Hindawi Publishing Corporation
International Journal of Aerospace Engineering
Volume 2016, Article ID 6806198, 15 pages
http://dx.doi.org/10.1155/2016/6806198

Research Article

A Method for SWIM-Compliant Human-in-the-Loop Simulation of Airport Air Traffic Management

Thomas Gräupl,[1] Martin Mayr,[2] and Carl-Herbert Rokitansky[2]

[1]*Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Institut für Kommunikation und Navigation, Münchner Straße 20, Oberpfaffenhofen, 82234 Wessling, Germany*
[2]*Universität Salzburg, FB Computerwissenschaften, Jakob-Haringer-Straße 2, 5020 Salzburg, Austria*

Correspondence should be addressed to Thomas Gräupl; thomas.graeupl@dlr.de

Received 17 March 2016; Accepted 7 June 2016

Academic Editor: Rafael Apaza

System Wide Information Management (SWIM), as envisioned by the *Single European Sky Air Traffic Management Research* (SESAR) program, is the application of service oriented architectures to the air traffic management domain. Service oriented architectures are widely deployed in business and finance but usually tied to one specific technological implementation. SWIM goes one step further by defining only the semantic layer of the application integration and leaving the implementation of the communication layer open to the implementer. The shift from legacy communication patterns to SWIM is fundamental for the expected evolution of air traffic management in the next decades. However, the air traffic management simulators currently in use do not reflect this yet. SWIM compliance is defined by semantic compatibility to the *Air Traffic Management Information Reference Model* (AIRM) and a SWIM service may implement one or more communication profiles, which specify a communication layer implementation. This work proposes a SWIM-compliant communication profile suitable to integrate SWIM-compliant tools into human-in-the-loop simulations for air traffic management research. We achieve this objective by implementing a SWIM communication profile using XML-based multicast messaging and extending the message format to support distributed human-in-the-loop simulations. We demonstrate our method by the evaluation of Hamburg Airport operations.

1. Introduction

The way air traffic management information is exchanged between stakeholders is changing. Legacy point-to-point connections and proprietary data formats as shown in Figure 1(a) shall be replaced by *System Wide Information Management* (SWIM) as illustrated in Figure 1(b). SWIM is the technical term for service oriented architectures in the air traffic management domain. Service oriented architectures are an approach to the integration of heterogeneous systems that is widely deployed in the industry [1]. SWIM was envisioned by the *Single European Sky Air Traffic Management Research* (SESAR) program. This program defines a framework for the sustainable modernization of the European air transportation system. It aims to increase capacity, improve safety, and reduce environmental footprint and costs. The improvement of air traffic management procedures by embracing SWIM [2] and automation [3] shall be one of the key enablers to achieve these objectives.

Changing the way air traffic management information is shared will also fundamentally change how air traffic management is performed. However, changes of air traffic management procedures and tools need to be evaluated carefully, often by simulation, before they are applied. Zellweger [4] identifies several stages from the early exploration of new concepts to the final evaluation of air traffic management procedures and support tools. All but the final stage involve the use of simulation for safety and cost reasons. The effects of potential changes are first assessed with fast-time simulations. Secondly, the details of the concept are explored with real-time simulations and evaluated in human-in-the-loop simulations. Only in the final stage real-world operations are considered in the form of shadow mode operations (in shadow mode operations, the new procedure is conducted

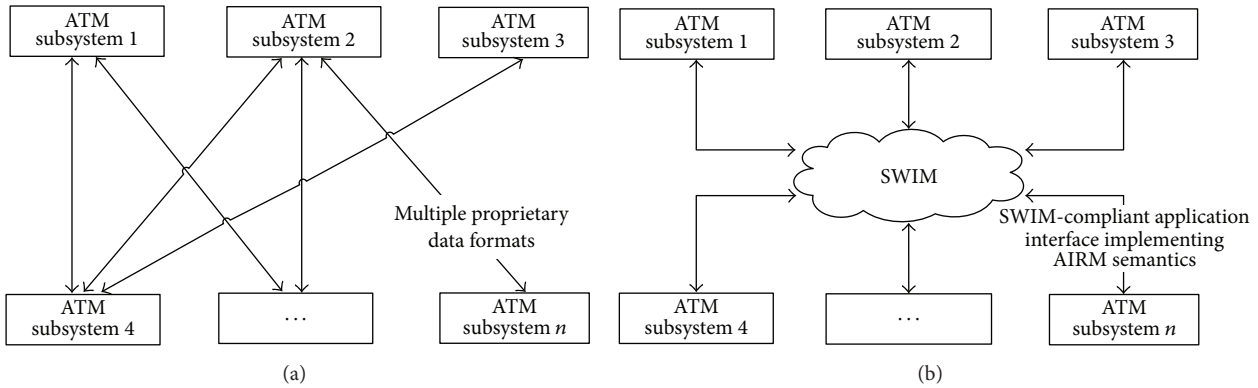


Figure 1: (a) The current *Air Traffic Management* (ATM) systems use legacy point-to-point connections with proprietary data formats. (b) This shall be replaced by the SWIM service oriented architecture for air traffic management.

in parallel to the established procedure). Human-in-the-loop simulations therefore provide an important contribution to the identification of the strengths and weaknesses of new air traffic management procedures and tools.

In this vein, Kaltenhäuser [5] identified the need to support standard air traffic control interfaces as an important quality of a flexible air traffic management simulation environment. Flexible, easy to adapt interface software was thus identified as a primary task for further research. Adelantado [6] addressed the use of standard interfaces by the use of the *High Level Architecture* (HLA) to implement a distributed airport simulation environment for the French Aerospace Research Center ONERA. The simulation was implemented with HLA federates, each simulating different aspects of the airport environment, connected to a central HLA runtime infrastructure. However, HLA was not universally accepted. Prevot et al. [7] indicate that the NASA Ames Research Airspace Operation Laboratory was implemented on the basis of the "Aeronautical Data Link and Radar Simulator" communication network and Edinger and Schmitt [8] report that the German Aerospace Center DLR implemented its air traffic management simulation environment on the basis of a client-server approach employing a "Data Pool" server process as communication hub. Shifeng and Danxia [9] developed a cost-effective approach to constructing a flight control tower simulation using commercial-of-the-shelf hardware. They also introduced the use of multicast to distribute simulation state encoded in "primitive commands" proprietary to their implementation.

A lot of work has thus gone into the creation of flexible and easy to adapt simulation interface software, but the support of standard air traffic control interfaces has not yet been fully realized. One major reason for this is certainly the multitude of legacy point-to-point connections and data formats used by air traffic management systems. However, we argue that with the specification of SWIM an opportunity to support a universal standard air traffic control interface becomes available. In addition, new air traffic management procedures and software will have to be SWIM-compliant and should therefore be evaluated in a SWIM-compliant environment. In our view, air traffic management simulation environments shall therefore support SWIM's *Air Traffic Management Information Reference Model* (AIRM) and SWIM's message passing based communication pattern to evaluate proposed air traffic management procedures and tools in a realistic simulation environment. SWIM compliance is achieved by supporting the AIRM semantic [10].

The objective of this paper is to present a SWIM-compliant method for human-in-the-loop air traffic management simulation that (1) uses the air traffic management information reference model as semantic interface, (2) uses multicast message passing as communication interface, and (3) therefore enables the evaluation of SWIM-compliant air traffic management software in complex simulation scenarios. The SWIM AIRM is already intended for real-time information sharing and can therefore be extended to support human-in-the-loop simulation. Using message passing for communication supports the integration of existing SWIM-compliant air traffic management tools. In addition, the use of multicast is attractive because it is efficient and offers the additional opportunity to replace the client-server architectures of the current simulation environments with a decentralized approach not requiring centralized simulation infrastructure.

