# IPSME- Idempotent Publish/Subscribe Messaging Environment [v1.4x-18-g036cd77-0423+2023]

Kim Nevelsteen
kim.nevelsteen@mitm.se
MiTM AB

Martin Wehlou
martin.wehlou@mitm.se
MiTM AB



**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, 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.

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

Nevelsteen, Kim and Martin Wehlou (2021). "IPSME- Idempotent Publish/Subscribe Messaging Environment." In: *International Workshop on Immersive Mixed and Virtual Environment Systems Proceedings (MMVE '21), Sept.28-Oct.1, 2021, Istanbul, Turkey*. ACM. DOI: https://doi.org/10.1145/3458307.3460966.

## 1 Introduction

The integration (interoperability) of highly disparate systems is an open topic of research in many domains, including Healthcare [Barthell et al. 2004; Landman et al. 2011], Computer Video Games [Morgan 2009], Internet of Things [Noura, Atiquzzaman, and Gaedke 2019; Nevelsteen, Kanter, and Rahmani 2016], Pervasive Applications [Nevelsteen 2016], and the Metaverse [Dionisio, Burns III, and Gilbert 2013]. And, by extension Cyber Physical Systems [Gürdür and Asplund 2018], which incorporates Internet of Things and the Metaverse [Rehm, Goel, and Crespi 2015]. The domain of this article can be generalized as the

integration of virtual environments of disparate systems, potentially forming a 'system of systems' [Madni and Sievers 2014].

A common approach for getting two highly disparate systems to be interoperable and communicate, is by having them speak the same protocol. 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 $n(n-1)/2$ [Noura, Atiquzzaman, and Gaedke 2019].

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' [Bass, Clements, and Kazman 2013] dependencies and incorporating 'interest management' [Nevelsteen, Kanter, and Rahmani 2016], 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' [Nevelsteen, Kanter, and Rahmani 2016; Nevelsteen 2018]. To provide interoperability and decouple dependencies, a publish/subscribe system (pubsub) is used in the set of conventions. Wang et al. [Wang et al. 2002] 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' [Noura, Atiquzzaman, and Gaedke 2019; Halevy 2005], and in the set of conventions, mappings are specified in a manner so as to resolve schema heterogeneity through 'decomposition' [Selberg and Austin 2008] of the problem. And, the dividing of systems into regions provides the required 'scalability' [Bass, Clements, and Kazman 2013] for the system [Noura, Atiquzzaman, and Gaedke 2019; Nevelsteen, Kanter, and Rahmani 2016]. 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 [Noura, Atiquzzaman, and Gaedke 2019]), is by having them speak the same protocol, either: by having a pre-agreed upon protocol with possibly varying implementations on each system, or by employing a framework (*e.g.,* Software Development Kit), specifically implemented for each system, to ensure communication. Integration is particularly a problem when dealing with existing 'legacy' systems [Madni and Sievers 2014], which speak different protocols (or divergent versions), that are difficult to alter.

The problem of integrating systems arises when many of these protocols/frameworks come into existence. Agreeing on protocols takes time and doesn't necessarily satisfy all 'stakeholders' [Madni and Sievers 2014]. This delay can stifle fast paced innovation [Shapiro and Varian 1999]. Different protocols can be bridged by making a mapping between the several layers of interoperability. A key issue with mappings is that they also 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, and the process is labor-intensive and error-prone" [Halevy 2005]. Mappings must not only satisfy syntactical and semantic interoperability, but also the conceptual layers of interoperability [Noura, Atiquzzaman, and Gaedke 2019]. There is not necessarily a single correct mapping [Halevy 2005] and when protocols are updated, any existing mappings must also be updated, exaggerating the problem.

# 3 Related Work

When faced with the problem of integrating highly disparate systems, many [Landman et al. 2011; Morgan 2009; Branton, Carver, and Ullmer 2011; Barthell et al. 2004] argue that interoperability can be achieved through standardization, but standardization only has limited success [Halevy 2005]. Features are either lacking at the time of standardization or the standard can become bloated with features requested from various stakeholders. 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 and update a standard [Noura, Atiquzzaman, and Gaedke 2019], *e.g.,* the proposed standard being outdated before it is standardized.

