IPSME- Idempotent Publish/Subscribe Messaging Environment

Kim Nevelsteen
MiTM AB
Uppsala, Sweden
kim.nevelsteen@acm.org

Martin Wehlou
MiTM AB
Uppsala, Sweden
martin.wehlou@mitm.se

Abstract
The integration (interoperability) of highly disparate systems is an open topic of research in many domains. A common approach for getting two highly disparate systems to be interoperable, is through an agreed-upon protocol (e.g., via standardization) or by employing a common framework. The problem of integrating systems arises when many of these protocols/frameworks come into existence. Both, agreeing on protocols/frameworks and creating mappings between protocols takes time and effort. An interoperability solution must be scalable and should not require stakeholders to adapt to major changes in their system i.e., systems should not need to be re-engineered as other systems are added, removed or replaced in the integration.

IPSME is introduced as a solution for integrating highly disparate systems. IPSME decouples the dependencies between interacting participants. Interoperability is achieved through dynamic translations, with integrations external to the systems being integrated, avoiding the need for agreed-upon protocols or frameworks. Scalability is achieved by not having a limitation on the number of messaging environments or the topological organization thereof. IPSME is minimally invasive and through a network effect reduces the overall complexity of integrating many systems to linear. IPSME has been evaluated and thus far been tested in three use cases.

CCS Concepts
- Software and its engineering → Interoperability: Ultra-large-scale systems; Software evolution; Virtual worlds software; - Computer systems organization → Heterogeneous (hybrid) systems.

Keywords
publish/subscribe; integration; interoperability; system of systems; mapping; scalability; evolution; architecture; metaverse; heterogeneous; systems; internet of things; virtual; idempotence; invocation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Kim Nevelsteen and Martin Wehlou. 2021. IPSME- Idempotent Publish/Subscribe Messaging Environment. In International Workshop on Immersive Mixed and Virtual Environment Systems (MMVE ’21), September 28-October 1, 2021, Istanbul, Turkey. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3458307.3460966

1 Introduction
The integration (‘interoperability’ [2]) of highly disparate systems is an open topic of research in many domains, including Healthcare [1, 12], Computer Video Games [14], Internet of Things [16, 19], Pervasive Applications [17], and the Metaverse [6]. And, by extension Cyber Physical Systems [9], which incorporates Internet of Things and the Metaverse [20]. The domain of this article can be generalized as the integration of virtual environments of disparate systems, potentially forming a ‘system of systems’ [13].

A common approach for achieving interoperability and getting highly disparate systems to communicate, is by having them speak the same protocol. A common protocol can be achieved through ‘standardization’ [22] or by requiring systems to adopt a ‘framework’ [2] as a dependency. The problem of integrating systems arises when many of these protocols/frameworks come into existence. If n distinct systems are to be integrated, then the eventual complexity will be an exponential n(n – 1)/2 [19].

To address the problem, this article provides – not a call for standardization or a framework as dependency, which are too restrictive – but, a set of conventions which enables integration by: ‘decoupling’ [2] dependencies and incorporating ‘interest management’ [16], allowing for mappings between protocols dynamically, specifying the integrations to be external to the systems being integrated, and the division of communicating systems into ‘regions’ [16, 18]. To provide interoperability and decouple dependencies, a publish/subscribe system (pubsub) is used in the set of conventions. Wang et al. [24] state that pubsub is “a communication infrastructure that enables data access and sharing over disparate systems and among inconsistent data models”. Interest management for each system is incorporated by having participants simply drop messages not understood or found interesting. Mappings between protocols can bridge several ‘layers of interoperability’ [10, 19], and in the set of conventions, mappings are specified in a manner so as to resolve schema heterogeneity through ‘decomposition’ [21] of the problem. And, the dividing of systems into regions provides the required ‘scalability’ [2] for the system [16, 19]. The set of conventions together form what is introduced in this article as an Idempotent Publish/Subscribe Messaging Environment (IPSME).

To the best of the authors knowledge, as of this writing, IPSME is the only interoperability solution that is, independent of the systems being integrated, and allows for the integration of n systems with linear complexity, while maintaining scalability.