The application of our method is demonstrated with human-in-the-loop simulation experiments evaluating airport operations at the Hamburg Airport. The experiments discussed in this paper were originally performed in the context of the SESAR Work Package E Project "Zero Failure Management at Maximum Productivity in Safety Critical Control Rooms" (ZeFMaP) by Zeh et al. [11] but were extended to produce the results presented in this paper. The objective of the extended simulation experiments was to capture the decision process of the human air traffic controllers in a rule-based model that shall provide the basis for the development of automated support tools for airport operations.

2. Definitions

In this paper we follow Stal [12] in the view that service oriented architectures are loosely coupled "software islands" with a common semantic layer interconnected through

a communication layer. The communication interface may have several different implementations.

Glickman [13] defines SWIM as the application of the service oriented architecture approach to the air traffic management domain. Instead of developing and implementing specific solutions for sharing data between application pairs, SWIM specifies a semantic model, common infrastructure, and set of processes for sharing and managing data in the air traffic system.

According to Wilson et al. [10] particular implementation is *SWIM-compliant*, if its entities and their properties can be mapped to the semantic AIRM information reference model defined by SESAR. A SWIM service need not implement the complete semantic model. It is sufficient to implement the relevant parts of the semantic model. SWIM compliance does also not depend on a specific implementation of the communication layer. Any communication layer available to the SWIM services is acceptable.

3. Background on SWIM

The air traffic management system is considered one of the most complex systems ever built, but many parts of it were not designed to be integrated with each other. This leads to a multitude of point-to-point connections between its subsystems using proprietary data formats.

SESAR aims to bridge this interoperability gap by the introduction of service oriented architectures into the air traffic domain. The SWIM service oriented architecture shall provide the infrastructure, data formats, and protocols to share information between all air traffic management subsystems in a scalable and interoperable way. SESAR defines the semantic layer of SWIM in the AIRM information reference model as described by Wilson et al. [10]. In the communications layer it specifies the integration of air traffic management applications by message passing systems. Applications are outfitted with an interface to the message passing system and the message exchange within the system is handled by message brokers, which can scale from local software libraries to federated networks of broker servers shown in Figure 2(a). The actual implementation of the messaging protocol and message brokers may vary and is open to the implementer.

As of now, three SWIM implementation profiles have been defined by SESAR. The "yellow profile" utilizes the *Hypertext Transfer Protocol* (HTTP) as communication protocol with XML message formats defined in the *Web Services Description Language* (WSDL). The "blue profile" utilizes the *Data Distribution System* (DDS) protocol with a binary message format. The "purple profile" utilizes the *Advanced Message Queuing Protocol* (AMQP) as communication protocol with message formats defined in the web services description language.

The SWIM implementation profiles developed in SESAR are intended for pan-European operational use. They are complex to implement and not intended for simulation environments. It is therefore not always possible to use them within the time and budget constraints of research projects.

Our method, called X23 (in this paper we use the name "X23" interchangeably for the presented approach, communication profile, and its software implementations), is simplified implementation of SWIM suitable for research and laboratory trials. We designed a scaled-down implementation with analogous information sharing capabilities but extended for event-driven simulations. Our implementation does not provide the same WAN scalability, security, and safety mechanisms as the pan-European SWIM network but provides semantic compatibility according to [10]. It can thus be understood as a LAN-scale research-oriented SWIM profile with extensions for human-in-the-loop simulation.

X23 integrates SWIM-compliant simulation and air traffic management tools through a multicast messaging protocol as illustrated in Figure 2(b). Assuming wall-clock time aligned simulation-time, as is usually the case with human-in-the-loop simulations, our approach does not require the configuration of a central instance for coordination. Shared simulation state is injected into the multicast group for distribution; each simulation tool can then extract the required information from the common information pool. Using IP multicast for this purpose has the advantage that it is natively supported without configuration in local area networks as they are commonly deployed. No reconfiguration is required when moving to a different network. Within the IP multicast group a domain specific XML message format is used to represent the simulation state.

4. Method for SWIM-Compliant Human-in-the-Loop Simulation

Our X23 SWIM implementation utilizes a simple, layered, and distributed architecture based on the OSI reference model. It has been implemented as a software library in the Java and Delphi programing languages. The Java library has been used from within MATLAB by Hauf et al. [14]. A C++ implementation is under development by the authors.

The implementation covers the protocol stack from the application layer down to the network layer as illustrated in Figure 3. The application layer implements the interface between the simulation application and the X23 stack. This interface may be realized in different ways depending on the type of the application. The presentation layer covers the encoding of state changes in a machine-readable, platform-independent XML representation. The session layer of the stack establishes logical sessions that are robust against communication interruptions or restarted simulation processes. The transport layer encapsulates the XML information of the upper layers for end-to-end transmission between the connected processes. Finally, the network layer and the layers below take care of the actual multicast transmission over the network.

The application layer interface is specific to the application but provides access to the same distributed AIRM-compliant semantics and information for all applications. The interface uses a local object cache storing and merging the state of the distributed objects. This cache is constantly updated with the messages coming from the lower layers

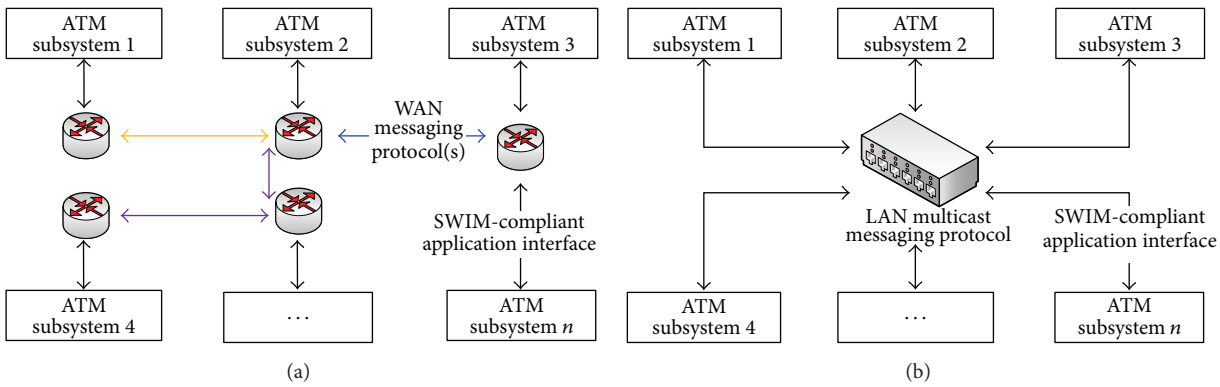


Figure 2: (a) The SESAR SWIM infrastructure is implemented by a *Wide Area Network* (WAN) of message brokers implementing several communication profiles. (b) Our method replaces the SWIM infrastructure with a *local area* multicast *network* (LAN) and the X23 communication profile.

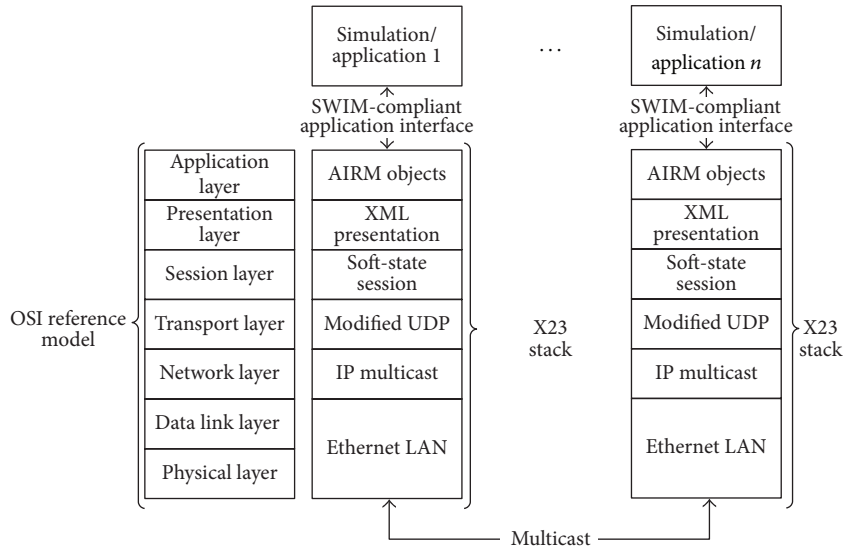


Figure 3: The X23 implementation covers the protocol stack from the application layer down to the network layer. The application layer implements the interface between the application and the X23 stack. The presentation layer covers the encoding of state changes in XML messages. The session layer establishes logical sessions robust against communication interruptions. The transport layer encapsulates messages in *User Datagram Protocol* (UDP) datagrams. The network layer and Ethernet take care of the actual multicast transmission.

of the X23 stack. The actual information is condensed into objects implementing AIRM entities, for example, flight objects, airport objects, or runway objects.

