OpenTwins：用于设计、开发和集成高效的3D-IoT-AI驱动数字孪生的开源框架

Julia Robles、Cristian Martín、Manuel Díaz

ITIS Software，马拉加大学，*Arquitecto Francisco Peñalosa 18*号，*29071*，西班牙马拉加

arXiv:2301.05560v1 [cs.SE] 12 Jan 2023

摘要

尽管数字孪生近来已成为可靠资产表征的一种明确替代方案，但现有的大多数数字孪生开发解决方案和工具都是针对特定环境量身打造的。此外，要实现可靠的数字孪生，往往需要协调机器学习、物联网和三维可视化等技术与范式，而这些技术与范式很少能够无缝协同。本文介绍了一种通用框架，可结合上述领域中的若干方面来开发高效的数字孪生。借助这一开放框架，可以轻松开发和编排数字孪生，并集成三维交互式可视化、物联网数据流以及实时机器学习预测。为验证该框架的可行性，我们开发了一个石化工业4.0用例。

关键词 *s:* 数字孪生框架、Kafka-ML、工业 4.0

1. 引言

工业4.0彻底改变了制造业、检测和汽车等众多领域的生产方式和技术能力。在当今时代，我们周围的资产（例如车联万物-V2X）日益互联，彼此之间以及与环境共享信息，以优化性能、防止危险情况并提高安全性。我们可以重点关注伴随工业革命而兴起的两种主要范式：物联网（IoT） [1], 使人们能够实时监测并控制多种物理现象；人工智能（AI）和机器学习（ML）[2], 则为通过数据驱动的方法对系统和流程的行为进行建模（某些情况下，这些行为原本未知）铺平了道路。因此，随着周围信息来源不断增加，（物联网）分析技术也不断发展，以提升对环境的认知（人工智能/机器学习）。这种交汇催生了当今的深度学习技术：它利用大量（物联网）数据，在单一流程中对特征提取、分类和/或复杂模式检测进行建模，值得关注。

对资产实时行为及其可能未来状态进行持续分析和预测的需求，促成了所谓数字孪生的出现 [3]。数字孪生可以定义为对物理资产进行数字化、准确且可信的表征 [4]，通过持续监测、预测和优化来支持决策。以列车车轮轴承为例，其维护和生产成本往往较高 [5]。能够预测轴承何时需要维修和/或更换，以及其预计使用寿命的数字孪生，可以大幅优化运营成本，并让铁路公司更好地进行规划。尽管数字孪生与

电子邮箱地址： cristian@uma.es(Cristian Martín), mdiaz@uma.es(Manuel Díaz)

提交至 *arXiv* 的预印本 *2023*年*1*月*16*日

信息物理系统有相似之处，例如将资产整合到数字世界中，但数字孪生更进一步 [6], ，能够提供资产的准确副本，使其行为与物理世界中的实际行为一致。数字孪生利用推动工业4.0发展的范式来监测环境（物联网），并寻求对物理资产进行持续优化和预测（人工智能/机器学习）。在许多流程中，除了数据驱动模型之外，基于物理的模型还通过数学方程准确表征资产行为。然而，这些模型往往难以获得，因此会辅以人工智能/机器学习等数据驱动模型。因此，数字孪生是将多种技术、模型和范式整合到统一界面中的组合，用于资产控制和监测。在此背景下，资产的可视化（最好采用三维形式）也发挥着关键作用，因为构建可靠数字孪生的主要功能之一，是仿真资产在未知和极端情况下的表现，以评估其行为。

数字孪生所呈现的技术与范式编排，需要借助框架来持续开发和集成。过去几年中，涌现出大量支持数字孪生开发的框架。然而，在绝大多数情况下，数字孪生框架都局限于其所应用的领域 [7]。一个值得关注的开源示例是 Eclipse Ditto[1], ，它是多领域数字孪生最广泛使用的解决方案之一。Eclipse Ditto 支持对资产通信进行虚拟抽象，并提供细粒度的访问控制管理，但要实现有效的数字孪生，仍需付出大量额外的集成工作。例如，还需要与人工智能/机器学习技术无缝协调，集成三维渲染引擎、传感器故障检测以及生态系统的集成可视化等。此外，数字孪生通常面向单个资产，但这些资产可能属于一个全局系统（例如轴承和列车），此时可能需要对数字孪生进行编排，或进行数字孪生组合

[1]https://www.eclipse.org/ditto/

。本文介绍了一种基于 Eclipse Ditto 的开源框架，用于持续开发和集成有效的数字孪生。该框架针对数字孪生领域所讨论的主要技术和范式（即物联网、人工智能/机器学习、三维可视化）提供集成解决方案，并为其监控、管理和持续优化提供统一接口。该框架基于可扩展平台设计，能够实现容错和高可用性，并为未来扩展创造条件。本文的主要贡献如下:

1. 基于 Eclipse Ditto 构建集成生态系统，用于抽象和持续监控物理资产。2. 将人工智能/机器学习技术与数据流无缝编排，实现持续优化和预测，例如传感器故障检测。3. 集成三维渲染引擎，为三维数字孪生的设计与可视化提供支持。4. 结合上述目标，本文的总体贡献是提供一个集成的开源框架和界面，用于设计、开发和持续集成高效的数字孪生组合。5. 最后，本文通过石化行业的一个制造业用例验证了该框架。

本文其余部分安排如下：第 2 节讨论相关工作。第 3 节介绍数字孪生框架的架构及其组件，第 4 节则讨论实现细节。第 6 节对该框架进行评估。最后，第 7 节总结本文并展望未来工作。

2. 相关工作

近年来，关于数字孪生的研究产生了重大影响。然而，一些研究探讨了设计和架构方面 [8, 9], ，另一些则只关注特定领域。

[10, 11]。因此，应用开发者可能难以在工业4.0领域找到数字孪生框架，尤其是用于开发高效 3D-IoT-AI 驱动型数字孪生的开源工具。以下总结了该领域一些最具代表性的研究。

Kamath等人 [12] 旨在通过提供基于开源组件开发数字孪生的架构，弥合学术界与工业界之间的差距。我们的架构与他们的研究有一些相似之处，因为本研究也采用了 Kafka、InfluxDB 和 Eclipse Ditto 等开源技术。不过，我们的架构更进一步，不仅支持将数字孪生与实时数据监测相集成，还支持集成机器学习技术（Kafka-ML）和三维渲染。最后，在我们的架构中，除了使用开源组件外，整个数字孪生平台及相关组件也都可在 GitHub[2]上获取。

Karan等人 [13]也考虑采用 Eclipse Ditto 和 OpenPLC 等开放工具来实现数字孪生框架。他们编排了一个仿真模型，用于对便携式温控舱的计算流体力学（CFD）进行建模。尽管集成仿真模型有望将现实世界监测数据资产与物理模型结合起来，但该框架缺乏灵活性，且仅适用于 CFD 系统。

[14]介绍了一种模块化数字孪生框架。该框架可对资产进行监控、评估和三维可视化，并已在真实制造场景中得到验证。可通过 LabVIEW 定义一些关键绩效指标，用于测量和告警生成。该架构的主要局限在于，所有元素都连接到同一个 OpenPLC服务器。正如文章中所评估的那样，该架构面临可扩展性挑战；一种可能的解决方案是，为每个需要监控和建模的流程/机器复制框架结构。而我们的架构则完全基于容错容器，并通过 Kubernetes 实时管理这些容器。

[7]围绕 FIWARE生态系统构建了一套数字孪生开发指南。FIWARE 是欧盟委员会推动的知名生态系统，旨在支持多个领域的新一代应用开发。FIWARE 提供了用于服务实现的开源组件，包括数据摄取、数据分析、身份验证和数据存储等。该解决方案介绍了如何在 FIWARE 中定义数字孪生，但并未像我们的工作那样开发并验证适用于数字孪生的框架。

Khan等人[15] 提出了一种螺旋式数字孪生框架，旨在通过区块链安全、可靠地管理数字孪生数据。该框架旨在持续、一致地同步数字孪生与现实资产。为解决数字孪生与区块链技术融合的问题，研究者提出了一种新的区块链解决方案 twinchain。螺旋式数字孪生框架着重于提高数字孪生信息的可靠性，并未像我们的工作那样提供用于开发有效数字孪生体的通用框架。尽管组件之间的通信已加密，我们未来也将研究采用区块链来提高系统的可靠性。

3. 用于设计和开发3D物联网人工智能驱动的数字孪生体的开源架构

该架构的核心支柱是广为人知的数字孪生框架 Eclipse Ditto。Eclipse Ditto 提供用于对孪生体建模的实体、孪生体状态及其事件的存储、访问控制系统，以及支持多种连接类型的功能，从而能够与孪生体及其他后端进行交互。除来自 HTTP API 的消息外，发送至 Eclipse Ditto 的所有消息都必须遵循 Ditto 协议[3] 格式，或者必须在连接中配置负载映射。我们围绕 Eclipse Ditto 集成了架构所需的所有元素，既

[2]https://github.com/ertis-research/digital-twins-platform

[3]https://www.eclipse.org/ditto/protocol-overview.html

弥补其在数字孪生本质需求方面的不足，也扩展其功能。所有这些元素都是开源工具，尽管其中大多数是外部项目，但有些服务是专门开发的，用于添加特定功能或支持不兼容元素之间的连接。

其成果如图 1 所示，是一种微服务架构，满足数字孪生平台的基本需求，例如设备连接、实时数据存储和数据可视化；但其真正的亮点在于其他功能，例如通过机器学习预测实时数据流，以及借助交互式三维表示展示孪生体的状态。

该微服务架构具有很强的模块化、可复用性和可扩展性。各组件相互独立，分别负责特定功能。因此，可以轻松替换组件而不影响系统的其余部分，在其他项目中复用部分服务，并以较低的难度扩展系统。关于服务之间的通信，大多数服务都提供 API（应用程序编程接口）以便于交互；不过，在接收和发送实时数据时，也会考虑采用 AMQP 和 Apache Kafka 等其他发布-订阅协议。

为便于管理架构中的所有服务，服务均已打包到容器中，并由容器编排器进行协调。在本例中，Docker 用作容器技术，Kubernetes 用作容器编排器。对于每个外部项目，均选用了其对应的 Docker 容器版本，并将所有开发的服务容器化。此外，将每项服务封装在隔离环境中，可确保其在大多数系统中的可移植性和正确运行。