2 Problem Statement
A common approach for getting two highly disparate systems to be syntactically and semantically interoperable (assuming the systems can communicate i.e., technical interoperability exists [19]), is by having them speak the same protocol. A common protocol can be achieved through standardization (i.e., an agreed-upon protocol
with possibly varying implementations on each system) or by re-
quiring systems to adopt a framework (specifically implemented 
for each system), as a dependency to ensure communication.

The problem of integrating systems arises when many proto-
cols/frameworks (or versions thereof) come into existence. Integrat-
ton is a particular problem when dealing with ‘legacy’ [13] systems, 
which are difficult to alter and speak unsupported protocols.

Different protocols can be bridged by creating a mapping be-
tween the layers of interoperability. Mappings must not only satisfy 
syntactical and semantic interoperability, but also the conceptual 
layers of interoperability [19]. A key issue with mappings is that 
they take a considerable amount of effort e.g., “in a typical data 
integration scenario, more than half of the effort (and sometimes 
up to 80 percent) is spent on creating the mappings; the process 
is labor-intensive and error-prone” [10]. There is not necessarily a 
single correct mapping [10] and when protocols are updated, any 
eexisting mappings must also be updated, exaggerating the problem.

3 Related Work

Madni and Sievers [13] specify a list of five interoperability cases 
ranked in increasing order of difficulty. In the most difficult case, 
the systems to be integrated will not have been designed to interop-
erate, with no new system development underway to incorporate 
interoperability, and systems must be made to interoperate with 
minimal redesign/modification. The latter criteria correlates to the 
open challenge for interoperability in Internet of Things, which 
calls for a scalable integration, where stakeholders should not have to 
“adapt to major changes in their system; the solution should 
not be dependent on their system” [19]. This criteria is also one of 
the two principles for ‘dynamic evolution’ [15, 21] in the domain 
of System of Systems Integration i.e., systems do not need to be 
re-engineered as other systems are added, removed or replaced in 
the integration. The other principle required for dynamic evolu-
tion being that the complexity of the integration doesn’t grow as 
systems are added.

When faced with the problem of integrating highly disparate sys-
tems, many [1, 3, 12, 14] argue that interoperability can be achieved 
through standardization, but standardization only has limited suc-
cess [10, 21]. Shapiro and Varian [22] go into detail on the ad-
vantages and disadvantages of standardization. Standardization is 
a slow and limiting solution which can stifle fast paced innovation 
[22]. It is impossible for a standardization committee to foresee 
all possible usages of a given standard. A considerable amount of 
time is required to create/update a standard [19]. The standard 
can be outdated before it is standardized and the result does not 
necessarily satisfy all ‘stakeholders’ [2, 13] i.e., features lacking at 
the time of standardization or the standard bloated with feature re-
quests from various stakeholders. Updating a standard risks having 
various versions of the same protocol in use simultaneously and 
the existence of legacy systems [21].

Rather than agree on a standard, stakeholders can create a frame-
work with the expectation that all others will conform, leading to 
an abundance of interoperability frameworks. A framework must 
provide an implementation for each platform and/or programming 
language supported. When a framework is updated, each system 
using the framework must be updated or risk becoming a legacy 
system.

4 Introducing IPSME

IPSME is introduced to enable the integration of highly disparate 
systems. Rather than a framework or standardized protocol, IPSME 
is a set of conventions that forms an ‘architecture’ [2] where any 
participant to talk to any other participant, without need for a 
central authority and without standardization, provided groups of 
participants speak the same protocol.

IPSME defines the following conventions:

• A messaging environment (ME) is defined as:
  - A pubsub system which receives messages and relays those 
messages to all subscribed participants;
  - Messages must be idempotent or identifiable as duplicates.
• Each participant:
  - sends and receives messages in a (local) ME;
  - simply ignores messages, if not understood.
• Translation of messages is done by having a participant listen to 
messages on a (local) ME and sending out translated messages.
• Communication across ME boundaries is through a reflector 
pair: a participant listener with a counterpart in another ME, 
which relays messages there. Communication between reflec-
tors is left unspecified and completely up to the author(s) of 
the reflectors, as is the selection of messages to resend.

The author(s) of a participant is free to implement their own mes-
gage format as long as the above conventions are met. The set of 
conventions is purposely kept non-restrictive for easy adoption.