These objects are presented to the applications through a simple application interface or plugin realizing the message patterns specified for SWIM. Note that although SESAR specifies several message patterns, only two are actually used in the currently defined profiles: subscribe/notify and request/response. The application interface has thus only to support these two modes of operation. In many cases only one message pattern is required by a particular application.

Received messages are translated into AIRM-compliant objects and stored in the local object cache. Messages concerning the same object cause the object in the cache to be updated. This applies also to messages providing partial updates. If messages from different sources are received, this results in a transparent fusion of the information. Different sources may update the same attribute. However, in this work information fusion has only been used for updates of different attributes of the same object. This allowed X23 SWIM services and simulation modules to augment objects generated by different modules with additional information.

4.1. Message Format. Within our X23 stack, updates to the AIRM objects are implemented by simulation events distributed over the network as XML elements of the form:

```
<event_or_command attribute1="..."
attribtute2="..." ... />.
```

Each XML element corresponds naturally to one simulation event or simulation command. The properties of the

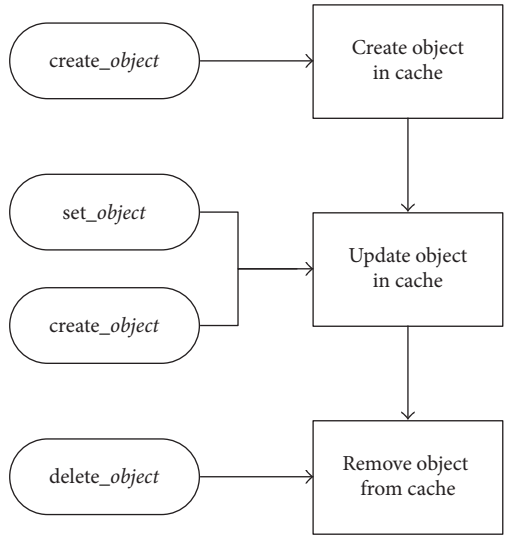


Figure 4: The life cycle of X23 objects in the object cache. Object creation is triggered by create_*object* messages, attributes are updated according to set_*object* messages, and objects deletion is finally caused by delete_*object* messages. As optimization cached objects may also be updated by create_*object* messages.

event or command are reported in the attributes of the XML element.

Simulation events report property changes of AIRM entities. The XML representation of the simulation events is analogous to the life cycle of the simulated objects as shown in Figure 4. Objects are created, used, and finally deallocated. A simulation module processing only selected simulation objects subscribes to these objects by accepting only messages reporting changes of these objects. This is best illustrated with an example: each simulated flight is represented in the simulation by a flight object the attributes of which can be traced back to the properties of the AIRM flight entity. A simulation module processing flight objects subscribes to the `<create_flight ... />`, `<set_flight ... />`, and `<delete_flight ... />` messages. The creation of a new simulated flight is presented by a `<create_flight ... />` message. This message has various attributes like the position of the flight, its departure airport, its arrival airport, and several others defined in the air traffic management information reference model. Analogous messages are defined for other AIRM entities, for example, airport entities or runway entities.

Simulation events conveying *Controller-Pilot Data Link Communication* (CPDLC is used to exchange clearances for the progress of the flight between the pilot and the air traffic controller via air-ground data link) follow the request/response message pattern and are represented by `<cpdlc_message ... />` messages based on the FANS-1/A message definitions. FANS-1/A is an international standard (specified in ARINC 622 and EUROCAE ED-100/RTCADO-258) for CPDLC that is not part of SWIM. Each message is specified by a unique identifier and its parameters.

Simulation commands are messages used for the coordination of the simulation environment. They are also not part of SWIM, but an extension introduced in our implementation. As our simulation environment is only intended for wall-clock time aligned human-in-the-loop simulations the simulation commands could be restricted to a small set of messages. Five commands are supported: *start*, *restart*, *pause*, *resume*, and *stop*. *Start* and *stop* are only used to initialize or shut down the simulation environment remotely. *Restart* causes all simulation processes to reinitialize and clear their object cache. *Pause* and *resume* allow pausing human-in-the-loop simulations without losing the current simulation state.

Examples of X23 messages can be found in the Appendix of this paper.

4.2. SWIM-Compliance. Our method implements the relevant entities of AIRM in X23 classes. For illustrational purposes the semantic trace of two fields of the X23 Runway class to the corresponding attributes of the AIRM runway entity is displayed in Figure 5.

A semantic trace of the X23 classes to entities of AIRM constitutes a formal proof of "level 1" SWIM-compliance according to Wilson et al. [10]. A semantic trace of the fields of the X23 classes to the attributes of AIRM entities is a proof of "level 2" SWIM compliance.

Figure 5 provides thus evidence for the "level 2" SWIM compliance of two fields of the X23 Runway class. Note that the object under assessment need not implement all entities and attributes specified in the reference model and may have additional attributes, for example, for simulation purposes that may be ignored.

4.3. Interfacing with SWIM-Compliant Applications. The interface of an application with particular SWIM implementation is application-dependent. It follows, however, the logical approach that is central to the SWIM concept: application

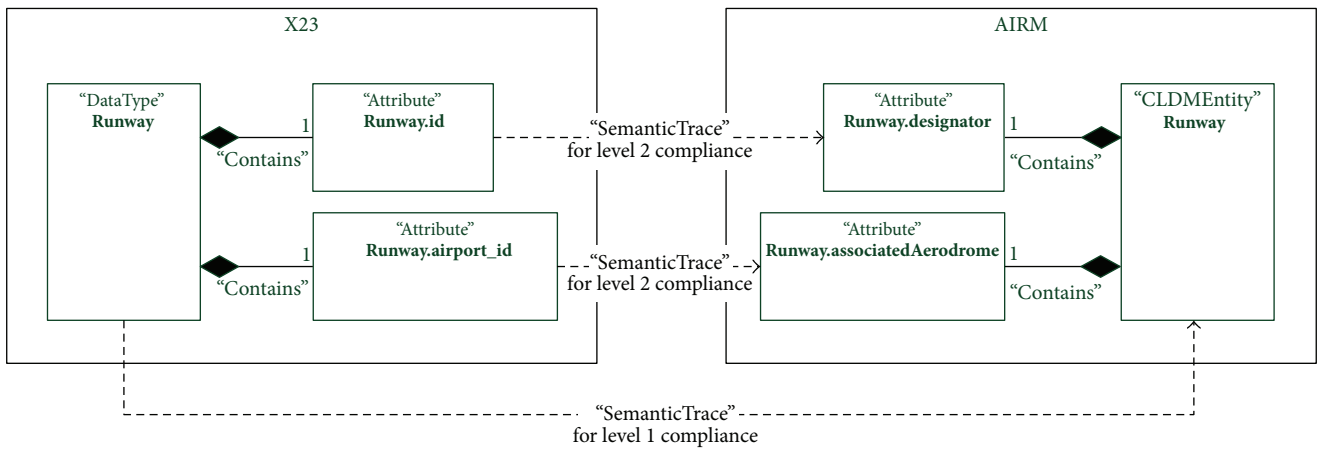


Figure 5: Semantic trace of the *id* and *airport_id* fields of the X23 Runway class to the corresponding attributes of the AIRM runway entity (top two arrows in diagram). This mapping provides evidence for the "level 2" SWIM-compliance the X23 Runway class. The semantic trace of the X23 Runway class to the AIRM runway entity would only have provided evidence for "level 1" compliance (bottom arrow in diagram).

layer events are mapped to messages of the communication layer according to a common semantic model. The properties of the application layer event, AIRM entity state changes, are mapped to the message format of the communication profile. This is always possible as SWIM-compliant applications and communication profiles must follow the AIRM semantics. It is not necessary to change the application logic for this purpose. However, it is required to add mapping logic.