In Internet of Things, where there is currently active research in interoperability, gateways employing 'mediators' have been developed between devices to "bridge between different specifications, data, standards, and middleware's etc." [Noura, Atiquzzaman, and Gaedke 2019] These gateways can be expandable via plugins to upgrade interoperability with more systems, but "this approach has [a] limitation on scalability" [Noura, Atiquzzaman, and Gaedke 2019]. Open challenges for interoperability include scalability and that stakeholders should not have to "adapt to major changes in their system; the solution should not be dependent on their system" [Noura, Atiquzzaman, and Gaedke 2019].

# 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' [Bass, Clements, and Kazman 2013] 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 resends messages there. Communication between reflectors 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 message 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' [Brown, Grossman, and Knight 2002], 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. Idempotent 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 broadcast using pubsub, messages are the 'implicit invocation' [Garlan and Shaw 1993] 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 [Eugster et al. 2003], 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 [Eugster et al. 2003]. By not specifying message content IPSME avoids having a predetermined 'expressiveness' [Carzaniga, Rosenblum, and Wolf 2001], 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 [Uzunov 2016] 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 the decomposition of the problem. The translators can collectively divide the problem vertically, horizontally [Uzunov 2016] 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.

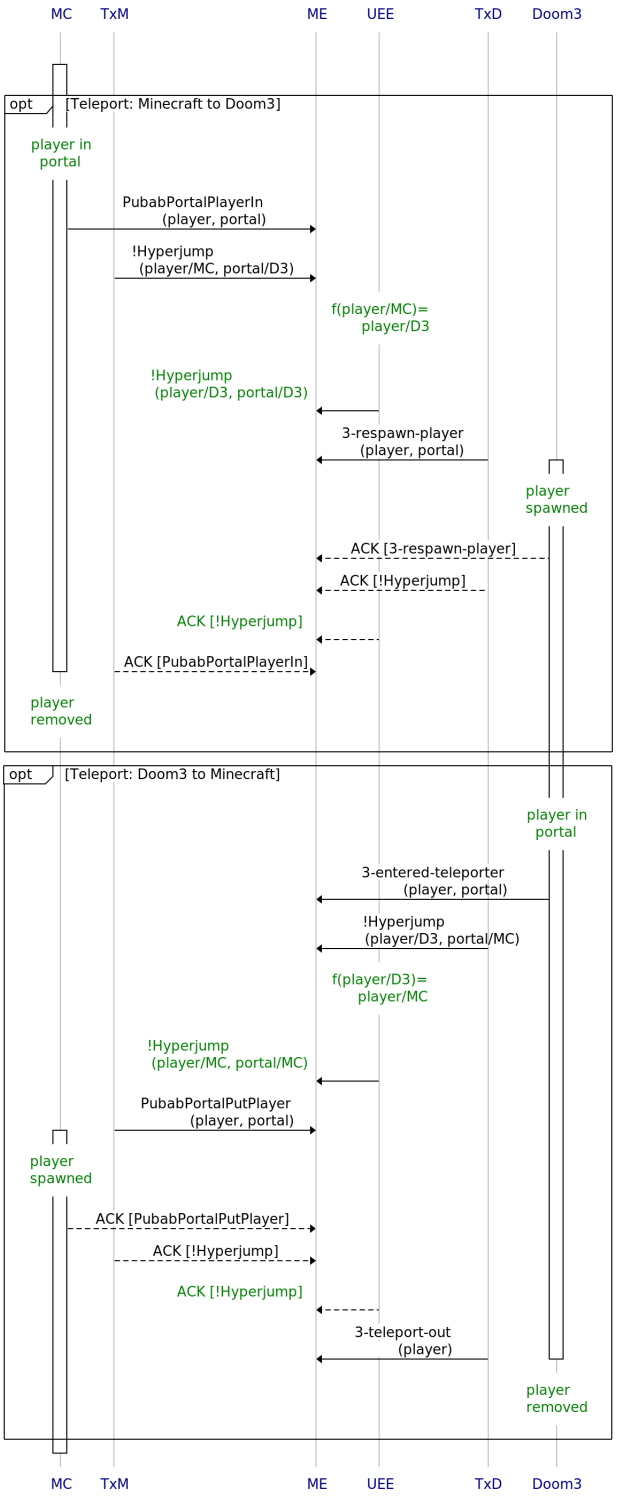
Participant communication is not constrained to be via a ME. Participants can negotiate to communicate directly allowing for 'explicit invocation' [Garlan and Shaw 1993]. It is the responsibility of the participants to negotiate such a communication, and beyond the IPSME specification.