4.1 Interoperability

A local ME employs the usage of a readily available pubsub resource. 
On the various operating systems, there is usually a platform-
specific messaging system for inter-process communication (IPC) 
where pubsub can be used e.g., NSNotificationCenter on macOS/iOS 
and MSMQ on Windows. If a platform-specific messaging system 
is not available, any other available pubsub can be used, as long as 
all participants that are to be local to that ME, know how to access 
the ME. If more than one pubsub is utilized on a platform and they 
are to interact, they must be interconnected by a pair of reflectors. 
Participants of a ME are usually processes running on the platform. 
Participants can be both publishers and/or subscribers in a ME. The 
pubsub in a ME serves no other purpose than to relay published 
messages by broadcasting them to subscribers.

Because pubsub is a broadcast system, messages in IPSME are 
required to be ‘idempotent’ [4], so as to promote asynchronous 
communication, by reducing the amount of acknowledges required, 
and that identifiable duplicates can be eliminated. If messages are 
passed across multiple MEs a universally unique identifier (e.g., 
UUID or GUID) can be employed to help achieve idempotence. Idem-
potent messages can be processed multiple times by a participant, 
but the processing of each message must give the same result after 
the application of the initial message. Because messages are broad-
cast using pubsub, messages are the ‘implicit invocation’ [8] of 
potentially multiple participants. This does add more complexity to 
the sender, since the sender must handle zero or multiple responses 
e.g., pick the best response or reply to all the responses within a 
given timeframe.
Rather than trying to obtain interoperability by enforcing a predefined structure or data model in messages [7], IPSME does not specify a format for message content i.e., it is possible to use strings, binary or any of the topic-, content- or type-based pubsub schemes [7]. By not specifying message content IPSME avoids having a predetermined ‘expressiveness’ [5], perhaps having many simultaneously. Participants send messages in their own protocol and only participants that understand those messages will be able to process them i.e., partitioning the semantic and syntactic space into any number of separate spaces. One of the main tenets that enables interoperability for IPSME is the interest management of each participant; if a participant does not understand a message, it simply drops the message and continues processing. This can lead to scenarios where certain participants might want affirmation that another participant has received the message. It is possible to send return messages (e.g., Remote Procedure Calls or Acknowledges) through IPSME, but such a return is not defined in the IPSME specification. The IPSME conventions are minimally invasive for existing systems, since those systems can continue to speak the protocol that was previously implemented, but can be externally bound to a ME.

Any authentication or message security [23] is up to the author(s) of the participants i.e., security is left as peripheral to this discussion, but has been taken into consideration during the design of IPSME. If standard and/or central authentication services are required, such a service can be provided through the use of a translator.

### 4.2 Mappings

If each participant sends messages in their own protocol, communication is limited to the number of participants that understand the sent messages. To broaden the set of participants that understand a message, specialized participants that translate messages can be inserted in to a ME. These translators listen for messages that adhere to a certain protocol, translate them to a different protocol (i.e., a mapping) and send out the translation; translating participants are considered mediators between other participants.

Between all participants of a local ME and through to other MEs via reflectors, translations have a *network effect*. Translations are transitively applicable to other participants e.g., if a translation X translates from participant A to B, then any participant C, that can communicate with A, can also communicate with B, via X. The network effect of translations has the potential to reduce the complexity of integrating n participant nodes from an exponential \( n(n - 1)/2 \), to a linear \( (n - 1) \).

The use of translators in this manner means IPSME alleviates the problem of resolving schema heterogeneity through decomposition of the problem. The translators can collectively divide the problem vertically, horizontally [23] or even incrementally. A major advantage of this architecture is a human related one. Authors of participants have a very limited scope of other communicating participants they must take into account.

Participant communication is not constrained to be via a ME. Participants can negotiate to communicate directly allowing for ‘explicit invocation’ [8]. It is the responsibility of the participants to negotiate such a communication, and beyond the IPSME specification.

### 4.3 Scalability

Through the use of pubsub, IPSME decouples the production and consumption of messages, increasing scalability by decoupling dependencies (i.e., time, space or synchronization) between interacting participants. IPSME is not fixed to specific properties such as expressiveness [5] or those related to quality of service [7], that can affect scalability. A single ME can take advantage of being centralized, but (for the integrated systems as a whole) a centralized architecture or a hierarchical topology should be avoided so as to promote scalability; a centralized architecture being a bottleneck and single point of failure, and a hierarchical topology having possible performance problems [5].