在 GitHub 上，[4] 可以找到系统说明、架构安装与连接手册、使用所需的文档，以及指向所有服务和插件的链接，这些插件除了包含生成的代码外，还有各自的安装手册和使用文档。

架构的讨论按照主要里程碑展开，分为作为数字孪生平台的基本功能、基于机器学习的数据预测，以及孪生体状态的三维表示。

3.1. 核心功能

这一里程碑的目标是构建一个平台，以便为由传感器组成的任意元素定义数字孪生。为此，必须定义孪生体、获取并连接物联网信息、以实时序列的形式存储孪生体数据，并能够以用户友好的方式查询这些数据。

虽然次要一些，但纳入孪生类型的创建和管理也很方便。这简化了繁琐的任务：创建多个孪生体，它们虽然对应不同的物理设备，却具有完全相同的特征。例如，如果现实世界中有十个相同的传感器，我们只需定义一种类型，就能为它们创建十个孪生体。

架构中的蓝色组件（图1）代表了与此项基本功能相对应的架构部分。这部分主要由开源项目组成，但也包含两个为实现所需功能而开发的元素。

核心组件是 Eclipse Ditto，正如前文所述，它是一个用于构建数字孪生的开源框架。Eclipse Ditto 不提供用于获取设备发送信息的系统，因此将使用 Eclipse Hono[5] 来实现这一目的。Eclipse Hono 是一个平台，提供多个接口，可连接大量物联网设备，并将它们统一到一个 AMQP 1.0 端点，从而读取接收到的信息，并发送命令以触发任意设备上的操作。它可以通过常见的物联网协议（如 MQTT、AMQP、HTTP 和 CoAP）以及自定义适配器接收信息。该工具是与 Eclipse Ditto 配合使用的推荐工具，而且多亏了

[4] https://github.com/ertis-research/digital-twins-platform

[5] https://www.eclipse.org/hono/

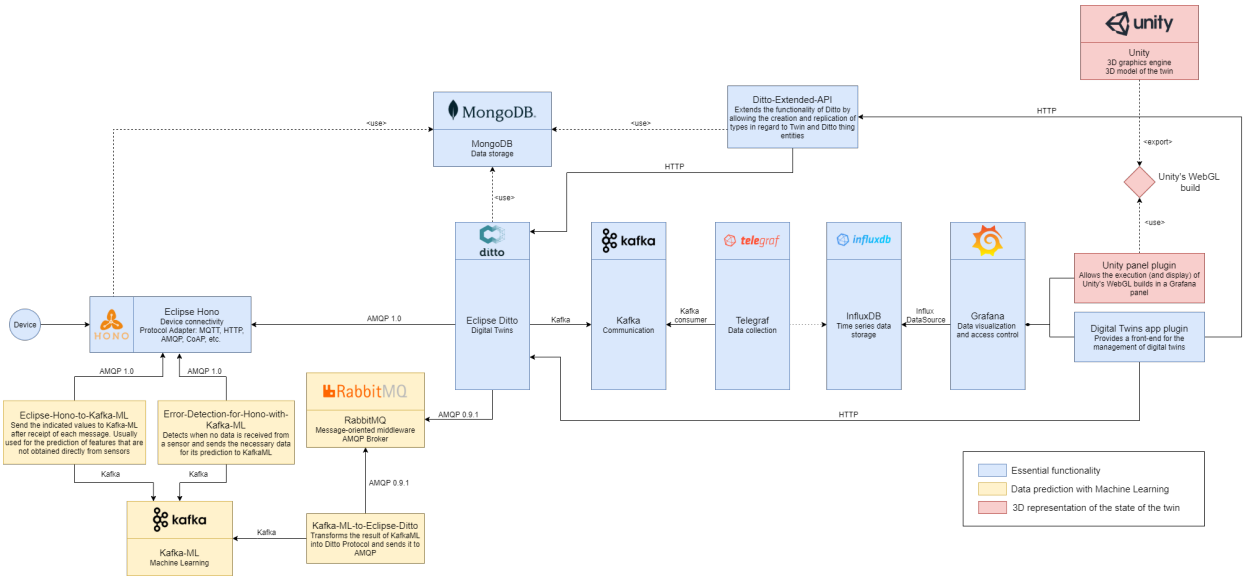


图 1：概览 3D-IoT-AI 驱动的数字孪生架构。蓝色表示基本功能，红色表示三维表示，黄色表示基于机器学习的数据预测。

Eclipse cloud2edge 软件包[6]，两者的集成非常便捷。

一方面，Eclipse Ditto Thing 实体可以采用不同方式使用。因此，在研究了各种选项后，例如考虑将一个 Ditto Thing 视为一个完整的孪生体，并将其中的每个传感器作为其特征，我们决定将一个 Ditto Thing 分配给单个实体或传感器，并在这些实体之间建立父子层级结构。这些功能已在名为*Ditto-Extended-API*的服务中实现。该服务可视为 Eclipse Ditto 之上的一层，并提供一个取代该技术原有 API 的接口。除了验证指定约束是否得到满足外，该服务还为孪生类型添加了所有相应功能，使其既可管理这些类型，也可根据这些类型创建孪生体。

另一方面，Eclipse Ditto 缺少的另一项特征是存储孪生状态在不同时间点的值。为了解决这一问题，我们选择了 InfluxDB[7] 作为时间序列数据库。这是一款知名数据库，拥有庞大的社区，非常适合处理传感器数据。为了收集数据，我们考虑了 Telegraf[8]，是一款由插件驱动的服务器，支持大量数据源，并可用于配置将数据摄取到数据库中的方式。由于 Telegraf 无法直接从 Eclipse Ditto 收集数据，因为这两种技术都没有实现其所支持协议的消息代理，因此我们需要一个能够让它们连接起来的中间组件。Apache Kafka[9]，是最知名的实时数据流处理平台之一，是一个很好的替代方案：Telegraf 提供 Kafka 消费者[10] 插件，而 Eclipse Ditto 则支持将事件发布到 Kafka 主题。

最终，选择 Grafana[11] 作为前端，即面向最终用户的用户界面。这项技术支持对最常用数据库中的指标进行可视化，包括 InfluxDB。它允许使用所选数据源定义的语言进行查询，并将结果呈现在不同类型的交互式面板中。这些面板

[6]https://www.eclipse.org/packages/packages/cloud2edge/[7]https://www.influxdata.com/products/influxdb-overview/

[8]https://www.influxdata.com/time-series-platform/telegraf/[9]https://kafka.apache.org/[10]https://www.influxdata.com/integration/kafka-telegraf-integration/[11]https://grafana.com/

是仪表板的一部分，用户可以根据自己的喜好对其进行修改。它还通过角色提供访问控制系统。它的另一大优势，也是该项目最看重的优势之一，是支持创建个性化面板，并通过插件添加任何类型的功能，同时提供所需的库和文档。

Grafana 的插件应用[12]，Digital Twins 已经开发完成。该插件为管理孪生体和类型提供了图形界面，并将其集成到前端已有的应用中，用于查询孪生体的状态，从而统一平台的所有功能。它的功能基本上是以用户友好的方式调用 Ditto-Extended-API 服务。

3.2. 使用机器学习进行数据预测

这一里程碑旨在实现平台与机器学习算法的集成。这有助于数字孪生预测其下一状态或故障情况。例如，当传感器未返回真实数据时，无论是因为传感器已关闭，还是出现了某种故障，都可以预测它应返回的值。

架构中负责实现这一目标的部分对应于图 1 中的黄色组件。在这部分中，主要组件是 Kafka-ML [16]，，它将负责机器学习生命周期管理，并对该架构进行补充。此外，为了将其与 Eclipse Hono 和 Eclipse Ditto 集成并实现所需功能，开发了三个专用服务：*Eclipse-Hono-to-Kafka-ML*、*Error-Detection-for-Hono-with-Kafka-ML* 和 *Kafka-ML-to-Eclipse-Ditto*。

Kafka-ML 是由我们团队开发的开源框架 [13]，可通过连续数据流管理生产环境中机器学习/人工智能应用的整个生命周期。与处理数据集或静态文件的传统框架不同，Kafka-ML 支持使用连续数据流进行训练和推理，使用户能够精细控制输入数据在 TensorFlow 和 PyTorch 等流行机器学习框架中的摄取过程。Kafka-ML 目前支持 TensorFlow 和 PyTorch 等流行机器学习框架，并通过易用的 Web 用户界面支持机器学习模型的管理和部署，涵盖从模型定义到最终部署以进行推理的整个过程。

机器学习领域中聚焦于数字孪生的一项主要用途，是预测孪生体的未来状态；如果要对孪生体所代表的对象进行控制或改进，这种预测可能极具实用价值。它还旨在预测孪生体的某些特征或值，而这些特征或值无法直接测量或获取，或者无法以足够准确的方式获取。如上所述，这些模型将部署在 Kafka-ML 中。为将它们与 Eclipse Hono 连接，我们开发了一项服务，名为 *Eclipse-Hono-to-Kafka-ML*[14] 。该服务持续从与租户相对应的 Eclipse Hono 端点读取数据；租户是一种实体，可将设备逻辑划分为不同的组，其中包含数据必须发送至已部署的 Kafka-ML 模型的设备。当其中一台设备发来消息时，该服务会处理数据，使其符合所需格式，并自动将其发送到相应的 Kafka-ML 输入主题，模型将在该主题中处理数据并生成预测结果。随后，数字孪生会根据预测结果进行更新。值得注意的是，在预测孪生体未来状态时，不会更新主数字孪生，而是在 Eclipse Ditto 中创建其副本，并在副本中包含预测状态。该副本会被视为同一个孪生体，只是时间上向前推进了一段时间。

这部分的另一个优势是能够检测传感器何时停止发送数据，并据此采取行动。为此，我们创建了一项新服务*Error-Detection-for-Hono-with-Kafka-ML* [15]，它主要读取指定 Hono 租户接收到的信息，并检查其中包含的设备是否

[12]https://github.com/ertis-research/digital-twins-plugin-for-grafana/[13]https://github.com/ertis-research/kafka-ml/

[14]https://github.com/ertis-research/eclipse-hono-to-kafka- ml[15] https://github.com/ertis-research/error-detection-for-eclipse-hono-with-kafka-ml/