## 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 [Carzaniga, Rosenblum, and Wolf 2001] or those related to quality of service [Eugster et al. 2003], 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 [Carzaniga, Rosenblum, and Wolf 2001].

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' [Uzunov 2016] 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 that 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.




Sequence Diagram 1: Minecraft/Doom integration: messaging for teleporting between Minecraft & Doom3, and back

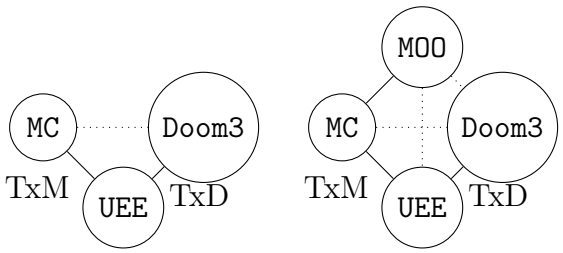
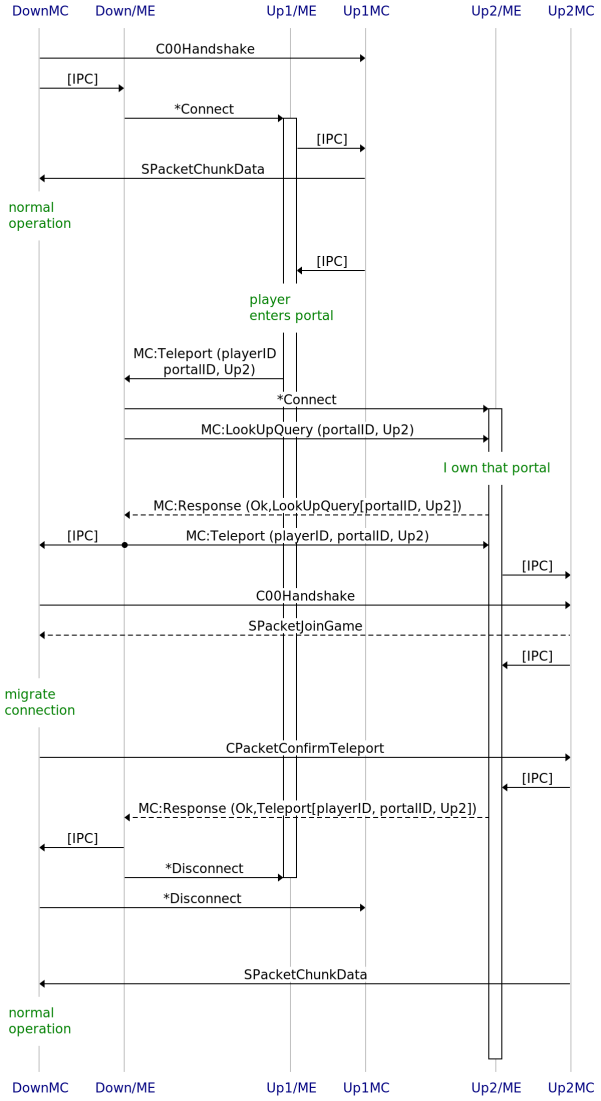



Figure 2: Translation graphs for three and four participants, respectively (solid lines are implemented translation; dotted lines are gained through a network effect).




Sequence Diagram 2: Minecraft-Metaverse via MiM: an illustration of using direct communication together with IPSME

# 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 prototype*[4] was a precursor to the three use cases, integrating the video games Minecraft and Doom3, and the virtual world of LambdaMOO. The proof-of-concept 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 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 Doom), a fully connected translation graph is required; see Figure 2 (left). The two translations provided by TxM and TxD were implemented, but a third translation from MC to Doom3 is 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 proof-of-concept, 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

[1] *Minecraft/Doom integration forming a Metaverse*: https://youtu.be/knKZd15rhJE
[2] *Medical Resource Scheduling*: https://youtu.be/m_b1PotByx0
[3] *Insane Minecraft Teleportation, in VR, via MiM*: https://youtu.be/ORWto-Oo1W4
[4] *Metaverse prototype verbose demo*: https://youtu.be/etKJMUyPn-8

between MOO and MC; see Figure 2 (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 primary focus of the Medical Resource Scheduling use case was the ability of various Care Planner components to dynamically find and negotiate with various medical resource providers (*e.g.,* operating rooms). Although not in the use case, the system was designed such that each of the various components could have been on different platform. The entire system is design to be scalable and allow various stakeholders to negotiate without the need for a central authority. A large degree of fault tolerance was built into the use case, by detecting failed participants and executing multiple copies of the same participant to achieve redundancy.