Expecting all prospective participants to be connected to the same ME is impractical; not all prospective participants would easily route to a single ME and a single ME would certainly be overloaded. Participants can be divided (i.e., into regions) using multiple MEs. IPSME specifies participants should connect to a local ME, but places no limitation on the number of MEs or the ‘topology organization’ [23] thereof. IPSME specifies reflectors for communication across ME boundaries, and it is this communication that allows MEs to be connected and organized into a general graph topology.

A reflector is a participant in a ME that listens and filters for particular messages that should be routed to another ME. The reflector communicates directly with a reflector (i.e., its counterpart) in the other ME. Upon receiving a message, the counterpart publishes the message to its local ME; participants of that local ME will receive messages from the distant ME transparently i.e., without being aware of any communication complexity thereof. Communication between two reflectors is left as undefined and is completely up to the author(s) of those participants. A reply message is routed back through reflectors in a reverse fashion. Similar to how adding translators adds functionality without changing the existing system, reflectors can also be inserted into a ME without changing existing implementations; to the participants using a reflector the ME is simply expanded.

### 5 Evaluation

IPSME has thus far been tested in the following three use cases: the *Minecraft/Doom integration*¹, *Medical Resource Scheduling*², and a *Minecraft-Metaverse via MiM*³.

A proof-of-concept dubbed *Metaverse prototype⁴* was a precursor to the three use cases, integrating the video games Minecraft and Doom3, and the virtual world of LambdaMOO. The precursor did not use IPSME; the integration between Minecraft and Doom, and the integration between Minecraft and LambdaMOO were implemented directly in each participant. No integration from Doom to LambdaMOO was achieved.

### 5.1 Minecraft/Doom integration

In the Minecraft/Doom integration use case (depicted in Sequence Diagram 1), both Doom and Minecraft were slightly altered to: connect to the local ME, and expose an API for teleporting in and

---

¹Minecraft/Doom integration forming a Metaverse: https://youtu.be/knKZd15rhJE
²Medical Resource Scheduling: https://youtu.be/m_b1PotByx0
³Insane Minecraft Teleportation, in VR, via MiM: https://youtu.be/ORWto-Oo1W4
⁴Metaverse prototype verbose demo: https://youtu.be/eKJMu5Pu-w8
out of their respective virtual environments. A single local ME was used, with the following components attached to it: Doom3, Minecraft (MC), a trivial universal interface component (UEE) and a Minecraft translation component (TxM). The Doom3 translation participant (TxD) was implemented in the UEE component. In hindsight, it would have been sufficient for the translation components (TxM and TxD) to call the Doom3 and MC APIs directly, reducing the required changes to Doom3 and Minecraft. Although all components were on the same operating system (i.e., macOS), communication spanned two programming languages: Java for MC and TxM, and C++ for the ME, UEE and Doom3. TxM was responsible for translating the custom Minecraft protocol to a custom Hyperjump protocol i.e., syntactical mapping. UEE understood the Hyperjump protocol and was also responsible for translating to the custom Doom3 protocol. It should be noted that each signal in Sequence Diagram 1 is a broadcast to all other components through the local ME. A signal is a reply to the signal that precedes it, even though no arrow is depicted showing the arrival of the broadcast.

Minecraft was altered so that when a player enters a portal, a PubabPortalPlayerIn message is broadcast, which is understood by TxM causing it to subsequently send out a Hyperjump message. UEE accepts the Hyperjump message and sends out a translated message, with the Minecraft player (player/MC) translated to the corresponding Doom3 player (player/D3), via internal semantic and context mapping of player profiles and inventories. TxD understands the Hyperjump message and translates the message to the Doom3 3-respawn-player message. It would have been sufficient for TxD to call Doom3 via an API call, but instead Doom3 accepts the 3-respawn-player message via ME and spawns the player. An acknowledge is sent back through in reverse order, so that the player can be removed from the Minecraft virtual environment. A teleport from Doom3 to Minecraft is handled similarly.