In the case of X23 this means that it is required to implement a bridge between the *Application Programming Interface* (API) of the SWIM application and the X23 stack mapping AIRM entities and attributes to X23 objects and fields. Requests and responses are directly mapped to messages. Entity state changes are applied to the local object cache triggering the distribution of the event by X23.

4.4. Multicast Communication Protocol. The X23 network layer uses IP multicast over Ethernet. The rationale for using multicast is threefold: first, multicast is the most efficient way to exchange information between multiple nodes without a central node in a local area network. Secondly, using multicast enables self-configuring deployment. All simulation modules join the same local IP multicast group and listen on the same port for messages. This is the only required configuration and can be performed automatically and independently from the actual configuration of the simulation network. If stateless autoconfigured IP addresses are provided by the operating system, a deployment network is not required and the simulation hosts can be connected directly. This is of advantage if simulation experiments need to be performed with partner institutions that cannot allow external parties access to their internal networks. Thirdly, using multicast offers additional flexibility in setting up the simulation environment as simulation nodes may join and leave the network during runtime without the need to reconfigure other nodes.

At the transport layer UDP is used. At reception the XML messages are extracted from the received multicast UDP datagram and checked against a regular expression for syntactic integrity. If the message is invalid, the message is discarded with logging. Each XML message is tagged with a unique sequence number and source identifier. XML messages may be reordered or duplicated if applications are deployed on multihomed hosts (in practice this typically happens when simulation modules are deployed in virtual machines, which create additional (virtual) network interfaces on the host). Thus, the X23 transport layer protocol on top of UDP provides the deduplication of XML elements and filtering according to the application's subscription.

The X23 transport protocol header is transparently added as additional XML attributes. It comprises four fields: `<event_or_command source="..." sequence_number = "..." sender_uid="..."/>`. The `event_or_command` field specifies the type of the object conveyed in the message. It may indicate either a state change of a simulation object or a simulation command. The `source` field indicates the logical source (e.g., HMI and air traffic simulation) of the message to allow filtering without knowing the identity of the sending process. The `sequence_number` and `sender_uid` fields uniquely identify the message in the network. The `sequence_number` is incremented for each message that is not a retransmission. The `sender_uid` is a hash value uniquely identifying the process generating the message as several processes may implement the same logical source, for example, multiple HMI instances.

Messages following the request/response pattern are sent using acknowledgements, because single messages can get lost or be corrupted from time to time. A request/response is transmitted by the X23 transport layer until it is acknowledged by the desired receiver identified in the source field, or the maximum number of retransmissions is reached. A message is considered acknowledged if at least one acknowledgement message is received. Note that this protocol does not support acknowledgements from multiple receivers for now. The acknowledgement message is implemented as an X23 message, because UDP does not support transport layer

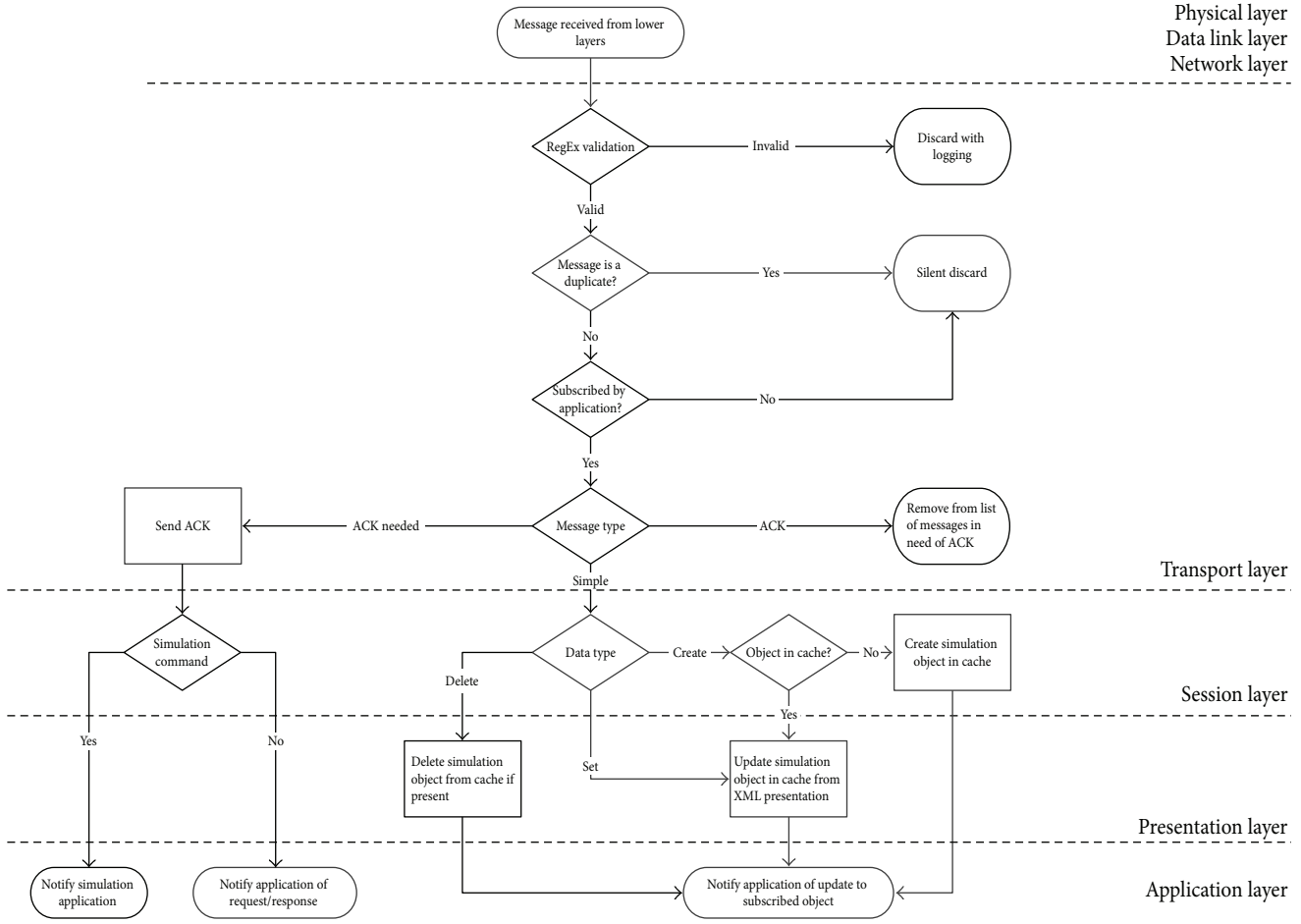


Figure 6: Processing of received messages in the X23 stack.

acknowledgements. The acknowledgment message is a XML message with the `sequence_number` and `sender_uid` of the original message and the source of the receiver. This allows the sender of the request/response to identify the acknowledgement unambiguously and other receivers to discard it silently. The acknowledgment message is not forwarded to the upper layers of the stack and not identical to the response to a request that is often called "logical acknowledgement," which is a higher layer message.

Messages available through the subscribe/notify message pattern are sent unacknowledged and are never retransmitted as the session layer redistributes them in regular intervals as described in the next paragraph.

The session layer uses a soft-state pattern allowing restarted simulation applications to recover lost simulation state. This is realized by redistributing the essential parts of the simulation state needed for recovery in regular intervals. In the case of flight objects this is implemented by the retransmission of the complete state vector of all flight objects currently in the simulation through newly generated `<create_flight ... />`messages every thirty seconds. Retransmissions for individual flights are randomly offset to avoid message bursts. Airport and runway information is retransmitted in an analogous way. This applies to all information available through the subscribe/notify message pattern. Request/response messages may be retransmitted several times to recover from lost packets but have to be reinitialized if the session breaks.