## 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[5] 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 upstream components. Up1 determines that 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; a `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

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

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 simulated player teleport is complete. Up2 broadcasts out a corresponding `MC:Response`, which signals Down to disconnect from Up1, which was hosting the source end of the portal, and subsequently DownMC from Up1MC.

### 5.4 Discussion

Rather than having to wait for features to be added to a standard, IPSME allows a translating participant to be added dynamically to the system, mediating communication with a mapping. Each translating participant only provides the required mapping, avoiding the problem of bloat. If many redundant translating participants are present, those which are not widely used can be detected, removed and replaced with a more minimal set of translators. Since translating participants can be added dynamically, it is possible to avoid the delay of waiting for an agreed upon standard. The same dynamism ensures openness to unforeseeable future requirements.

IPSME can largely be considered a generalization (in software) of Internet of Things gateways as mediators, with the dynamic insertion of translating participants being similar to gateway plugins. The advantage of IPSME being that it offers a network effect by which translations become transitively applicable, removing the requirement that all mappings must be one-to-one. IPSME provides for scalability when the gateway approach was noted as being limited. Also, because of the dynamism of IPSME, stakeholders are not forced to adapt their systems to major changes in protocols; which solves that open challenge in the domain of Internet of Things.

## 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 removing dependencies (*i.e.,* time, space or synchronization) 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 pre-agreed upon protocols or frameworks, and also avoiding any delay that is incurred through coming to agreement. IPSME is minimally invasive and can be used to integrate legacy systems or even the most difficult interoperability cases [Madni and Sievers 2014].

Translations in IPSME can handle several layers of interoperability. The amount of effort it takes to create a mapping might not be reduced, but through the network effect granted by IPSME, translations can potentially be reused, reducing the exponential complexity of fully integrating many systems to linear. And, when protocols are updated, rather than update each mapping, with IPSME a translation can be added that maps from the original protocol to the updated one and the integrated systems continue.

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


Barthell, Edward N et al. (2004). "Disparate Systems, Disparate Data: Integration, Interfaces, and Standards in Emergency Medicine Information Technology." In: *Academic Emergency Medicine* 11.11, pp. 1142–1148. DOI: 10.1197/j.aem.2004.08.008.

Bass, Len, Paul Clements, and Rick Kazman (2013). *Software Architecture in Practice*. 3rd. Addison-Wesley Professional. ISBN: 978-0-321-81573-6.

Branton, Chris, Doris Carver, and Brygg Ullmer (2011). "Interoperability standards for pervasive games." In: *Proceedings of the 1st International Workshop on Games and Software Engineering*. New York, NY, USA: ACM, pp. 40–43. DOI: 10.1145/1984674.1984689.

Brown, Jeremy, Jeff P Grossman, and Tom Knight (2002). "A lightweight idempotent messaging protocol for faulty networks." In: *Parallelism in Algorithms and Architectures*. New York, NY, USA: ACM, pp. 248–257. DOI: 10.1145/564870.564912.

Carzaniga, Antonio, David S Rosenblum, and Alexander L Wolf (2001). "Design and Evaluation of a Wide-Area Event Notification Service." In: *ACM Transactions on Computer Systems (TOCS)* 19.3, pp. 332–383. DOI: 10.1145/380749.380767.

Dionisio, John David n., William G. Burns III, and Richard Gilbert (2013). "3D Virtual Worlds and the Metaverse: Current Status and Future Possibilities." In: *ACM Computing Surveys* 45.3, 34:1–34:38. DOI: 10.1145/2480741.2480751.

Eugster, Patrick Th. et al. (June 2003). "The Many Faces of Publish/Subscribe." In: *ACM Computing Surveys (CSUR)* 35.2, pp. 114–131. DOI: 10.1145/857076.857078.