Even with such a small number of participants, the benefit of the network effect of translations is noticeable. To obtain the complete integration of three participants (i.e., MC, UEE and Doom3), a fully connected translation graph is required; see Figure 1 (left). The two translations provided by TxM and TxD were implemented, but a third translation from MC to Doom3 was achieved through the network effect of transitively applying TxM and TxD. The complexity of the full integration is thereby reduced from $3(3 - 1)/2 = 3$ to $(3 - 1) = 2$.

In the precursor, LambdaMOO was integrated with Minecraft. If LambdaMOO were to be integrated into the Minecraft/Doom integration, using IPSME instead, a MOO participant would be added to the translation graph, with an implemented translation between MOO and MC; see Figure 1 (right). The integration of MOO and UEE, and also MOO and Doom3, would be obtained through the network effect. The complexity of the four fully integrated participants is $(4 - 1) = 3$, rather than $4(4 - 1)/2 = 6$.

### 5.2 Medical Resource Scheduling

The Medical Resource Scheduling use case allowed various Care-Planner components to dynamically find and negotiate with various Op components, medical resource providers for operating rooms. The primary focus of this use case was to test IPSME with respect to: components residing on different platforms and programmed in different languages; the ability of various systems to negotiate
without the need for a central authority; and, test the integrated systems for ‘fault tolerance’ [2].

All components could have registered themselves with a ‘discovery service’ [2], but instead the prototype was simplified by having all components on the same network. Components were made aware of each others existence by sending Announcement and Discovery messages on start; other components replied to a Discovery message with an Announcement. To avoid a central authority, negotiations were handled by a form of ‘bidding’ [25], with transactions between components to avoid race conditions [2].

The pubsub of IPSME ensures there were no dependencies between participants. All interactions were implemented with timeouts and handling the possibility of zero response. The integrated systems were fault tolerant against ‘omission failures’ [11]. A trivial form of ‘redundancy’ [2] was prototyped by keeping two identical copies of an Op running, simultaneously. The integrated systems were then fault tolerant to one Op becoming unresponsive. Since each copy of the Op produced identical responses to incoming messages, the idempotency criteria of IPSME allowed the redundant messages to be dropped.

5.3 Minecraft-Metaverse via MiM

The most recent use case, a Minecraft-Metaverse via MiM (depicted in Sequence Diagram 2), was an attempt to link existing Minecraft server instances, without altering the Minecraft client or server; this meant manipulating the streaming data of the proprietary Minecraft network protocol (a task handled by components with a -MC suffix). The normal stream between client and server was redirected to be: from client (not depicted in Sequence Diagram 2), through the downstream -MC component, DownMC, through one of the upstream -MC components, Up1MC or Up2MC, to a corresponding Minecraft server (not depicted). Every -MC component had a corresponding local ME i.e., the downstream participant, Down, and upstream participants, Up1 & Up2, connected to their corresponding local MEs on the client-side and server-side, respectively. Reflectors dynamically connected downstream participants to any number of upstream participants, forming a star topology for a single Minecraft client. The Protocol Buffers [3] interface description language was used to generate a protocol between upstream and downstream participants. Events (detected in the Minecraft protocol stream or received through the MEs) were shared between Up/Down and -MC component pairs (e.g., Up1 and Up1MC) via IPC. The programming language for all components was Java, but components resided on various operating systems e.g., macOS, Linux and Windows.

Sequence Diagram 2 depicts how, after receiving a MC:Teleport message from Up1, the direct (proprietary Minecraft protocol) communication between DownMC and Up1MC is migrated over to be between DownMC and Up2MC instead. The sequence begins with a successful connection to Up1MC and Up1 i.e., the packet SPacketChunkData being sent from server to client, in normal operation. Up1MC detects the movement of the player in the Minecraft protocol and shares that with Up1 via IPC. Since Minecraft servers were not altered, portal lists were kept in the upstream components. Up1 determines if a player has entered a portal, broadcasting out a MC:Teleport message in that event, which is then passed over to Down/ME via reflectors. Down reads the destination ‘Up2’ out of MC:Teleport and sets up a connection to the potential destination. MC:LookUpQuery is broadcast out by Down, which is reflected to all connected upstream participants. If no valid response to MC:LookUpQuery is received within reasonable time, MC:Teleport is dropped by Down and normal operation continues. As depicted, Up2 receives MC:LookUpQuery, confirms that it owns the exiting end of the portal and broadcasts out a corresponding MC:Response. Reflectors carry the response back to Down triggering: a rebroadcasting of the MC:Teleport message, notifying Up2 of the players arrival; and, an IPC call to DownMC to pause the communication with Up1MC and open direct communication with Up2MC. When DownMC detects a successful connection with Up2MC (i.e., SPacketJoinGame received), the active connection is migrated from Up1MC to Up2MC. A player teleport is simulated originating from the Minecraft server (i.e., inserted in the Minecraft protocol stream, not depicted), and confirmed with a CPacketConfirmTeleport reply from the client. Up2MC notifies Up2 (via IPC), when the