The presentation layer takes care of the creation and updating of X23 objects according to the received messages. The objects are stored in a local object cache accessible by the application layer.

The application layer notifies SWIM services and simulation modules of received requests, responses, or state changes according to the application-specific SWIM-compliant interface.

The processing of received X23 messages is illustrated in Figure 6. The processing of transmitted X23 messages is performed analogously and in reverse order.

5. Application to Human-in-the-Loop Simulation of EDDH Airport Air Traffic Management

The simulation experiments illustrating the application of our method were originally performed within SESAR Work

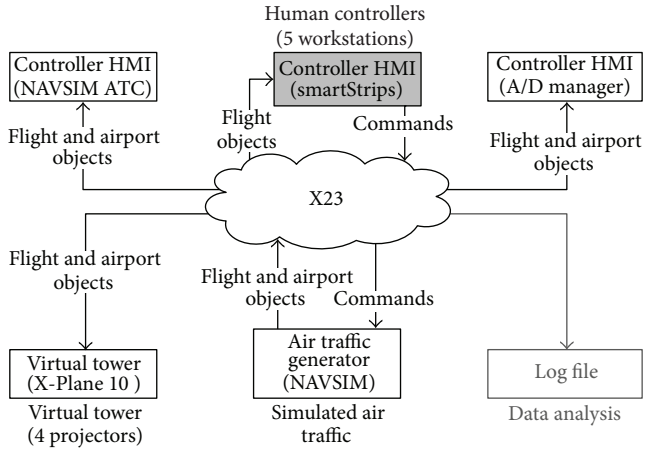


Figure 7: Logical information flow in the simulation's X23 SWIM network for the EDDH scenario. Flight and airport objects are distributed in the subscribe/notify message pattern. Commands follow the request/response message pattern. The shaded HMI was later replaced with a simulated controller.

Package E. The project was called "Zero Failure Management at Maximum Productivity in Safety Critical Control Rooms" (ZeFMaP). It investigated effects of the *Failure Mode and Effects Analysis* (FMECA) productivity improvement processes on the performance of air traffic management as reported by Zeh et al. [11].

The objective of the simulations was to provide the test-subjects of the study, trained air traffic controllers, with a realistic environment to assess the improvement process under evaluation. This was accomplished by the detailed simulation of the Hamburg (EDDH) Airport, the interactive simulation of realistic air traffic, and the integration of commercial air traffic control tools into an authentic control room environment and workflow. The simulation scenario and control room environment are described in detail in the Appendix of this paper.

The previous work was extended to produce the results presented in this paper. The objective was to increase knowledge on how the air traffic management performance of the human controllers can be understood and modeled with the intention to develop automatic decision support tools in the future. To this end, the same scenarios were executed with a rule-based simulated controller replacing the human controllers.

The study was conducted using commercial-of-the-shelf air traffic management software. Controllers managed aircraft by entering messages into the air traffic management tool, and the interactive air traffic simulation responded in the same manner. All simulation modules and tools were connected through our X23 SWIM implementation.

5.1. Integration of COTS and Custom Air Traffic Management Software. The simulation experiments used a combination of academic simulation tools and commercially available air traffic management software deployed on commercial-of-the-shelf hardware. The NAVSIM research air traffic simulator (developed by "Mobile Communications R&D GmbH, Salzburg" in cooperation with University of Salzburg) was used for the interactive simulation of realistic air traffic. The controllers taking part in the experiment used the commercially available smartStrips electronic flight strips tool (developed by Frequentis AG) as human machine interface to interact with the simulated air traffic. smartStrips was augmented by a simulated radar display (NAVSIM ATC, developed by "Mobile Communications R&D GmbH, Salzburg," in cooperation with University of Salzburg) and a generic arrival/departure manager (developed by the University of Salzburg) shown on auxiliary screens. The human controllers were also provided with a virtual tower view based on the graphics engine of the X-Plane simulation software (developed by Laminar Research). This virtual tower view was projected onto a 270° cylindrical screen.

The interface to X23 was implemented differently for each application. The NAVSIM simulator was extended with an X23 interface by the addition of mapping logic mediating between its internal representation of AIRM entities and X23 objects. The interface between smartStrips and X23 was implemented in a bridge class extending the smartStrips application with similar mapping logic. Frequentis AG provided guidance and compiled Java classes for this purpose. The generic arrival/departure manager was built directly on top of the X23 object cache. No mapping was therefore required in this case. The X-Plane software supports plugins that can be loaded at runtime. The X23 bridge was thus implemented by compiling the X23 stack and the required mapping logic into an X-Plane plugin. In all three cases the implementation of the mapping logic was straightforward as it concerned only the mediation between objects implementing the same semantic model.

The information flow between the simulation components is illustrated in Figure 7. NAVSIM ATC, smartStrips, and the arrival/departure manager are actually present five times in the experiment setup, once for each human controller. The virtual tower application is present four times, once for each projector to achieve a 270° field of view. Note that the update messages of the simulation events and

Table 1: Controller performance for EDDH "high traffic" scenario, 60 A/C per hour, 1 h duration.

	Human controllers		Simulated controller		
	Avg.	Stdev.	Avg.	Stdev.	
Delay (departures)	−64.78	446.06	10.55	145.66	s
Taxi time (arrivals)	253.24	61.95	295.37	20.44	s
Taxi time (departures)	141.69	38.38	135.77	39.27	s
Taxi distance (arrivals)	2372.48	393.47	2547.56	225.55	m
Taxi distance (departures)	1887.56	324.5	1795.91	324.10	m

commands were not only distributed over the network, but also stored in log files for further analysis. All computers in the network were synchronized via NTP to have consistent time-stamps in the log files.

5.2. Simulated Controller. In the extended experiment the human controllers were replaced by a rule-based simulated controller. Note that the only necessary change to the simulation setup was to replace the smartStrips tool by the simulated controller. Due to the SWIM-based simulation network, no changes to the other parts of the simulation were required.

The rules for the simulated controller were derived from the controller training material of Hamburg Airport which take specific aspects of the airport into account. The selection of the appropriate departure runway holding point was based on taxi time duration, aircraft type, and available runway takeoff length. The issuing of engine start-up clearances was based on taxi time duration and calculated takeoff time taking the current airport situation into account; that is, the rules were formulated to issue "just in-time" start-up clearances to avoid unnecessary fuel consumption, reduce CO_2 emissions, and avoid congestion at runway holding points. Runway line-up sequencing was then performed on the basis of the calculated takeoff time. The takeoff clearance was issued according to the wake classes of previously departing, arriving, or landing aircraft. If reassignments of the gates and parking positions were necessary, they were calculated during the approach phase on the basis of the currently available gates and the wingspan of the aircraft. Landing clearances were issued taking currently departing and arriving aircraft into account. The taxi routes of arriving aircraft were calculated to minimize the taxi time duration in the current airport situation. This included runway crossing clearances giving way to landing and departing aircraft.

The key performance indicators of the taxiing performance of the human and simulated controllers for one selected simulation scenario are displayed in Table 1.

6. Discussion

Air traffic management tools and procedures will increasingly interface using SWIM. It is therefore important to evaluate them in a realistic SWIM-compliant simulation environment to ascertain the validity of the results. Our work shows that extending the SWIM air traffic information reference model to include simulation specific information is straightforward and does not conflict with SWIM-compliance. The benefits of using SWIM as simulation interface were demonstrated with the integration of commercial air traffic management software into our research simulation.

Our method and X23 SWIM implementation were applied to human-in-the-loop simulation experiments investigating the application of productivity improvement processes to air traffic management and evaluating a rule-based controller model against the performance of human air traffic controllers.