按照各自的周期发送数据。如果某个设备到期后仍未发送数据，该服务就会将从该传感器接收到的最后几个值以及日期等其他必要数据，发送给一个使用历史数据训练的 Kafka-ML 模型。Kafka-ML 将预测系统的下一个状态，以避免因传感器故障而导致服务中断，直至传感器恢复可用。

另一方面，Kafka-ML 生成的数据必须由 Eclipse Ditto 消费。最初的想法是利用 Ditto 提供给 Kafka-ML 的负载映射功能，为每个模型发送预测结果的 Kafka-ML 输出主题创建一个源连接，并对接收到的信息进行映射，使其能够被 Eclipse Ditto 支持。但这行不通，因为在开发该平台时，Eclipse Ditto 尚未实现与 Apache Kafka 的连接，无法将其用作数据源。因此，决定构建一个中间服务 [16] ，由它从 Kafka 读取信息，将其映射为 Eclipse Ditto 协议，并发布到 Eclipse Ditto 可以连接的消息代理——本例中为使用 AMQP 0.9.1 的 RabbitMQ[17], 。

3.3. 孪生体状态的三维表示

数字孪生平台的一个重要方面是数据的表示。在最具代表性的工业平台中，通常可以找到孪生体的三维表示，这使得用户能够更加轻松地理解其信息以及当前所查询的组件。在本里程碑中，主要目标就是实现孪生体的这种三维表示。

为三维可视化设计的架构对应于图 1 中红色的组件，基本上由一个 Grafana 面板插件构成，该插件可显示使用 Unity 开发的三维模型[18] ，此外还支持双向交互。

Unity 是一款主要用于电子游戏开发的软件，不过也可以应用于其他场景。它是最受欢迎、拥有最大社区的图形引擎之一。虽然它不是开源工具，但个人使用或用于预算有限的项目时可以免费使用。这项技术可以通过脚本为三维对象设定特定行为，并使其与用户及环境中的其他元素进行交互。它支持导入多种格式的三维模型，包括 Blender。Blender 是一款开源三维创作套件，同样拥有庞大的社区，可用于创建或修改三维对象。Unity 还支持将项目构建为多种格式，其中包括一种专用于网页渲染的格式 Unity WebGL。以这种格式进行编译，对于架构的这一部分也至关重要。

以这种格式构建孪生体的三维表示后，下一步是在 Grafana 中运行它。Grafana 是我们选定的平台前端工具。为此，我们开发了一个面板插件。这类插件支持创建自定义面板，并可将其包含在任意仪表板中。它通常会表示或使用数据序列，而这些数据序列取决于用户在面板配置中指定的查询和数据源。此外，还可以对其进行实现，使其在配置中向用户提供一系列自定义选项，从而能够轻松修改其行为或可视化效果。

除了渲染 Unity WebGL 构建外，插件面板还必须能够与 Grafana 和用户进行双向交互。换言之，用户与 Grafana 的交互会反映在三维模型中；反过来，用户与该表示形式的交互也可能影响仪表板其他面板所显示的信息。此插件称为 Unity 面板插件。[19]

[16]https://github.com/ertis-research/kafka-ml-to-eclipse- ditto/[17] https://www.rabbitmq.com/[18]https://unity.com/

[19]https://github.com/ertis-research/unity-plugin-for-grafana/

4. 开发与实施

接下来将详细说明平台的完整开发过程，包括不同技术的连接方式、它们之间的关系，以及上述已开发服务的设计。

4.1. 连接基础架构

为了将不同技术部署在由 Kubernetes 管理的 Docker 容器中，我们选择使用 Helm[20]。Helm 是 Kubernetes 的包管理工具，通过一条命令即可组合并安装各项技术推荐的部署配置，从而省去手动创建 Kubernetes 对象的步骤。Helm 的软件包称为 Chart，只需修改 YAML 值文件中的必要参数即可轻松自定义。

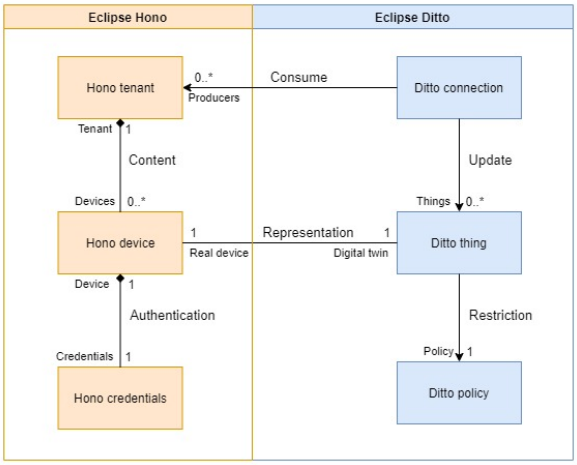


图 2：Eclipse Ditto 与 Eclipse Hono 之间的关系

Eclipse Ditto 和 Eclipse Hono 是通过 Eclipsecloud2edge 软件包安装的。该软件包会安装这两项技术，使它们能够相互通信，并通过一个示例设备在它们之间建立连接。为了确保安装正常运行，必须禁用 Eclipse Hono 中的 Grafana 和 Prometheus 选项，还需要创建一个 Kubernetes 持久卷，以对应同一工具的设备注册服务，从而实现数据持久化。

仅凭这一点，我们就可以创建数字孪生的模型，并利用其所代表的传感器提供的信息对其进行更新。此时，强调这两种工具之间的关系十分重要，如图 2 所示。在 Eclipse Hono 中，只需为设备分配一个标识符，就能在租户内定义设备。租户只是设备的集合。实际设备若要向该实体发送数据，就必须使用此标识符以及预先添加的凭据。我们可以通过查询 AMQP 端点中与设备所属租户相对应的地址，来访问该设备的数据。另一方面，在 Eclipse Ditto 中，我们可以创建 Ditto Thing 实体，这些实体将受到策略的限制，其标识符（thingId）由命名空间和一个标识符组成，并遵循 namespace:id 格式。为了让 Ditto Thing 接收更新消息，必须按照 cloud2edge 软件包中的说明建立 Ditto 连接与 Eclipse Hono AMQP 端点之间的连接，并将相应 Ditto Thing 的 thingId 设置为 Eclipse Hono 设备的标识符。

对于传感器而言，物联网设备预计会以 Ditto Protocol 格式向 Eclipse Hono 发送消息；如果无法做到这一点，也始终可以在连接到 Eclipse Ditto 时建立映射。消息应发送至负责所选连接协议的 Eclipse Hono 服务，并且连接应使用特定凭据进行身份验证，这些凭据会指明消息所对应的设备。

按照架构，下一步是将孪生体的状态发布到 Apache Kafka 主题中。在本例中，我们重复利用了已有资源来安装 Apache Kafka 和 CMAK[21]，（用于管理 Kafka 集群的图形界面）。准备就绪后，我们创建了一个主题，用于接收来自 Eclipse Ditto 的信息。Eclipse Ditto 支持创建目标连接，将孪生体信息发送到外部系统，包括 Apache Kafka 代理。

[20] https://helm.sh/

[21] https://github.com/yahoo/CMAK

定义连接时，需要指定要订阅哪些数据主题。在我们的案例中，订阅的是孪生体的事件，因为 Ditto 每次更新后都会触发一个事件，并在消息中包含孪生体的新状态。此外，还需要在授权上下文中指定一个有效用户，该用户至少拥有对孪生体的读取权限。我们可以在关联策略中查看现有用户及其权限。在 Eclipse Ditto 中创建连接后，每当孪生体更新，其新状态就会发布到我们创建的 Kafka 主题中。

安装 InfluxDB 时，也选择了 Helm 版本。服务类型设为 NodePort，并为管理员设置了专用密码，从而避免软件包需要重启时出现问题。安装过程中，还需要创建一个与自定义存储类相关联的持久卷。尽管 InfluxDB 提供了非常直观的界面，也是主要使用的界面，但安装并配置 Influx CLI 以使用用户管理等专属功能也很有意义。

配置好 InfluxDB 后，我们需要收集先前创建的 Kafka 主题中公开的数据。为此，我们使用 Telegraf，这款工具同样通过 Helm 安装。要向 Telegraf 指定所选插件及其配置，安装时需要使用一个自定义的 YAML 格式值文件。除了 Kubernetes 部署相关参数外，该文件中还包含一个配置部分。此时，务必查阅该工具的文档，了解每个插件所需的参数，无论它们属于输入端还是输出端。在我们的案例中，输出端使用第二版 InfluxDB，其中一个字段需要填写此前在 InfluxDB 中创建的、具有写入权限的令牌。输入端则使用 Kafka 消费者，并指定消息代理的地址以及它应订阅的主题。应用此配置后，我们的 InfluxDB 实例就应已开始接收来自 Kafka 的指标。

最后是 InfluxDB 与 Grafana 之间的连接，这样就可以通过 Grafana 对 InfluxDB 数据库执行查询。面板将显示查询结果。首先，已安装 Grafana 图表并启用持久化。工具准备就绪后，在同一界面中将 InfluxDB 配置为数据源，指定使用 Flux 作为查询语言的版本，并填写其余所需数据。工具确认连接成功后，我们便可以在 Grafana 中呈现孪生体信息，从而完成所有现有技术之间的连接，并着手开发额外功能。

4.2. Ditto-Extended-API 的开发

Eclipse Ditto 为我们提供了 Ditto Thing 实体，它始终属于某个命名空间，基本由一个标识符以及一系列属性和特征组成。属性对应实体的静态部分，而特征对应动态部分。借助这项技术，我们可以完全自由地定义模型的形式，以及工具的使用方式。如果一个 Ditto Thing 对应单个传感器，我们可以将制造信息等存储为属性，而它接收的数据则视为特征。另一种方式是让 Ditto Thing 实体封装一组传感器，并在其特征中分别存储每个传感器的数据。

为了理解所作出的设计决策，我们有必要先澄清一些有关数字孪生的概念。数字孪生既可以被视为接收来自单个设备信息的实体，也可以是由其他实体组成的实体，而这些实体也可被视为孪生体。因此，孪生体可以表示为一棵树，其中每个叶节点代表一个传感器。这样一来，拥有三台机器人且每台机器人都配有特定传感器的工厂，就可以如图3所示进行表示。