---

5Protocol Buffers: https://developers.google.com/protocol-buffers

Sequence Diagram 2: Minecraft-Metaverse via MiM: an illustration of using direct communication together with IPSME
simulated player teleport is complete. Up2 broadcasts a corresponding MC:Response signaling Down to disconnect from Up1, which was hosting the source end of the portal, and subsequently disconnect DownMC from Up1MC.

### 5.4 Overhead / Worst-case Scenario

An argument against using IPSME might be the overhead of externally implementing the integrations, e.g., broadcast overhead or translation hops. Message passing using pubsub on a local machine can be achieved at roughly the same speeds as IPC i.e., negligible with respect to network transmission times. Integrating processes on local machine via IPSME is then roughly equivalent to implementing the integration directly via IPC, with the exception of the overhead of having to analyze unrecognizable broadcast messages.

The network effect of IPSME means it is possible that a message must go through several translations hops, before being understood by the recipient. Each hop incurs overhead. Given $n$ nodes (i.e., systems) to integrate, the worst-case scenario for IPSME is when the connectivity graph (i.e., MEs connected by reflectors) is fully connected, but only a unique translation path exists, connecting all nodes i.e., a unique protocol between each communicating pair, and the start and end node of the translation path do not implement a common protocol; see Figure 1 (solid lines). Message broadcasting overhead is then maximum; from the start node, the first message would produce a broadcast of $(n - 1)$ messages, and each subsequent translation would incur another $(n - 1)$ broadcast messages, meaning an overhead of $(n - 1)(n - 1)$ broadcast messages (without interest management optimization). Message traversal through the connectivity graph, before the end node understands the translated start node, is also maximum i.e., a message from start must go through $(n - 1)$ translation hops, corresponding to transversal of the entire unique translation path. The latter might seem grim and the quality of the translation might be extremely poor, but if IPSME was not used, such translations would not be possible and a translation would have to be implemented specifically for each combination of protocols; a tactic which reduces overhead in an IPSME architecture also. Massive overhead in the integrated systems can be detected and translations can be created for critical paths i.e., implementations between combinations of protocols so as to reduce overhead. Overhead can also be reduced through the (possibly automated) installation of translations found in the connectivity graph to other systems, provided that the translations are compatible with the intended system. Such an install does not reduce the number of translations, but reduces the translation path.

### 5.5 Discussion

In this section, IPSME is evaluated with respect to the challenges presented in related work. IPSME supports dynamic evolution, satisfying both required principles: in an architecture using IPSME, participants can be added, removed and updated dynamically i.e., without re-engineering other participants; and, it has already been shown that the complexity of IPSME is linear. The scalability of IPSME combined with dynamic evolution answers the open interoperability challenge in Internet of Things, and allows IPSME to be applicable to the most difficult interoperability cases.

In the work by Selberg and Austin [21], standardization is exemplified as a limiting case which dynamic evolution resolves. Each participant in IPSME speaks their own protocol i.e., alleviating the need to foresee all possible usages of a protocol and satisfy all stakeholders. As previously mentioned, a major advantage of IPSME is a human one e.g., stakeholders can write a translation from participant A to B, while not worrying or even knowing about participants C, D, and E talking to A or B. Rather than endure a delay such as when updating a standard, IPSME supports dynamic evolution. Redundant translations can be detected and removed from the integration to avoid bloat. To support legacy systems and various versions of the same protocol, a translation can be added to the integration, mapping the outdated version of the protocol to the new version.

