21] M. Klusch and P. Kapahnke, "The isem matchmaker: A flexible approach for adaptive hybrid semantic service selection," *Web Semant.*, vol. 15, pp. 1–14, Sep. 2012.

[22] A. Nonnengart, "A deductive model checking approach for hybrid systems," Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, Research Report MPI-I-1999-2-006, November 1999.

[23] C. Krauß and A. Nonnengart, "Formal analysis meets 3d-visualization," in *Concurrent Engineering Approaches for Sustainable Product Development in a Multi-Disciplinary Environment*, J. Stjepandic, G. Rock, and C. Bil, Eds. Springer London, 2013, pp. 145–156.

[24] T. A. Henzinger, "The theory of hybrid automata," in *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, ser. LICS '96. Washington, DC, USA: IEEE Computer Society, pp. 278–292.

[25] M. Bratman, *Intention, Plans, and Practical Reason*, ser. Center for the Study of Language and Information - Lecture Notes Series.

[26] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, J. Allen, R. Fikes, and E. Sandewall, Eds. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, pp. 473–484.

[27] Backbone.js, "Backbone.js," Project Web Page, 2013, http://www.backbonejs.org.

[28] A. Goldberg, "Information models, views, and controllers," *Dr. Dobb's J.*, vol. 15, no. 7, pp. 54–61, May 1990.

[29] J. C. Anderson, J. Lehnardt, and N. Slater, *CouchDB: The Definitive Guide Time to Relax*, 1st ed. O'Reilly Media, Inc., 2010.

[30] J. Franks, P. Hallam-Baker, J. Hosteler, S. Lawrence, P. Leach, A. Luotonen, and L. Stuart, "Http authentication: Basic and digest access authentication," June 1999. [Online]. Available: http://tools.ietf.org/html/rfc2617

[31] T. Alatalo, "A Virtual World Web Client Utilizing An Entity-Component Model," in *IEEE Internet Computing*, vol. 15, no. 5, pp. 30-37, September/October, 2011.

[32] M. Sacco, C. Redaelli, C. Constantinescu, G. Lawson, M. D'Cruz, and M. Pappas, "Difac: digital factory for human oriented production system," in *Proceedings*

of the 12th international conference on Human-computer interaction: applications and services, ser. HCI'07, 2007, pp. 1140–1149.

[33] M. Pappas, V. Karabatsou, D. Mavrikios, and G. Chryssolouris, "Development of a web-based collaboration platform for manufacturing product and process design evaluation using virtual reality techniques," in *International Journal of Computer Integrated Manufacturing*, vol. 19(8), 2006, pp. 805–814.

[34] K. Smparounis, K. Alexopoulos, and V. Xanthakis, "A Web-based Platform for Collaborative Product Design and Evaluation," in *15th International Conference on Concurrent Enterprising (ICE)*, 2009.

[35] "Collaborative Factory Planning in Virtual Reality," *Procedia {CIRP}*, vol. 3, pp. 317 – 322, 2012, 45th {CIRP} Conference on Manufacturing Systems 2012.

[36] L. Wan, C. Wang, T. Xiong, and Q. Liu, "A Modelica-Based Modeling, Simulation and Knowledge Sharing Web Platform," in *20th ISPE International Conference on Concurrent Engineering*, September 2013.