The comparison of key performance indicators of the air traffic management performance indicates a good agreement of the rule-based model with the human controller performance. It is, however, noteworthy that the rule-based model accrued significantly later off-block times than the human controllers. The controllers would typically plan ahead and let aircraft leave the gate shortly before their planned departure times. This was not reflected in the simulated controller rules as they were formulated with the intent to meet the calculated takeoff times as precisely as possible.

The flexibility of our multicast communication approach proved of great practical value. Joining and removing simulation nodes at runtime were helpful in the debugging of the simulation software. Not relying on a configured deployment network allowed the live demonstration of the simulation environment at several external occasions.

We were surprised by the quality of the scenery and navigation databases used in the different commercial modules of the simulation. The air traffic management simulation and the virtual tower used independent databases for the geographic layout of the airport. The navigation data for the air traffic management simulation was provided by the European air traffic authorities through SESAR. The origin of the X-Plane 10 scenery database, which is used by the virtual tower, is unclear. Nevertheless the scenery and navigation databases matched extremely well down to the level of individual taxi lanes.

The simulation method presented in this paper provides the features required for the SWIM-compliant distribution of simulation state and simulation events in loosely real-time synchronized human-in-the-loop simulations running in a single local area network. It does not provide nor intend to provide the scalability or security features required for large-scale pan-European SWIM networks. It is intended for quick deployment and rapid prototyping in research laboratory settings.

It does also not provide or intend to provide the rich distributed simulation capabilities of frameworks like HLA. As of now, our approach is limited to wall-clock time aligned human-in-the-loop simulations because no sophisticated simulation time management was implemented. In theory, this could be added; however, it is questionable whether the coupling of commercial-of-the-shelf air traffic management

software would still be possible with similarly low effort in this case. Commercial-off-the-shelf air traffic management software is obviously designed to operate in wall-clock time and has no concept of simulation time.

Our method is functionally equivalent to the human-in-the-loop simulation environments presented by Prevot et al. [7] and Edinger and Schmitt [8]. The major difference is the use of the semantic model the air traffic management domain is currently converging to. It contributes therefore to the realization of the need to support standard air traffic management interfaces identified by Kaltenhäuser [5].

With regard to the implementation our method builds on the work of Shifeng and Danxia [9] to use multicast to distribute the simulation state. However, we go one step further by using multicast to build a completely decentralized simulation communication architecture.

The method presented in this paper is currently limited to the aeronautical domain because of the use of AIRM. This model is not applicable to other application domains. However, application domains where similar semantic models are specified can also be covered by our approach with appropriate adaptations.

The limitations of the evaluation of the controller model lie in the limited number of experiments that could be performed with human controllers to collect the data. In total, four measured experiments of approximately one-hour duration each have been performed. Within this paper only the "high air traffic" scenario was presented.

7. Conclusion

In this work we presented a SWIM-compliant method for human-in-the-loop air traffic management simulation that uses the air traffic management information reference model AIRM as semantic interface and multicast message passing as communication interface. By this, it enables the evaluation of SWIM-compliant air traffic management software in research laboratory settings without the need for specialized infrastructure.

Air traffic management tools and procedures will increasingly interface using SWIM as standard interface. It is therefore of high interest to evaluate them in a SWIM-compliant simulation environment to ascertain the validity of the evaluation.

It has been shown that our implementation enables the use of commercial-of-the-shelf air traffic management software in simulation experiments improving the applicability of the results. Our use of multicast allows distributed simulation environments without a dedicated SWIM communication infrastructure. This makes deployment more rapid and more robust.

Our method was applied to human-in-the-loop simulations experiments evaluating the performance of a rule-based controller model against human controllers in Hamburg Airport operations. Future work will investigate enhancing the controller model towards an automated optimization and decision support tool.

Table 2: The flight BER561K is created in the simulation approximately eight minutes before arriving at EDDH.

<create_flight	
Attribute	Description
source="navsim_tg_accs" sequence_number="135" sender_uid="8437c549bc2eab0d"	X23 protocol header
time="57303.0"	Time in seconds since midnight
id="6" flt_number="BER561K"	Unique id and callsign of the flight
cocr_phase="arrival" cocr_domain="TMA"	Flight phase and ATC domain
wpt_current="-" wpt_next="PISAS" wpt_next_timeover="57420"	Current and next navigation waypoint
pos_lat="53.727817" pos_lon="9.91923" pos_alt="914.4" pos_heading="69.126" pos_climb_rate="0" pos_climb="0" pos_bank="0" pos_speed_tas="118.322" pos_speed_gs="118.322" pos_speed_ias="111.625"	Current position, orientation, and speed of the flight
desired_alt="914.4" desired_heading="69.126" desired_speed="118.322"	Desired altitude, heading, and speed
flt_aobt="52510.435" flt_eobt="52500" flt_ctot="52860" flt_stot="52860"	Actual and estimated off-block time; calculated time of takeoff
apt_dept_id="LOWW" apt_dept_time="52860" apt_dept_rwy="29"	Departure airport and runway
apt_dest_id="EDDH" apt_dest_time="57780" apt_dest_rwy="23"	Destination airport and runway
ac_type="B733" ac_gear="up" ac_wake="M" ac_squawk="A0001" flt_ac_op="BER"	Aircraft type, gear-status, wake-category, squawk, and operator
navsim_ac_color="65535" navsim_ac_label="___8min"	Display color and label
/>	

Appendix

A. Example X23 Message of Flight Object Creation and Update

The creation of a simulated flight approaching EDDH airport is illustrated in Table 2. Note that although the simulation time is aligned with the wall-clock time, the offset of the simulation time to the wall-clock time is included in the

Table 3: The flight BER561K is updated with a new position.

<set_flight	
Attribute	Description
source="navsim_tg_accs" sequence_number="224" sender_uid="8437c549bc2eab0d"	X23 protocol header
id="6"	Unique flight id
time="57304.11"	Updated time
pos_lat="53.728237" pos_lon="9.921093" pos_heading="69.128"	Updated position and heading
/>	

Table 4: The flight BER561K is contacted by the GND controller.

<cpdlc_message	
Attribute	Description
source="smartstrips" sequence_number="36" sender_uid="8437c549bc2eab0d"	X23 protocol header
flight_id="6"	Unique flight id
cpdlc_message_standard="FANS-1/A" cpdlc_message_uid="UM117" cpdlc_message_param="GND;0"	CPDLC message standard, message type, and message parameters
/>	

message to allow the postprocessing of aircraft arrival and departure times.

While the aircraft is cruising, the current position and some other attributes of the flight are updated via <set_flight ... /> XML elements. Note that only changed attributes are updated as illustrated in Table 3. Unchanged attributes are not retransmitted in the <set_flight ... /> message. When the aircraft finally arrives at its destination airport, the flight object is removed from the simulation (after taxiing, etc.) with the <delete_flight id="..." /> XML message (not shown in the table).

B. Example X23 Message of FANS-1/A CPDLC Message

Table 4 provides an example where the flight BER561K is contacted by the ground controller using the FANS-1/A UM117 "CONTACT [unitname] [frequency]" message. In the CONTACT command, the ground controller indicates the "0" voice frequency enforcing data link operations. The simulated flight has then to respond with a "WILCO," "UNABLE, or "STANDBY" message to the request (not shown in the table).

C. EDDH Simulation Scenario

Hamburg Airport (EDDH) is a typical medium-sized European airport. It has two crossed runways for takeoff and landing and two apron areas for parking aircraft. Figure 8 displays the layout of the airport as it was simulated.

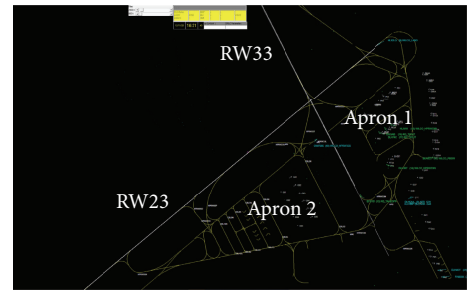


Figure 8: Hamburg Airport. RW33 is approximately oriented in north-south direction. Apron 1 is to the east of RW33 and apron 2 to the west; both aprons are south of RW23; screenshot from NAVSIM air traffic simulator as presented to the air traffic controllers.