IPSME forms an architecture, but not a framework: a pubsub can be found or implemented on most any system; and, the remaining conventions are agnostic to a specific implementation or virtual environment e.g., operating system, platform, programming language and/or application programming interfaces. A framework for IPSME communication can be written for each platform and/or programming language, but the usage of such a framework would be optional. The IPSME conventions are purposefully kept basic and should not require updating, unless a flaw in the soundness of IPSME is found. When a framework is used and it is updated, each system implementing the framework must also be updated; in an IPSME architecture, translations for outdated participants are added dynamically i.e., allowing participants to update individually.

### 6 Conclusion

In this article, IPSME is introduced as a solution for integrating highly disparate systems. Through the use of pubsub, IPSME decouples the production and consumption of messages, increasing scalability by decoupling dependencies between interacting participants. Rather than requiring two systems to speak the same protocol, IPSME achieves interoperability through translations that can be dynamically added to the system, avoiding the need for an agreed-upon protocol or framework. Mappings can handle several layers of interoperability and mappings in IPSME are specified in a manner so as to resolve schema heterogeneity through decomposition of the problem. IPSME supports dynamic evolution, meaning systems (e.g., legacy systems) do not need to be re-engineered as other systems are added, removed or updated in the integration. When participants using IPSME are updated, rather than expect all participants to conform, a translation can be added to the integration mapping the outdated version to the new version. When mappings require updating, rather than update each mapping, another mapping can be added reconciling the differences between mappings. The complexity of integrating systems using IPSME is linear, rather than exponential due to a network effect i.e., the amount of effort it takes to create a mapping is not reduced, but mappings can be reused, reducing the number of required mappings.

The intent is for IPSME to be broadly accepted for the integration of highly disparate systems, but there are environments which have specialized requirements e.g., the medical field requires security. The set of conventions defined here as IPSME is only the primary layer of a multilayered system; additional layers (e.g., to address service discovery and security) are left for subsequent publications.
References

[1] Edward N Barthell, Kevin Coonan, John Finnell, Dan Pollock, and Dennis Cochrane. 2004. Disparate Systems, Disparate Data: Integration, Interfaces, and Standards in Emergency Medicine Information Technology. Academic Emergency Medicine 11, 11 (2004), 1142–1148. https://doi.org/10.1119/1.aem.2004.08.008

[2] Len Bass, Paul Clements, and Rick Kazman. 2013. Software Architecture in Practice (3rd ed.). Addison-Wesley Professional.

[3] Chris Branton, Doris Carver, and Brygg Ullmer. 2011. Interoperability standards for pervasive games. In Proceedings of the 1st International Workshop on Games and Software Engineering. ACM, New York, NY, USA, 40–43. https://doi.org/10.1145/1984674.1984689

[4] Jeremy Brown, Jeff P Grossman, and Tom Knight. 2002. A lightweight idempotent messaging protocol for faulty networks. In Parallelism in Algorithms and Architectures. ACM, New York, NY, USA, 248–257. https://doi.org/10.1145/564870.564992

[5] Antonio Carzaniga, David S Rosenblum, and Alexander L Wolf. 2001. Design and Evaluation of a Wide-Area Event Notification Service. ACM Transactions on Computer Systems (TOCS) 19, 3 (2001), 332–383. https://doi.org/10.1145/380749.380767

[6] John David n. Dionisio, William G. Burns III, and Richard Gilbert. 2013. 3D Virtual Worlds and the Metaverse: Current Status and Future Possibilities. Comput. Surveys 45, 3 (2013), 34:1–34:38. https://doi.org/10.1145/2480741.2480751

[7] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. Comput. Surveys 35, 2 (6 2003), 114–151. https://doi.org/10.1145/857076.857078

[8] David Garlan and Mary Shaw. 1993. An Introduction to Software Architecture. In Advances in Software Engineering & Knowledge Engineering, Vincenzo Ambriola and Genoveffa Tortora (Eds.), Vol. 2. World Scientific, 1–39. https://doi.org/10.1145/978912790839_0001

[9] Didem Gürdür and Fredrik Asplund. 2018. A systematic review to merge distributed technologies. In Proceedings of the 6th International Workshop on Software Engineering for Systems-of-Systems (SESoS ’18, Montpellier, France). ACM, New York, NY, USA, 15–20. https://doi.org/10.1145/2489850.2489853