关于孪生类型，以工厂为例，如果机器人1和机器人2是同一型号，我们可以创建一种类型，以便在工厂添加更多同类型机器人时简化其创建过程。此外，组成机器人的传感器也应各自形成类型，因为例如，传感器2（因此也包括传感器4）可以与传感器5属于同一型号，即使它们属于不同的机器人类型。

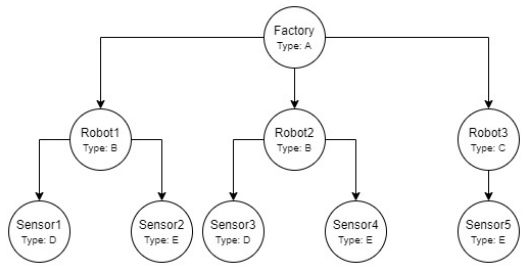


图3：以树形式表示的数字孪生示例。

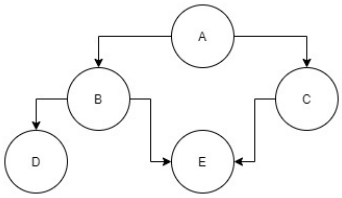


图4：以图形式表示的孪生体类型示例。

这样一来，孪生类型便构成一个无环有向图，在本例中可如图4所示。

遵循这些指导原则，我们设计了一个小型服务，该服务以符合这些属性的方式使用 Ditto Thing 实体，并将 Eclipse Ditto 作为其数据库。图5展示了该应用如何使用 Ditto 实体的示意图。基本上，我们利用了 Ditto Thing 的属性部分对应任意 JSON 对象这一事实：除了所需的静态信息外，还可以在其中加入一系列特定属性。首先，我们添加了一个布尔属性 *isType* ，用于指示该 Thing 是否对应某种类型。根据此值（如果该属性不存在，则视为 false），将施加不同的限制。如果该 Thing 不对应某种类型，则可以使用文本属性 *type* 指示该孪生体所属类型的标识符（如果有）。另一方面，我们还建立了父级和子级体系。在这两种情况下，Thing 都可以拥有任意数量的子级，但父级的情况并不相同：孪生体最多只能有一个父级，而类型可以有多个父级。这是因为孪生体对应于真实实例，按照逻辑，它们只能位于一个位置；

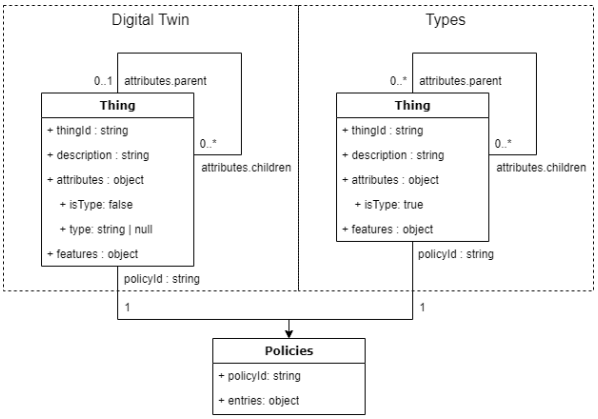


图5：ditto-extended-api 对 Ditto 实体的使用。

而类型对应于抽象实例，可以成为多个不同实例的一部分。为满足这一要求，对于孪生体，*parent* 属性对应一个包含父级标识符的文本字段；而对于类型，该属性是一个 JSON 对象，其中每个键都是其某个父级的标识符，值为 1。两个实体共有的 *chil-dren* 属性，其格式与后者类似，不同之处在于：孪生体的每个子级标识符对应的值必须严格等于 1，而类型对应的值可以大于或等于 1。这四个属性不能手动添加或修改，尝试执行此操作时将收到错误消息。

该服务作为构建于 Eclipse Ditto 之上的一层，提供了一个与 Eclipse Ditto 非常相似的 API，同时考虑到了上述所有限制。因此，如果我们要创建一个新实体，该服务会根据要创建的是单个孪生体、类型、由类型创建的孪生体，还是作为另一个孪生体子级的孪生体，自动初始化相应的特定属性。编辑时，始终会确保特定属性不被修改，并且只能通过特定调用修改父子关系。另一方面，查询返回的实体会隐藏 isType 属性，因为这些实体将被视为独立实体。删除孪生体时，可以选择仅删除该实体并让其所有子级成为无父级实体，或者级联删除该实体及其所有子级。对于

类型，只能选择第一种方式。与 Ditto 策略实体的交互保持不变，但新增了一个查询，用于返回所有现有策略的列表。图 6 概述了 API 当前提供的调用。

该服务使用 NodeJS 框架开发，其编程语言为 JavaScript。这个开源框架在运行时运行，在开发服务器端工具时能提供出色的性能。

实现并测试完成后，该服务被打包为 Docker 容器，并部署在 Kubernetes 上。需要注意的是，实现过程中未使用 ES6，因为它与集群中使用的 Kubernetes 版本不兼容。最后，要使用该服务，必须将 Eclipse Ditto 的部署 URI、与其交互所需的有效凭据，以及 Ditto 策略数据库的 MongoDB URI 设置为环境变量。

4.3. Grafana 数字孪生应用插件

Grafana 提供三种类型的插件：数据源插件、面板插件和应用程序插件。在本例中，我们关注最后一种，因为它允许我们在 Grafana 应用程序左侧栏中添加一个新部分，其中包含一个页面，并可按需添加任意数量的选项卡。每个选项卡中都可以添加任何类型的功能，不过需要注意的是，Grafana 有一套既定样式，因此建议使用这些样式以保持一致性。除了这个页面，每个插件还包含一个配置部分，它同样是一个带选项卡的页面，通常可以在其中找到插件自述文件、启用/禁用插件的选项，以及（如果有的话）插件正常运行所需的所有配置选项。

该插件应作为平台的前端，为用户提供一个整合平台各项功能的界面。具体而言，它必须支持通过 Ditto-Extended-API 服务管理数字孪生和类型，在 Eclipse Ditto 中直接管理策略和连接，通过开发的两个服务连接 Kafka-ML，并创建在 Eclipse Hono 中创建设备。所有这些操作都将通过 API 调用完成，以尽可能减少插件中的业务逻辑。

为此，主页将为每种实体分别设置一个选项卡：孪生体、类型、策略和连接。该选项卡会根据 URL 中收到的参数调整显示方式。初始时，它会显示该实体的所有元素列表；不过，对于孪生体和类型，可以将列表限制为不属于任何其他元素（没有父级）的元素，因为这些元素被视为主要实体。所有元素都可以选中，以便查询、编辑或删除。查询孪生体或类型时，会显示该元素的信息，以及用于查询其父级和子级的列表。查看孪生体时，还可以管理它与 Eclipse Hono 和 Kafka-ML 的关系。要建立这些关系，必须为每种技术选择一个有效的连接。在连接部分，可以创建 Eclipse Hono 租户并将其连接到 Eclipse Ditto，也可以将已创建的租户连接到 Eclipse Ditto。

配置部分仅包含用于填写地址和凭据的字段，这些信息是使用构成平台的不同技术所必需的。如果未填写这些字段，应用程序将无法运行。

在实现方面，Grafana 为每种类型的插件都提供了一个模板，既可作为基础，也能帮助了解插件的工作方式。所有插件都包含一系列必要文件，以便 Grafana 检测并配置插件。编码使用开源的、基于组件的 JavaScript 框架 ReactJS。该框架的主要用途是为单页应用创建用户界面。每个选项卡都可以分配一个 ReactJS 组件。在本例中，该组件将用作组件选择器，以便呈现实体的不同模式：查看其元素、查看单个元素以及编辑单个元素。此选择将通过 URL 中包含的参数完成。另一方面，Grafana 提供了一系列库，可与该工具更好地集成。使用 grafana/ui 库对于保持插件中的表单和其他视觉元素与 Grafana 的一致性至关重要，