The NAVSIM air traffic simulator was used to simulate the air traffic at Hamburg Airport. The air-ground communication with the simulated aircraft is performed at LAN speeds and does not yet involve an air-ground data link emulator introducing the increased communication delay expected by the wireless air-ground link. NAVSIM is an event-driven simulator providing detailed worldwide runway-to-runway (resp., gate-to-gate for those aerodromes for which taxi and gate information is available) air traffic simulation; it offers an X23 interface and can be integrated with human-in-the-loop simulation environments for human factors experiments [15], the development of novel meteorological air traffic routing algorithms [14], aeronautical communication frequency planning, and aeronautical communication volume estimation [16].

The simulated aircraft have a flight management system that simulates the movements of the aircraft according to the characteristics of the aircraft type. This flight management system is controlled by a set of rules reflecting the flight plan of the aircraft. The simulated pilot will therefore contact the controller for clearances of the next steps of the flight plan. Having received a clearance the pilot will implement the flight-plan using automatic path-finding, following the rules for taxiing on an airport and avoiding obstacles like other aircraft and buildings. The flight-plans used in the simulation exercise were based on real flight-plans of Hamburg Airport, but modified to create stressful situations for the controllers by adding additional flights.

In the presented simulation scenario the number of aircraft was set to 60 aircraft per hour, which is close to the theoretical maximum capacity of Hamburg Airport. Instrument meteorological conditions were used during all simulation runs. The simulated wind component was 10 kts from 270. Due to this, the simulation operated with IFR traffic only.

Five trained air traffic controllers were assigned the task to manage aircraft arriving and departing from Hamburg Airport and the movement of all aircraft taxiing on the ground. Air traffic sectors adjacent to Hamburg Airport were automatically controlled by the NAVSIM air traffic

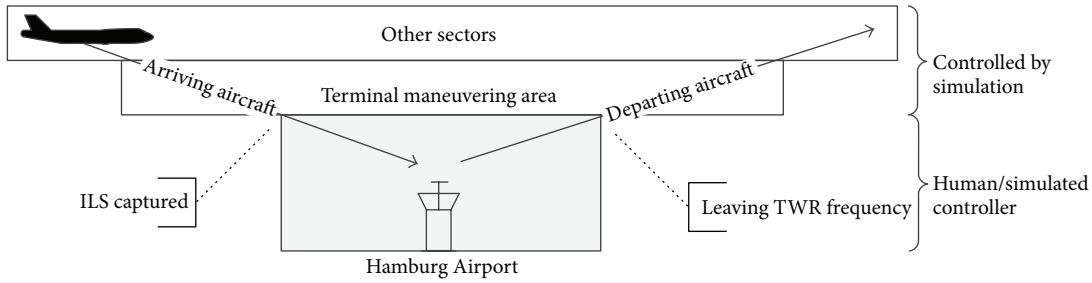


Figure 9: Distribution of air traffic management responsibilities between the air traffic simulation and the human controllers/simulated controller.

Figure 10: Control room of the Hamburg Airport human-in-the-loop simulation exercise. Each controller workstation is labeled with the controller's function in the experiment.

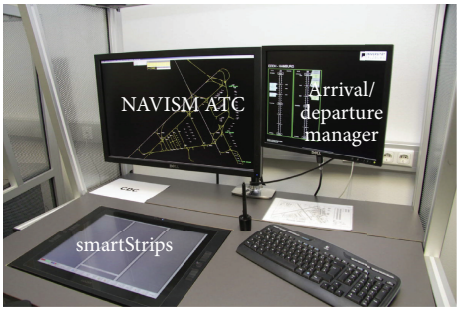


Figure 11: One of the five controller working positions.

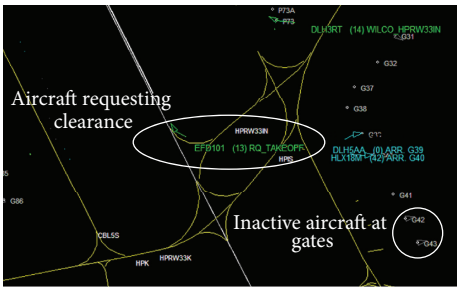


Figure 12: NAVSIM ATC simulated radar display. Aircraft requesting clearances are highlighted in color. Inactive aircraft are muted in grey.

simulator. The traffic in the Hamburg terminal maneuvering area was also controlled by the air traffic simulator. Arriving aircraft were handed off to the human Hamburg tower controller when capturing the *Instrument Landing System* (ILS); clearance-delivery, apron 1, apron 2, and the taxiways were also controlled by humans; departing aircraft were handed off by the human controllers to the air traffic simulator when leaving the tower frequency. This is illustrated in Figure 9.

The control room used for the experiment is displayed in Figure 10.

D. Simulation Human Machine Interface

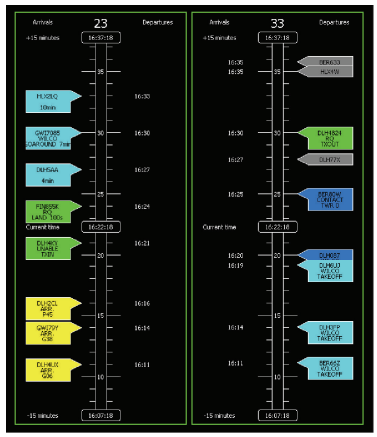
D.1. Controller Workstations. The controller working positions comprised a radar screen, a flight strip tool, and an arrival/departure manager. The radar screen was a modified version of the NAVSIM air traffic simulator stripped down to the graphical user interface for the display of the simulated radar image.

The electronic flight strips tool "smartStrips" was supplied by Frequentis AG and used to display the state of the simulated aircraft and to exchange CPDLC messages with the simulated pilots. Command messages generated by the flight strips tool were sent over the simulation SWIM network to the air traffic simulator to request the appropriate aircraft behavior.

The auxiliary screen was generic implementation of an arrival/departure management tool providing the controller with increased time awareness. A controller working position is displayed in Figure 11. The radar screen is on the top left, the flight strips tool on the bottom left, and the arrival/departure manager on the right.

The airport controllers were supported in their task by a nonintrusive level of automation. The controller user interfaces were extended to display state information, using additional text and color codes; for example, aircraft requesting clearances were highlighted while aircraft at parking positions were displayed in muted colors.

A detailed view of the radar screen is shown in Figure 12. It displays the runways, taxiways, and parking positions of the airport. Aircraft are displayed with a position vector and a label indicating their callsign and a unique simulation internal identifier for postprocessing. Altitude and heading can be displayed optionally. In our simulations additional



(a) (b)

Figure 13: Simulated arrival/departure manager. The timeline for RW23 is on (a); the timeline for RW33 is on (b). The top of the timeline is 15 minutes in the future; the bottom is 15 minutes in the past. Arriving aircraft are displayed on the left of timeline (a) and departing aircraft on the right of the timeline (b).

information was displayed within the label to assist the controller. Aircraft with unanswered requests were highlighted by changing the color and the last CPDLC request was displayed. Flight state and communication state originate from different simulation modules but were consolidated through our X23 SWIM implementation.

The auxiliary screen showing the arrival/departure manager is displayed in Figure 13. This tool provided the controller with a time line for each active runway. In our experiments runway 23 was used for arrivals and runway 33 was used for departures; hence two time lines were displayed. Flight labels on the time line provided the controller with time awareness for the planned arrival (a) and departure (b) times of the past ten minutes and the future fifteen minutes, that is, one-slot duration. The labels used identical color-coding as on the radar screen and display the last data link messages.

The data link commands for air traffic control were entered into the electronic flight strips tool shown in Figure 14. Each flight is represented by a flight strip autogenerated from the SWIM data, and flight strips could be shared by different controllers or stored in bays only accessible to a single user. Commands entered through the user interface were injected into X23 SWIM where they could be accessed by the other simulation components, for example, the air traffic simulator responding to the commands or the various human machine interfaces.