Garlan, David and Mary Shaw (Dec. 1993). "An Introduction to Software Architecture." In: *Advances in Software Engineering & Knowledge Engineering*. Ed. by Vincenzo Ambriola and Genoveffa Tortora. Vol. 2. World Scientific, pp. 1–39. DOI: 10.1142/9789812798039_0001.

Gürdür, Didem and Fredrik Asplund (Mar. 2018). "A systematic review to merge discourses: Interoperability, integration and cyber-physical systems." In: *Journal of Industrial Information Integration* 9, pp. 14–23. DOI: 10.1016/j.jii.2017.12.001.

Halevy, Alon (Nov. 2005). "Why Your Data Won't Mix." In: *Queue* 3.8, pp. 50–58. DOI: 10.1145/1103822.1103836.

Kshemkalyani, Ajay D. and Mukesh Singhal (2008). *Distributed Computing Principles, Algorithms, and Systems*. Cambridge University Press.

Landman, Adam B. et al. (2011). "An Open, Interoperable, and Scalable Prehospital Information Technology Network Architecture." In: *Prehospital Emergency Care* 15.2, pp. 149–157. DOI: 10.3109/10903127.2010.534235.

Madni, Azad M. and Michael Sievers (July 2014). "System of Systems Integration: Key Considerations and Challenges." In: *Systems Engineering, The Journal of The International Council on* 17.3, pp. 330–347. DOI: 10.1002/sys.21272.

Morgan, Graham (July 2009). "Challenges of online game development: A review." In: *Simulation & Gaming* 40.5, pp. 688–710. DOI: 10.1177/1046878109340295.

Nakagawa, Elisa Y. et al. (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. New York, NY, USA: ACM, pp. 13–20. DOI: 10.1145/2489850.2489853.

Nevelsteen, Kim, Theo Kanter, and Rahim Rahmani (June 2016). "Comparing Properties of Massively Multiplayer Online Worlds and the Internet of Things." In: *2016 IEEE Symposium on Computers and Communication (ISCC)*. Digital Entertainment, Networked Virtual Environments and Creative Technology (DENVECT), Messina, Italy. IEEE, pp. 1–5. DOI: 10.1109/ISCC.2016.7543704.

Nevelsteen, Kim J. L. (May 2016). "Distributed Technology-Sustained Pervasive Applications." PhD thesis. Stockholm University, Department of Computer and System Sciences. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:su:diva-129151.

Nevelsteen, Kim J. L. (May 2018). "Virtual World, Defined from a Technological Perspective, and Applied to Video Games, Mixed Reality and the Metaverse." In: *Computer Animation & Virtual Worlds* 29.1, e1752. ISSN: 1546-427X. DOI: 10.1002/cav.1752.

Noura, Mahda, Mohammed Atiquzzaman, and Martin Gaedke (June 2019). "Interoperability in Internet of Things: Taxonomies and Open Challenges." In: *Mobile Networks and Applications* 24, pp. 796–809. DOI: 10.1007/s11036-018-1089-9.

Rehm, Sven-Volker, Lakshmi Goel, and Mattia Crespi (2015). "The Metaverse as Mediator between Technology, Trends, and the Digital Transformation of Society and Business." In: *Journal For Virtual Worlds Research* 8.2. DOI: 10.4101/jvwr.v8i2.7149.

Selberg, Scott A. and Mark A. Austin (2008). "10.1.1 Toward an Evolutionary System of Systems Architecture." In: *INCOSE International Symposium* 18.1, pp. 1065–1078. DOI: 10.1002/j.2334-5837.2008.tb00863.x.

Shapiro, Carl and Hal R. Varian (1999). *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business School Press. ISBN: 978-0-87584-863-1.

Uzunov, Anton V (2016). "A survey of security solutions for distributed publish/subscribe systems." In: *Computers & Security* 61, pp. 94–129. DOI: 10.1016/j.cose.2016.04.008.

Wang, Chenxi et al. (Jan. 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, pp. 3940–3947. DOI: 10.1109/HICSS.2002.994531.

Wilenius, Jim (2009). "Bidding in Combinatorial Auctions." PhD thesis. ISBN: 978-91-554-7554-3. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-102960.