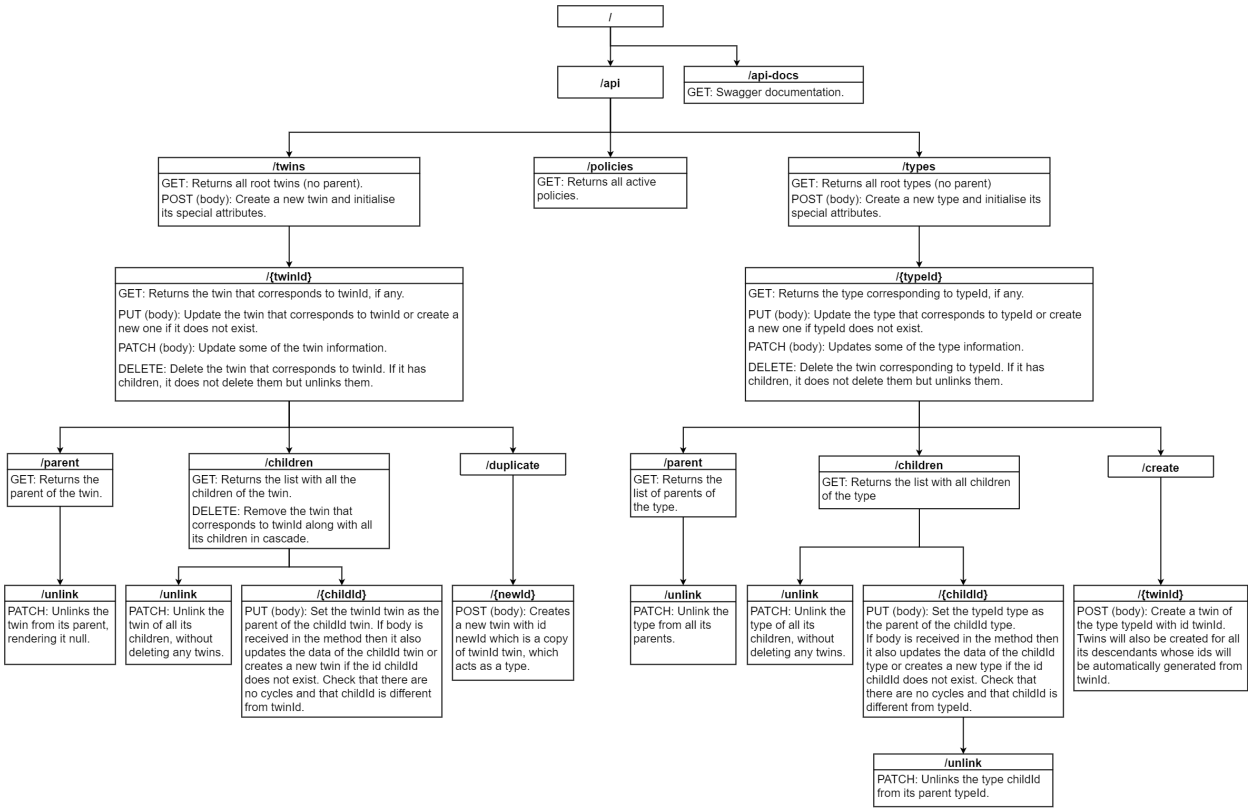


图 6：ditto-extended-api 中可用的 API 调用。

并保持表单及其他视觉元素与 Grafana 的一致性。

插件正确构建后，必须将其代码添加到 Grafana 插件文件夹，并在应用程序的相应部分中启用。配置中的必填字段填写完毕后，插件即可使用。

4.4. 使用 *Kafka-ML* 对 *Eclipse Hono* 进行错误检测

Eclipse Hono 可以将从多个设备接收的数据汇集到单个 AMQP 1.0 输出中，并可在设备所属租户对应的端点中查询这些数据。该端点的格式为 *telemetry/tenantid*，其中*tenantid* 是租户标识符。Eclipse Hono 的问题在于，其功能不包括任何用于控制特定设备是否定期发送数据的方式。它最接近的特征是作为设备通知接收的 *ttd* 字段。该字段允许你指出设备是否始终可用、不可用，或者指定设备在收到最后一条消息后还将可用多少秒。这对于在向设备发送消息前检查设备是否可用很有用，但与我们的目的实际上并不相关，因为该字段无法针对 AMQP 和 MQTT 协议进行手动配置；即使可以配置，我们也不知道每台设备发送数据的频率。

所需的功能正是如此：持续验证设备是否定期发送数据。此外，如果在通常的时间间隔内未收到数据，则会假定传感器出现了某种错误，并通过机器学习生成预测数据，以弥补信息缺失。这种错误检测不能依赖于接收传感器发送的任何特殊消息，因为服务必须能够应对所有类型的错误，包括可能导致连接中断的错误。

为此，设计了一项服务

该服务会为每个需要监控的 Eclipse Hono 租户启动一个线程。在这些租户中，用户指定的设备将受到监控。如果未收到任何设备的数据，则会将最后获取的值以及预测所需的信息按照用户指定的格式发送至 Kafka-ML。租户和设备的监控可以轻松停用和启用，而不会删除其余数据。系统将使用 MongoDB 数据库存储建立连接和生成 Kafka-ML 条目所需的所有信息。每个租户需要提供的信息的具体内容如图 7 所示。

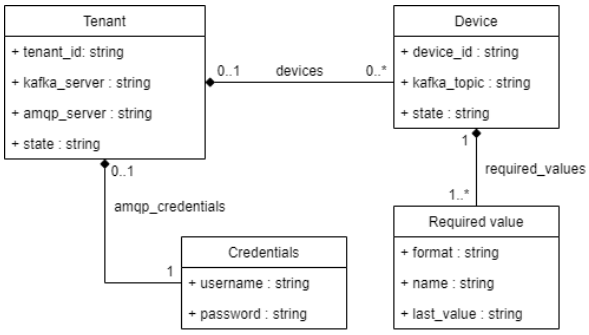


图 7：Eclipse Hono 与 Kafka-ML 错误检测的实体

线程的管理将通过 API 完成，该 API 支持创建、查询、删除、激活和停用租户及设备。可用的调用见图 8。应用程序启动后，将启动所有对应于标记为活动状态的租户的线程；若应用程序关闭，则会预先终止所有线程并断开所有连接。

每个线程都会建立两个连接：一个连接到与该租户对应的 Eclipse Hono 端点，另一个通过 Kafka 连接到 Kafka-ML 工具。此外，分配给该租户的每个活动设备都会关联一个永久计时器，该计时器初始处于关闭状态，且未设定间隔。计时器关联了一个函数，该函数必须在计时结束后触发，并负责向 Kafka-ML 发送相应的消息。线程初始化后

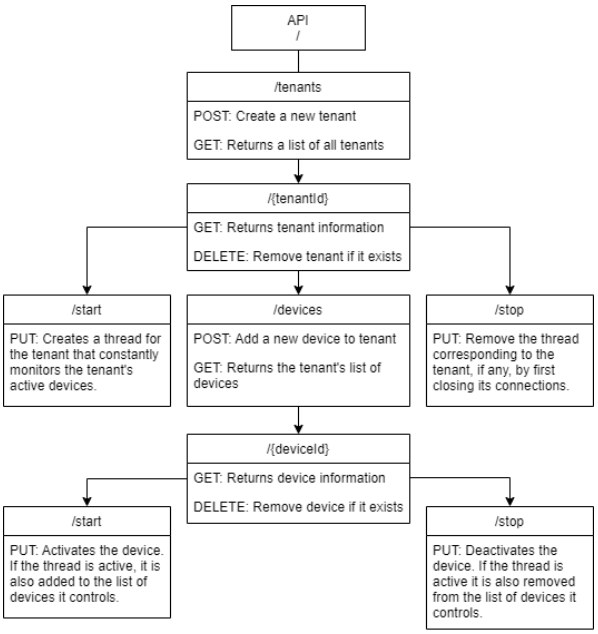


图 8：使用 Kafka-ML 对 Eclipse Hono 进行错误检测的端点。

初始化完成后，它将进入循环，直到收到表示必须停止并关闭连接的事件。该线程将充当 AMQP 端点的接收器，不断等待从该端点接收消息。收到消息后，它会检查消息是否来自某个活动设备；如果是，则对其进行处理。否则，将忽略该消息。消息处理包括将收到的值保存到设备所需的_值数组中各自的 last_value 字段，并重启相应的计时器。计时器的时间间隔为该设备收到这条新消息与上一条消息之间的时间差，向上取整后再加上 0.2 秒以留出不确定性。如果这是该设备收到的第一条消息，则应启动计时器，而不是重置计时器。计时从此刻开始，之后有两种情况：在计时器触发前，设备收到新消息并重启计时器；或者计时结束，将相应消息发送到 Kafka-ML，并按上一次设定的间隔重启计时器。在后一种情况下，计时器将持续重启并向 Kafka-ML 发送消息，直到设备收到新消息。该消息与上一条消息之间的时间间隔

```
"required_values": [
  {
    "format": "float64",
    "name": "$year"
  },
  {
    "format": "float64",
    "name": "$month"
  },
  {
    "format": "float64",
    "name": "$day"
  },
  {
    "format": "float64",
    "name": "temperature",
    "last_value": null
  },
  {
    "format": "float64",
    "name": "humidity",
    "last_value": null
  }
]
```

图 9：所需的_值字段示例

不应计入，因为两者的差值可能过大且不一致，因此应维持上一次设定的时间间隔。

发送 Kafka-ML 消息时，会查询设备所需的_值数组，并按照指定格式，将指定元素按出现顺序组成一个数组。若要表示时间值，必须将值名称定义为 datetime 库中的字段名称，并在前面加上符号 $。例如，要表示年份，值名称必须为 $year。对于其他值，则使用最近接收到的值，这些值必须存储在同一个对象中。构造完成后，该数组将以比特形式发送至 Kafka-ML 工具。这样，所需的值例如对于一个需要以 float64 格式发送数组的传感器，格式为 [year, month, day, temperature, humidity] ，则对应于图 9 中的对象。

该服务使用 Flask 框架实现，其编程语言为 Python。Flask 在创建 API 时提供了极大的便利，而且与 NodeJS 不同，它支持多线程处理。主要使用的库是 Python 线程处理库，它提供线程和定时器。其余库中，最值得关注的是用于支持不同连接的库：用于 Kafka 生产者的 kafka-python、用于 AMQP 1.0 接收器的 python-qpid-proton，以及用于操作 MongoDB 数据库的 Flask-PyMongo。此外，值得一提的还有 Marshmallow 库，它会在将用户发送的数据插入数据库之前，检查数据是否符合指定模式。服务实现完成并通过相关测试后，便将其容器化以用于 Docker，并部署到 Kubernetes 中。

4.5. Eclipse Hono 到 *Kafka-ML*

Eclipse Hono 到 Kafka-ML 服务旨在自动输入传感器数据，以供部署在 Kafka-ML 中的其他机器学习模型使用，例如能够预测孪生体未来状态或无法测量的孪生体特征的模型。

该服务的设计和实现与上一节所述的 Eclipse Hono 与 Kafka-ML 错误检测服务基本相同，因此此处不再赘述。API 的设计与构建、线程和连接的管理，以及将从 Eclipse Hono 接收的消息映射到 Kafka-ML，均与上述相同。可以将其视为对前一服务的简化，因为其区别在于：无论来自活动设备的消息之间间隔多长，都会在每次收到消息后向 Kafka-ML 输入主题发送一条消息。因此，*ditto_*消息字段不会存储最近收到的属性值，因为从指定设备收到的所有消息都会被映射，并直接发送到相应的 Kafka-ML 输入主题。

4.6. Kafka-ML 到 *Eclipse Ditto*

Eclipse Ditto 支持通过多种类型的连接接收消息。在

开发该平台时，与 Kafka 的连接仍在开发中，因此当时可创建的连接包括 AMQP、MQTT 和 HTTP。对于消息，Eclipse Ditto 要求它们采用 Ditto Protocol 格式，以便理解要执行的操作以及要对哪个孪生体执行该操作。如果无法更改连接生产者发送消息所采用的格式，则可以对连接应用负载映射器。Eclipse Ditto 集成了多种类型的映射器，其中最灵活的是 JavaScript 映射器。另一方面，Kafka-ML 会以数组格式将预测值发布到输出主题，目前不提供设置特定格式的方法。