D.2. Virtual Tower. In addition to the individual controller working positions a virtual tower view was provided to all controllers. The virtual tower view comprised a 3D visualization of the Hamburg Airport projected onto a 270° screen as shown in Figure 15. The 3D view was generated by an adapted version of the commercially available X-Plane 10 software integrated into the simulation's SWIM network. Color-coded labels attached to each flight enhanced the rendering.

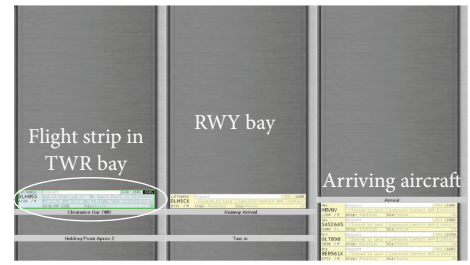


Figure 14: Electronic flight strips (smartStrips by Frequentis AG). The electronic flight strips are dynamically generated and updated according to the progress of the simulation. Flight strips are placed in different bays according to controller responsibility. The bay configuration displayed in the image is used by the *tower* (TWR) and *ground* (RWY) controller. The tool uses the X23 SWIM interface to connect to the simulation.

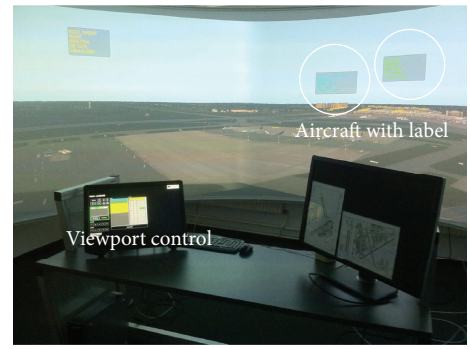


Figure 15: Virtual tower view of Hamburg Airport (X-Plane by Laminar Research). The viewport control is used to pan and zoom the projection. Aircraft are augmented with labels containing air traffic management information.

The SWIM interface of the virtual tower was implemented in the Delphi programing language. We used the X-Plane 10 plugin interface to access the simulator's 3D rendering engine. The X-Plane 10 simulation logic was not used. Commercially available high-detail Hamburg Airport scenery was used; therefore our plugin focused on the display of aircraft 3D models and air traffic management information at the positions and with the orientation provided by the simulation. In the current implementation three models, small (Beach Baron), medium (A320), and large aircraft (B747), were used to approximate the aircraft type. To provide smooth display of aircraft movements, position and orientation were interpolated with Catmull-Rom splines between simulation state updates according to Barry and Goldman [17]. In addition to the full 270° view, the plugin supported zooming in on a selected aircraft. This is similar to how a controller would use binoculars in the tower.

D.3. Flight Simulator (Not Used in EDDH Simulation Scenario). In addition to the air traffic simulation piloted flight simulators can be used as additional sources of air traffic, too. As of now the X23 stack has been outfitted with an application interface to the Diamond DA-42 flight simulator displayed

(a) (b)

Figure 16: (a) Diamond DA-42 flight simulator and (b) X-Plane flight simulator coupled with X23 into the EDDH scenario.

in Figure 16(a) and the X-Plane flight simulator shown in Figure 16(b). Using these simulators a human pilot can take part in the simulation indistinguishable from the simulated air traffic for the controllers.

Acronyms and Abbreviations

AIRM: Air Traffic Management Information Reference Model
AMQP: Advanced Message Queueing Protocol
API: Application Programming Interface
ATM: Air Traffic Management
DDS: Dynamic Distribution System
HLA: High Level Architecture
HMI: Human Machine Interface
HTTP: Hypertext Transfer Protocol
IP: Internet Protocol
LAN: Local Area Network
NTP: Network Time Protocol
OSI: Open Systems Interconnection
SESAR: Single European Sky Air Traffic Management Research program
SWIM: System Wide Information Management
UDP: User Datagram Protocol
WAN: Wide Area Network
WSDL: Web Services Description Language
XML: Extensible Markup Language.

Disclosure

Opinions expressed in this work reflect the authors' views only, and EUROCONTROL and/or the SJU shall not be considered liable for them or for any use that may be made of the information contained herein.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was cofinanced by EUROCONTROL acting on behalf of the SESAR Joint Undertaking (the SJU) and the EUROPEAN UNION as part of Work Package E in the SESAR Programme. The authors are particularly grateful to the controllers who participated in the experiment for their valuable effort and patience. They also thank Elias Pschernig for the implementation of the X-Plane 10 plugin. They thank M. Schnell from DLR (German Aerospace Center) and K. Eschbacher from the University of Salzburg for their helpful comments. Parts of this work were performed while Thomas Gräupl was an employee of the University of Salzburg.

References

[1] L. Da Xu, "Enterprise systems: state-of-the-art and future trends," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 630–640, 2011.

[2] J. S. Meserole and J. W. Moore, "What is system wide information management (SWIM)?" *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, no. 5, pp. 13–19, 2007.

[3] J. P. McGee, R. Parasuraman, A. S. Mavor, and C. D. Wickens, *The Future of Air Traffic Control: Human Operators and Automation*, National Academies Press, 1998.

[4] A. Zellweger, "Simulation in CNS/ATM research with examples from the United States," *Simulation Modelling Practice and Theory*, vol. 11, no. 3-4, pp. 173–185, 2003.

[5] S. Kaltenhäuser, "Tower and airport simulation: flexibility as a premise for successful research," *Simulation Modelling Practice and Theory*, vol. 11, no. 3-4, pp. 187–196, 2003.

[6] M. Adelantado, "Rapid prototyping of airport advanced operational systems and procedures through distributed simulation," *Simulation*, vol. 80, no. 1, pp. 5–20, 2004.

[7] T. Prevot, N. Smith, E. Palmer et al., "The airspace operations laboratory (AOL) at NASA ames research center," in *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, pp. 25–74, August 2006.

[8] C. Edinger and A. R. Schmitt, "Rapid prototyping for ATM operational concept development," in *Deutscher Luft- und Raumfahrtkongress*, Berlin, Germany, September 2012.

[9] M. Shifeng and W. Danxia, "Implementation of a flight control tower simulator using commercial off-the-shelf hardware," *SIMULATION*, vol. 86, no. 2, pp. 127–135, 2010.

[10] S. Wilson, R. Suzic, and S. van der Stricht, "The SESAR ATM information reference model within the new ATM system," in *Proceedings of the Integrated Communications, Navigation and Surveillance Conference (ICNS '14)*, pp. L3-1–L3-13, IEEE, Herndon, VA, USA, April 2014.

[11] T. Zeh, V. Grantz, S. Kind et al., "Zero failure management at maximum productivity in safety critical control room," in *Proceedings of the 1st SESAR Innovation Days*, December 2011.

[12] M. Stal, "Using architectural patterns and blueprints for service-oriented architecture," *IEEE Software*, vol. 23, no. 2, pp. 54–61, 2006.

[13] S. Glickman, "The business case for system-wide information management," *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, no. 10, pp. 3–12, 2007.

[14] T. Hauf, L. Sakiew, and M. Sauer, "Adverse weather diversion model DIVMET," *Journal of Aerospace Operations*, vol. 2, pp. 115–133, 2013.

[15] J. A. Langlo, A. W. Eide, A. Karahasanovic et al., "Usefulness of FMECA for improvement of productivity of TWR process," in *Proceedings of the 2nd SESAR Innovation Days*, vol. 2012, November 2012.

[16] M. Carandente and C.-H. Rokitansky, "VDL Mode 2 Capacity and Performance Analysis," 2015, http://www.sesarju.eu/sites/default/files/documents/news/SJU_VDL_Mode_2_Capacity_and_Performance_Analysis.pdf.

[17] P. J. Barry and R. N. Goldman, "A recursive evaluation algorithm for a class of Catmull-Rom splines," *Computer Graphics (ACM)*, vol. 22, no. 4, pp. 199–204, 1988.