[10] John David n. Dionisio, William G. Burns III, and Richard Gilbert. 2013. 3D Virtual Worlds and the Metaverse: Current Status and Future Possibilities. Comput. Surveys 45, 3 (2013), 34:1–34:38. https://doi.org/10.1145/2480741.2480751

[11] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. Comput. Surveys 35, 2 (6 2003), 114–151. https://doi.org/10.1145/857076.857078

[12] David Garlan and Mary Shaw. 1993. An Introduction to Software Architecture. In Advances in Software Engineering & Knowledge Engineering, Vincenzo Ambriola and Genoveffa Tortora (Eds.), Vol. 2. World Scientific, 1–39. https://doi.org/10.1145/978912790839_0001

[13] Azad M. Madni and Michael Sievers. 2014. System of Systems Integration: Key Considerations and Challenges. Systems Engineering, The Journal of The International Council on 17, 3 (7 2014), 330–347. https://doi.org/10.1002/sys.21272

[14] Graham Morgan. 2009. Challenges of online game development: A review. Simulation & Gaming 40, 5 (7 2009), 688–710. https://doi.org/10.1177/1046878109340295

[15] Elisa Y. Nakagawa, Marcelo Gonçalves, Milena Guesati, Lucas B. R. Oliveira, and Flavio Oquendo. 2013. The State of the Art and Future Perspectives in Systems of Systems Software Architectures. In Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems (SESoS ’13, Montpellier, France). ACM, New York, NY, USA, 15–20. https://doi.org/10.1145/2489850.2489853

[16] Kim Nevelsteen, Theo Kanter, and Rahim Rahmani. 2016. Comparing Properties of Massively Multiplayer Online Worlds and the Internet of Things. In 2016 IEEE Symposium on Computers and Communication (DCC) Digital Entertainment, Networked Virtual Environments and Creative Technology (DENVECT), Messina, Italy, IEEE, 1–5. https://doi.org/10.1109/ISCC.2016.7543704

[17] Kim J. L. Nevelsteen. 2016. Distributed Technology-Sustained Pervasive Applications. Ph.D. Dissertation. Stockholm University, Department of Computer and System Sciences. http://urn.kb.se/resolve/urn:urn:nbn:se:uu:diva-129151

[18] Kim J. L. Nevelsteen. 2018. Virtual Worlds, Defined from a Technological Perspective, and Applied to Video Games, Mixed Reality and the Metaverse. Computer Animation & Virtual Worlds 29, 1 (5 2018), e1752. https://doi.org/10.1002/cav.1752

[19] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. 2019. Interoperability in Internet of Things: Taxonomies and Open Challenges. Mobile Networks and Applications 24 (06 2019), 796–809. https://doi.org/10.1007/s11036-018-1089-9

[20] Sven-Volker Behm, Lakshmi Goel, and Mattia Crespi. 2015. The Metaverse as Mediator between Technology, Trends, and the Digital Transformation of Society and Business. Journal For Virtual Worlds Research 8, 2 (2015). https://doi.org/10.4101/jvwr.v8i2.7149

[21] Scott A. Selberg and Mark A. Austin. 2008. 10.1.1 Toward an Evolutionary System of Systems Architecture. In Proceedings of the 6th International Workshop on Software Engineering for Systems-of-Systems (SESoS ’13, Montpellier, France). ACM, New York, NY, USA, 15–20. https://doi.org/10.1145/1103822.1103836

[22] Alon Halevy. 2005. Why Your Data Won’t Mix. Queue 3, 8 (11 2005), 50–58. https://doi.org/10.1145/1103822.1103836

[23] Chris Branton, Doris Carver, and Brygg Ullmer. 2011. Interoperability standards for pervasive games. In Proceedings of the 1st International Workshop on Games and Software Engineering. ACM, New York, NY, USA, 40–43. https://doi.org/10.1145/1984674.1984689

[24] Chenxi Wang, Antonio Carzaniga, David Evans, and Alexander L Wolf. 2002. Security issues and requirements for Internet-scale publish/subscribe systems. In System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on. IEEE, 3940–3947. https://doi.org/10.1109/HICSS.2002.994531

[25] Jim Wilenius. 2009. Bidding in Combinatorial Auctions. Number 656 in Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. Acta Universitatis Upsaliensis. http://urn.kb.se/resolve/urn:urn:nbn:se:uu:diva-109260