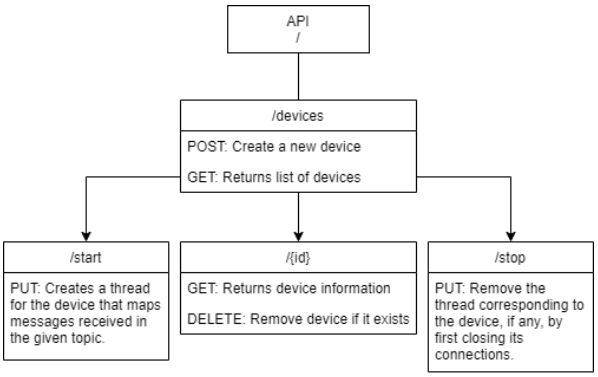


图 10：Kafka-ML 到 Eclipse Ditto 的端点。

在我们的案例中，需要接收 Kafka-ML 预测的数据，以便完全或部分更新相应孪生体的状态。由于 Eclipse Ditto 无法直接连接到 Kafka-ML 输出主题，因此需要一项服务从该主题收集信息，按照给定格式将其转换为 Ditto 协议，并通过 Eclipse Ditto 提供的某个连接发送出去。未来，可以用直接连接到 Kafka-ML 输出主题的方式替代这项服务，并为这些主题配置各自的 JavaScript 负载映射器。

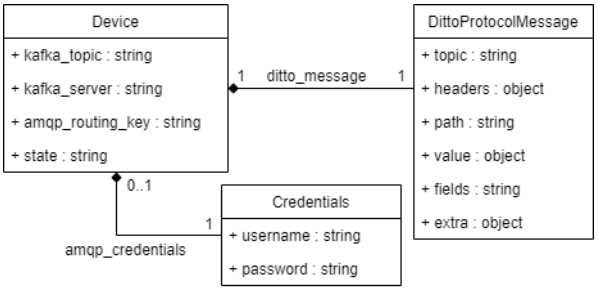


图 11：Kafka-ML 到 Eclipse Ditto 的实体。

该服务名为 Kafka-ML-to-Eclipse-Ditto，其设计与上一节所述的服务类似。总体而言，它会为每个需要提取信息的 Kafka-ML 输出主题创建一个线程，对信息进行转换，然后将结果发送至 AMQP 0.9.1 消息代理。这些线程可通过 API 管理，如图 10 所示，可创建、查询、删除、启用和停用线程。所有线程使用相同的 AMQP 代理 IP，该 IP 将在服务的环境变量中设置。每个线程运行时都需要图 11 所示的信息，包括连接 Kafka 和 AMQP 所需的数据、线程的初始状态，以及将发送给 Eclipse Ditto 的消息模式。该模式是采用 Ditto 协议格式的 JSON 对象，用户需要在值部分标记值应放置的位置，这些值来自 Kafka-ML 的值应放在相应位置。为此，需要在大括号中注明该值在从 Kafka-ML 接收的数组中的位置编号。例如，如果我们有一个温度和湿度传感器的数字孪生体，而 Kafka-ML 向我们发送一个数组，其中第一个值是预测温度，第二个值是预测湿度，那么 ditto_message 字段应如图 12 所示。

至于线程，每个线程都会为其对应的主题创建一个 Kafka 消费者，并与 RabbitMQ 消息代理建立一个 AMQP 连接。初始化后，线程会持续检查主题是否有新消息，直到收到要求其停止并关闭连接的事件。收到消息后，给定模式副本中的相应元素将被接收到的值替换，生成的消息随后会通过 AMQP 发送到用户指定的队列。

与设计一样，实现方式也与之前的服务十分相似。Flask、Flask-pymongo、线程处理和 Mashmallow 也

```
"ditto_message" : {
  "topic": "test/DHT22/things/twin/
      commands/modify",
  "path": "/features",
  "value": {
    "temperature": {
      "properties": {
        "value": "{0}"
      }
    },
    "humidity": {
      "properties": {
        "value": "{1}"
      }
    }
  }
}
```

图 12：Ditto_消息字段示例

用于实现 Error-detection-for-Hono-with-Kafka-ML 服务中所述的相同目的。另一方面，此处使用 kafka-python 库创建能够读取 Kafka-ML 输出主题的 Kafka 消费者，并使用 pika 库建立 AMQP 0.9.1 连接。还需注意，在 Kubernetes 中部署 RabbitMQ 实例时，使用了相应版本的 Helm；需要启用持久化，指定先前创建的持久卷声明，启用卷权限，并将服务类型设置为 NodePort。

实现、测试、容器化并部署后，即可使用该服务。为了让 Eclipse Ditto 能够接收生成的消息，我们在其与 RabbitMQ 之间建立了 AMQP 0.9.1 连接，并在连接的源地址部分填入待消费的 AMQP 队列名称。此外，我们在策略中为其数据需要预测的孪生体添加了一个新主题，并将该主题纳入连接的授权上下文。为在 RabbitMQ 中创建连接所需的凭据，我们使用了该工具自身提供的界面。此外，为了区分来自 Eclipse Hono 和来自 Kafka-ML 的值，我们更新了 Telegraf 配置，将消息头中的 ditto-originator 字段设为标签键。为使二者有所区别，配置完成、创建必要的实体并激活线程后，应该就能毫无问题地接收到孪生体的更新。

4.7. Grafana 的 *Unity* 面板插件

如前所述，Grafana 是该平台的前端。它的主要功能是创建由面板组成的仪表板，以特定方式呈现对数据源的查询结果。该工具提供的大多数默认仪表板都侧重于通过某种图表来表示数据。为了创建自定义仪表板，Grafana 支持开发面板插件。这类插件与前文开发的应用插件类似，不同之处在于，它会提供所输入查询的数据结果，以及用户选择的面板选项，以构建相应的表示形式。

此处的目标是对数字孪生及其状态进行三维表示。此外，还希望能够与该表示进行交互，使交互影响 Grafana 仪表板上显示的数据；同样，该表示也会根据孪生体的状态而发生变化。目前，Grafana 尚无支持此功能的面板插件。因此，为满足这一需求，我们将创建一个插件面板，用于显示在 Unity 中开发的三维模型并与之交互。尽管该插件的开发重点是应用于数字孪生领域，但我们也希望尽可能抽象其功能，以便在其他领域复用。

该插件基于 React Unity WebGL 库实现，可将导出为 WebGL 格式的 Unity 编译版本嵌入使用 React 框架开发的任何应用程序中。此外，它还支持 Unity 模型与 React 应用程序之间的双向通信。该库提供一个 UnityContext 对象，需使用构成 WebGL

格式的四个文件对其进行初始化。要访问这些文件，必须将它们放置在 Grafana 公共文件夹中。要显示编译版本，只需在 JSX 元素中引用该对象，该元素也由此库提供。为了确保 Unity 模型正常运行，必须在其某个脚本中禁用对所有键盘输入的捕获，否则可能会干扰构成应用程序的其他 JavaScript 元素的运行。

要在 Unity 模型中接收来自 React 应用程序的信息，必须先在每个需要接收此类信息的 Unity 对象中定义一个函数。该函数将接收应用程序发送的数据作为参数，通常是文本或数字。在应用程序端，必须使用先前创建的 UnityContext 对象的 send 方法，并指定所指的 Unity 对象名称、上述函数的名称，以及（如有）要发送的数据。在我们的案例中，我们希望将用户在 Grafana 面板中输入查询后得到的数据发送给相应的 Unity 对象。这些信息包含在 Grafana 提供的 data 变量中，该变量作为组件的参数传入。此变量包含一个序列列表，具体内容取决于数据的分组方式。查询结果中的每个分组对应一个序列。例如，如果先按 thingId、再按特征对数据分组，并且有两个孪生体拥有相同的两个特征，那么就会生成 4 个分组，因此也会有 4 个序列。每个序列都包含一组字段，每个字段都有一个值列表。最后一个分组列的值将被视为独立字段，而其他分组列的值则显示为这些字段的标签。因此，在上述示例中，每个序列都包含一个时间字段和一个以其分组特征命名的字段，并且这两个字段都带有对应的 thingId 标签。考虑到以上情况，插件面板将从 Grafana 变量中提取相关数据并进行分析，然后为当前筛选的设备构建消息。消息发送后，将作为参数由相应 Unity 对象中的函数接收，并由该函数执行相应操作。

每次 data 变量发生变化后，都会执行发送操作，也就是说，每次 Grafana 接收到数据后都会发送。

要将数据从 Unity 模型发送到 React 应用程序，必须预先建立一个事件。在 Unity 模型中的特定文件夹内，需要添加一个 JSLib 文件，以定义事件名称和要发送的参数类型。需要发送信息到 React 应用程序的 Unity 脚本可以导入并使用此事件。要在面板插件中接收数据，必须调用 UnityContext 对象中相应的方法，并指定要监听的事件名称以及接收事件后要执行的函数。此函数必须包含事件中指定的参数。在我们的示例中，我们在 Unity 中定义了一个脚本，当用户点击某个 Unity 对象时，该脚本会发送一个包含其标识符的事件。这些对象的标识符必须与用于在 Eclipse Ditto 中构建孪生体的标识符相对应。此标识符将设为 Grafana 仪表板中变量的值，该面板位于此仪表板上。Grafana 允许定义多种类型的变量，并将其包含在查询中，从而使仪表板更具动态性和交互性。这样，仪表板的其他面板便可根据三维模型中被点击的 Unity 对象显示相应信息。

面板的所有可变元素，例如函数名称、事件名称、WebGL 编译位置和仪表板变量名称，都已定义为面板选项，以便用户轻松修改这些值。

与上一节开发的应用程序插件一样，必须将开发好的代码添加到 Grafana 插件文件夹中才能使用。激活后，该插件可添加到任何 Grafana 仪表板中，并可根据用户的偏好进行配置和调整。

5. 用例：石化工业中的虚拟分析仪

本文介绍的数字孪生平台将通过一个石化工业4.0用例进行验证。该用例旨在定义一种虚拟分析仪，能够预测

CEPSA 一种成品（润滑剂）的凝固点，该预测基于操作条件和原料属性。该流程在 CEPSA 位于西班牙圣罗克能源园区的工厂中进行，该工厂是西班牙最大的炼油厂之一。凝固点是一项重要参数，由于其特性，必须在实验室中测量。通过监测过滤器等不同操作条件，目标是借助 Kafka-ML 实时预测凝固点的状态，以便更好地控制流程。我们将在框架内将这一持续预测与受监测传感器的状态共同建模为数字孪生。此前，石化行业也开展过数字孪生研究。例如， [17], 提出了一种用于石化行业催化裂化装置生产控制的数字孪生。本文进一步对该行业中的一个流程进行建模、预测和三维可视化。

```
{"thingId": "cepsa:LSRC3002.PF",
```

图 13：用例中某个传感器的 Eclipse Ditto 示意图示例。

公司向我们提供了所需传感器的实时数据（通过MQTT协议）和历史数据（用于训练机器学习模型）。这些传感器各自都将被视为独立的孪生体，它们共同构成所要构建的主孪生体。通过将运行方式和描述相同的传感器归为一组，我们确定了不同的数字孪生类型，并为每种类型和孪生体定义了一个 Ditto Thing方案。在本案例中，所有传感器都会接收一个数值及其采集时间，而且每个传感器数据的接收格式也都相同。为便于映射和数据查询，所有孪生体和类型都使用相同的特征部分。通过连接到 Ditto-Extended-API 服务的 Grafana 数字孪生插件，我们创建了不同的孪生体。所有元素都归入同一个 Eclipse Ditto 命名空间（用于对信息和数字孪生进行分组的逻辑方式）。图 13 展示了其中一个传感器的 Ditto Thing 方案。

为了发送实时数据，已在 Eclipse Hono 中为 MQTT 连接创建了一个租户，并为每个可用传感器添加了已配置凭据的设备。

每个设备的名称与其对应孪生体的 thingId 相同，从而便于与 Eclipse Ditto 建立连接。此外，在同一连接中，还应用了 JavaScript 映射器，将接收到的消息转换为 Ditto Protocol 格式。

此时，孪生体应能正确接收数据，而孪生体随时间变化的状态将存储在 InfluxDB 中，因此可以根据需求创建 Grafana 仪表板。

作为该用例目标而开发的凝固点预测模型已部署在 Kafka-ML 中。为输入数据编写了一个简单脚本，定期调用 Eclipse Ditto，以收集代表模型所需传感器的孪生体当前状态。该模型的输出数据由 Kafka-ML-to-Eclipse-Ditto 服务收集，以更新数字孪生中包含的凝固点特征。

为了三维呈现孪生体的状态，我们在 Unity 中创建了一个模型，其中包含足够的元素来表示机器的每个传感器。在该模型中，每个元素都已重命名为其所代表传感器的 ID，并且已实-

现摄像机移动和通过点击选择元素所需的代码，以及确保模型在 Grafana 中正确运行所需的脚本。模型的 WebGL 导出已添加到 Grafanapublic 文件夹中，并且其中一个仪表板已加入 Unity 面板。

图14展示了针对所述用例开发的数字孪生三维表示的最终结果。最终获得了一个易于适配和扩展的工业流程孪生体，其能够以赏心悦目的方式呈现实时时状态，并使用不同类型的图形以及三维模型；该三维模型在接收传感器数据后，可以显示机器上的相关数据，例如机器的实际运动，并允许用户进行任意类型的交互。同样，该平台支持通过 Eclipse Ditto API 轻松查询孪生体的当前状态，也可以使用 InfluxDB 提供的任意查询选项查询其随时间变化的状态。此外，在预测方面，机器学习模型可以轻松集成到孪生体中，其生成的值将被视为孪生体的另一项特征。

6. 评估

接下来，将开展一系列测试，通过延迟分析验证性能和可扩展性，并利用上一节所述的构建方案评估平台的可用性，同时考虑该架构所包含的不同流程。

测试1将探测数字孪生核心功能流程的延迟和吞吐量，并考察其与连接的传感器数量以及所用客户端/连接数量之间的关系。同时，测试2将计算相同的属性，但重点考察平台的预测流程，并考虑并发客户端的数量。最后，测试3将通过获取服务的平均恢复时间，并检查这些服务发生故障是否会导致数据丢失，来检验平台的容错能力。

6.1. 实验设置

硬件配置。所有实验均在 VMware vCloud 私有云基础设施中的一个五节点 Kubernetes 集群上进行。每个节点有 2 个插槽，共配备 4 个虚拟 CPU 和 16GB 内存。发送信息并用于测量结果的客户端是一台配备 64GB 内存、1 个 CPU 和 10 个核心的 PC。

软件配置。五个节点均在 Ubuntu 16.04.7 LTS 上运行 Kubernetes v1.19.3 和 Docker 19.03.13。一个节点部署了 Kubernetes 主节点，其余四个节点为 Kubernetes 工作节点。运行客户端的 PC 使用 Ubuntu 服务器。

6.2. 测试*1* - 核心功能流程

该测试将评估所述石化工业用例中使用的数据流，从通过 MQTT 向 Eclipse Hono 发送数据到将数据存储在 InfluxDB 中这一过程的延迟和吞吐量。这里有两种测试情形：不同数量的传感器同时接收数据，以及不同数量的客户端/连接同时发送数据。测试未考虑所发送数据的大小，因为所有消息格式相同，因此大小几乎没有差异。

6.2.1. 关于传感器数量

在此测试中，我们按照公司指定的格式发送历史数据，为每个传感器建立一个 MQTT 连接，并通过使用线程逐步增加同时发送数据的传感器数量。随后查询每条输入数据存入 InfluxDB 的时间，并比较各时间以计算延迟和吞吐量。每种传感器数量对应的测试结果为十次重复测试的平均值。由于公司向我们提供了其 27 个传感器的数据，因此本次测试中可同时调用的传感器数量上限为 27 个。图15展示了所得结果。

显然，随着受影响的传感器数量增加，延迟几乎呈线性增长，而吞吐量则呈指数级下降。这一结果符合常态；此外还应考虑到，在实际



图14：该用例数字孪生的三维可视化

场景中，平台的所有传感器不会同时或以相同频率接收数据，因此可以确定，随着受影响的传感器（此处指孪生体）数量增加，该平台仍能良好运行。

6.2.2. 与客户端数量相关

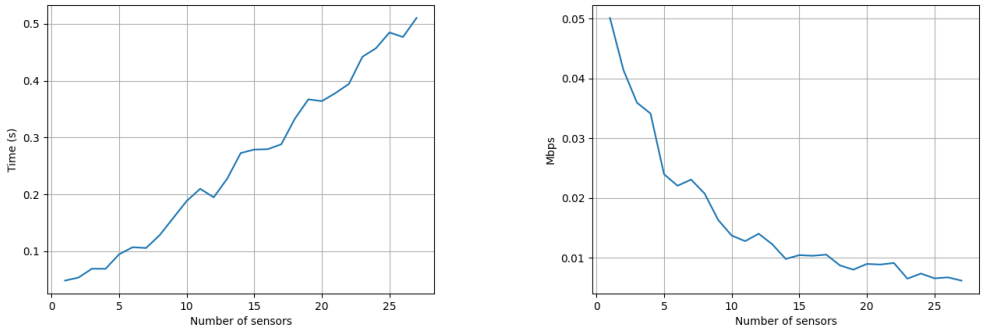
此测试与上一个测试相似，不同之处在于：不同数量的模拟客户端通过线程，仅向单个传感器发送数据更新。每条消息中发送的值始终唯一，因为每个客户端都会从 0 开始，将一个全局值递增 0.01。同样，每种客户端数量下的测试结果均为该测试重复十次后的平均值。图16展示了测试结果；为便于理解图表并与上文所述结果进行比较，客户端数量也限制为 27 个。

如图所示，并发客户端超过 20 个后，延迟超过 1 秒。另一方面，吞吐量则呈现出适度下降。这些结果并不构成问题，因为最常见的情况是孪生体或设备从单个数据源接收数据；例外情况是模拟或预测特征，此时客户端数量通常会增加一到两个。这种行为也属正常，因为将所有消息发送到单个孪生体可能会使其过载。因此，我们可以得出结论：随着客户端数量合理增加，系统能够做出正确响应。

6.3. 测试*2* - 机器学习预测流程

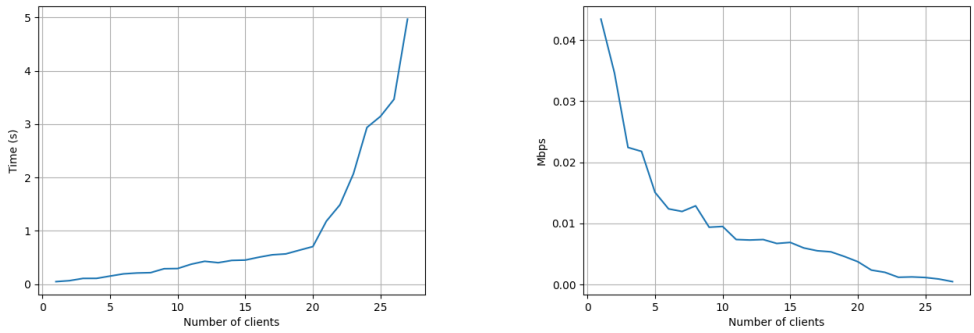
在此测试中，将计算架构中机器学习集成部分的延迟和吞吐量。由于该用例中只有一个预测模型在运行，用于冰点预测，因此测试始终会影响同一个 Eclipse Ditto 孪生体；我们将改变的是向模型发送输入数据的客户端数量。为了将每个客户端与模型的输出关联起来，我们使用了结果已知的输入数据，并避免在每次测试中重复使用这些数据。以特定客户端数量执行测试后，我们会查询 InfluxDB，获取每条数据的存储时间，以便随后进行比较。每项测试的结果均为执行 10 次后的平均值。图17展示了此项测试的结果。

就客户端数量而言，结果与第一次测试中显示的结果非常相似。延迟持续增长，直到并发客户端达到 17 个时超过 1 秒。类似地，吞吐量也大幅下降。这些



（a）延迟　　（b）吞吐量

图15：不同传感器数量下测试1的延迟和吞吐量



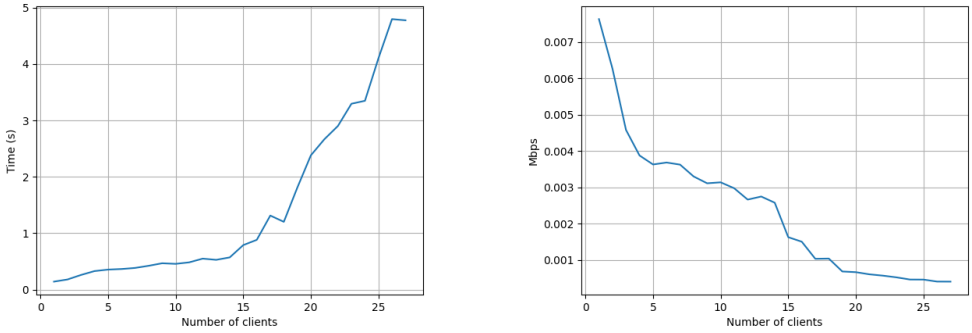
（a）延迟　　（b）吞吐量

图16：　　不同数量客户端下测试1的延迟和吞吐量　　的

相较于实时数据流，这些结果可以接受，因为两者之间的差异很小，而且必须考虑到流程中会针对每个客户端进行预测。此外，与前一种情况一样，通常由单个客户端发起数据流，因此并发客户端数量的限制不会成为问题。因此，我们可以确定，在通过机器学习进行预测的过程中，平台能够很好地应对并发客户端数量的增加。

6.4. 测试*3* - 容错性

这最后一项测试旨在检查平台的容错能力。为此，我们编写了一个脚本，每隔半秒通过 MQTT 向 Eclipse Hono 发送一条消息。在此过程中运行期间，会在 Kubernetes 中手动删除与待测试服务对应的 Pod。如果消息无法发送，则会根据需要反复重发，但仍保留首次发送的初始时间。测试结束后，将从 InfluxDB 中提取接收到的数据进行比较。该 Pod 的测试最大时间差将视为其恢复时间。每项测试结果均为 5 次测试运行的平均值。图 18 展示了测试结果。所选 Pod 按数据从发送到 Eclipse Hono，直到相应孪生体在 Eclipse Ditto 中更新这一过程的顺序排列，并按此顺序显示在图中。未纳入两个或更多 Pod 组合的结果，因为其结果与这些 Pod 中恢复时间最长者相同。



(a) 延迟　　(b) 吞吐量

图17：测试2的延迟和吞吐量

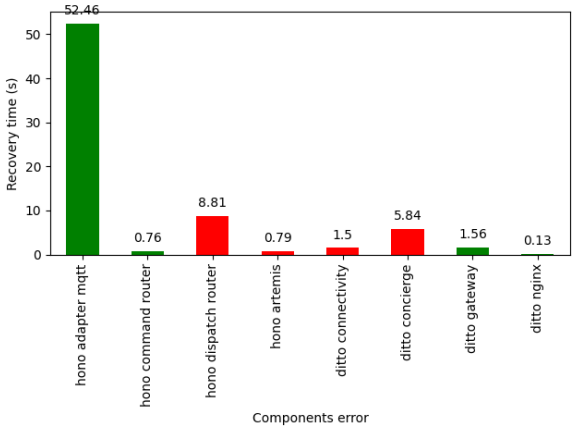


图18：各服务的恢复时间

首先需要指出，这些结果在很大程度上取决于部署这些服务的集群性能。可以看出，各服务的恢复时间往往非常短，表明测试结果非常理想。例外的是 Eclipse Hono MQTT 适配器，其恢复时间较长，这可能是因为它依赖于 MQTT 代理的创建、配置和连接。另一方面，通过查看测试获得的原始数据，我们可以确定哪些服务在恢复期间发生了数据丢失。发生数据丢失的服务在图 18 中以红色标出，未发生数据丢失的则以绿色标出。这并不是一个大问题，因为恢复时间很短，不过，其解决方案将作为未来工作的研究内容。综上所述，我们可以得出结论：该平台在应对错误方面表现良好，能够从错误中恢复，且不会造成严重的数据损失，也不需要人工干预。

7. 结论与未来工作

数字孪生正逐渐成为一种宝贵资源，可帮助人们更好地了解并预判资产在物理世界中可能面临的情况。尽管文献中已经提出了许多用于开发数字孪生的平台，但这些平台主要面向特定的垂直领域，要实现有效的数字孪生，仍需显著发展。本文提出了一个开源平台，用于开发可适配多种场景的有效数字孪生。我们所说的数字孪生，是指能够通过开放平台 Kafka-ML 无缝集成预测模型、借助 Unity 对三维表示的支持实现交互，并为简单和复杂模型提供统一界面，以实时可视化和管理物联网监测数据的数字孪生。为展示该平台的可行性，我们为石化工业中的一个工业流程构建了数字孪生。该流程用于监测和预测润滑剂生产中的凝固点。借助数字孪生，工厂用户可以可视

化工厂及其组件（例如过滤器）的状态，并直接在三维表示中查看，同时还能实时查看冰点预测结果。

作为该平台未来的工作，我们设想支持 FMI（功能模型接口）标准，以便能够仿真遵循该标准的复杂流程，并将其与平台的三维表示集成。此外，我们计划生成混合孪生体，利用边缘/雾计算系统提供的低延迟优势，并展示该平台在其他场景中的可行性（我们目前正在农业领域开展相关工作）。

参考文献

[1] M. Díaz, C. Martín, B. Rubio, 《物联网与云计算集成的最新进展、挑战及待解决问题》, 《网络与计算机应用杂志》67（2016）99–117。[2] D. De Silva, S. Sierla, D. Alahakoon, E. Osipov, X. Yu, V. Vyatkin, 《迈向智能工业信息学：人工智能在工业应用中的最新进展与未来方向综述》, 《IEEE工业电子杂志》14（2）（2020）57–72。[3] F. Tao, H. Zhang, A. Liu, A. Y. Nee, 《工业中的数字孪生：最新进展》, 《IEEE工业信息学汇刊》15（4）（2018）2405–2415。[4] A. Rasheed, O. San, T. Kvamsdal, 《数字孪生：从建模视角看其价值、挑战与赋能因素》, 《IEEE Access》8（2020）21980–22012。[5] A. C. Márquez, A. de la Fuente Carmona, J. A. Marcos, J. Navarro, 《基于预测分析和大数据工具为列车车轮轴承设计状态监测计划》, 《工业计算机》122（2020）103292。[6] A. A. Nazarenko, L. M. Camarinha-Matos, 《数字孪生在协作式信息物理系统中的作用》, 载于：《计算、电气与工业系统博士生会议》, Springer，2020年，第191–205页。[7] J. Conde, A. Munoz-Arcentales, A. Alonso, S. Lopez-Pernas, J. Salvachua, 《数字孪生数据与架构建模：以 FIWARE 为赋能技术的构建指南》, 《IEEE互联网计算》（2021）。[8] Y. Zheng, S. Yang, H. Cheng, 《数字孪生应用框架及其案例研究》, 《环境智能与人性化计算杂志》10（3）（2019）1141–1153。[9] J. Cheng, H. Zhang, F. Tao, C.-F. Juang, 《DT-II：面向智能制造的数字孪生增强型工业互联网参考框架》, 《机器人与计算机集成制造》62（2020）101881。[10] Y. Mo, S. Ma, H. Gong, Z. Chen, J. Zhang, D. Tao, 《Terra：用于复杂环境下稳健机器人部署的智能、灵敏数字孪生框架》, 《IEEE物联网期刊》8（18）（2021）14039–14050。[11] H. V. Dang, M. Tatipamula, H. X. Nguyen, 《基于云的数字孪生技术：利用深度学习进行结构健康监测》, 《IEEE工业信息学汇刊》（2021）。[12] V. Kamath, J. Morgan, M. I. Ali, 《面向智能工厂的工业物联网与数字孪生：用于应用设计和基准测试的开源工具包》, 载于：2020年全球物联网峰会（GIoTS），6月3日，线上，IEEE，2020年，第1–6页。[13] K. Shah, T. Prabhakar, C. Sarweshkumar, S. Abhishek 等, 《使用自由开源软件构建数字孪生框架》, 《IEEE互联网计算》（2021）。[14] R. P. Rolle, V. d. O. Martucci, E. P. Godoy, 《数字孪生模块化框架：开发与性能分析》, 《控制、自动化与电气系统杂志》32（6）（2021）1485–1497。[15] A. Khan, F. Shahid, C. Maple, A. Ahmad, G. Jeon, 《借助螺旋式数字孪生框架和 twinchain 迈向智能制造》, 《IEEE工业信息学汇刊》18（2）（2020）1359–1366。[16] C. Martín, P. Langendoerfer, P. S. Zarrin, M. Díaz, B. Rubio, 《Kafka-ML：将数据流连接到机器学习/人工智能框架》, 《未来一代计算机系统》126（2022）15–33。[17] Q. Min, Y. Lu, Z. Liu, C. Su, B. Wang, 《基于机器学习的数字孪生框架：石化行业生产优化》, 《国际信息管理杂志》49（2019）502–519。



Julia Robles于2021年毕业于西班牙马拉加大学，获得软件工程学位。此后，她加入马拉加大学的 ERTIS 研究小组，担任研究助理——助理，并且是马拉加大学 ITIS Software 研究所的成员。他的研究兴趣主要集中在数字孪生、物联网和人工智能领域。



Cristian Martín 获得了计算机工程硕士学位、软件工程与人工智能硕士学位，以及计算机科学博士学位，均来自Uni-

分别于2014年、2015年和2018年毕业于西班牙马拉加大学。目前，他是马拉加大学的博士后研究员。此前，他曾在多家采用射频识别技术的科技公司担任软件工程师，从事软件开发。他也是马拉加大学 ITIS软件研究所的成员。他的研究兴趣包括物联网与云计算／雾计算／边缘计算、机器学习、结构健康监测以及物联网可靠性的融合。



Manuel Díaz 是马拉加大学计算机科学系的正教授，也是 ERTIS研究组负责人及 ITIS软件研究所成员。他的研究

兴趣涵盖分布式与实时系统、物联网和点对点技术，尤其关注中间件平台和关键系统领域。近年来，他的主要研究方向是无线传感器网络和监测系统，特别是能源监测（FP7 e-balance项目）、水利基础设施监测（FP7 SAID项目）以及节能建筑（FP7 SEEDS）。他曾与 Tecnatom、Telefónica、Indra 和 Abengoa 等多家公司合作开展多项技术转移项目。他曾担任 FP6 SMEPP项目协调员，并作为 UMA 在 WSAN4CIP（ICT FP7）项目中的首席研究员。他还是衍生公司 Softcrits 的联合创始人，并担任其研发部门负责人。