**基于运行时知识图谱的智能家居应用开发的方法**

摘要综述：智能家居是物联网的一个重要的应用领域。但是，对于开发人员来说，智能家居应用场景的不同增加了理解这些场景的难度。对于开发人员来说，解决智能设备的编程接口的异构性和代码对于底层系统的耦合性都是重要的工作。此外，业务需求的复杂性和可变性也给应用逻辑的发展带来了不小的挑战。

在这篇论文中，展示了一个的方法，这是基于智能家居应用发展的运行时知识图谱。首先，定义了一个描述智能家居场景的概念模型。其次，智能设备的可管理性被抽象成运行时的知识图谱，这种知识图谱可以自动与相应的系统进行连接。最后，提出一种自动生成智能家居应用的方法。这种方法可以减少至少大约85%代码，并且一个真实的应用场景演示了可行性、有效性和对于智能家居应用发展这种新方法的好处。

1. 介 绍

数以万亿计的物联网 (IoT) 设备，例如智能可穿戴设备、智能机器人和其他智能应用，连接到网络中并产生大量数据。智能家居已成为物联网的重要领域。但智能家居应用开发难度大且复杂，这给智能家居应用以及智能家居的推广带来很大阻碍。从系统实现来看，这些应用通过智能家居设备的接口和分析、处理和反馈特定的智能设备和居住情况的信息，可以获得大量数据。在当前的智能家居应用的开发中，要想使用这些可编程的设备，就必须很清楚支持这些设备的操作系统，因此需要开发的人会处理操作系统的底层问题。这不仅分散了开发人员对应用程序逻辑的注意力，但也需要一个熟悉这些应用的底层操作系统的人。因此，开发人员面临许多挑战。首先，智能家居应用的开发是为各种场景。设备的多样性使智能家居服务越来越多元化，同时智能家居场景更加复杂多样。这场景的多样性给开发人员了解场景带来了更多的挑战。其次，智能家居应用被用来开发异构设备。设备数据的类型是不一致的，设备之间的通信是不方便的。处理编程接口的异构性以及代码和底层系统的耦合性对开发人员来说，也是一个重要的挑战。第三是智能家居应用用于来开发各种需求。业务需求的复杂性和可变性对应用逻辑的开发提出了很大的挑战。大多数的开发人员面对这些挑战，但是很难克服这些问题。因此，这有必要提供一种满足以上的这些问题的方法。MDE（model driven engineering, 模型驱动工程）是一种以建模和模型转换为主要途径的软件开发方法。MDE 旨在解决软件开发，通过提高抽象级别和自动生成应用程序工件。运行时的模型已广泛应用于不同的系统来支持数据操作、自我修复 和动态适应。

在这篇论文中，我们提出了一个基于智能家居应用开发的运行时的知识图谱。首先，提出描述智能家居场景的概念模型。第二，智能设备的可管理能力被抽象成运行时的知识图谱，这种图谱自动地与相应的系统相互连接。最后，提出一种自动生成智能家居的方法。这个方法只需要提供一个当前的智能家居场景的描述，配置需求文件，定义一组元模型和映射规则，从而极大地减少了手头上编码的工作量。通过在运行的知识图谱上执行程序，所有自动生成的应用逻辑都可以被执行。当需求发生变化时，开发者仅仅需要管理配置文件就可以快速实现场景的开发。开发人员不需要关注设备和数据，仅仅需要关注需求。

2.相关工作

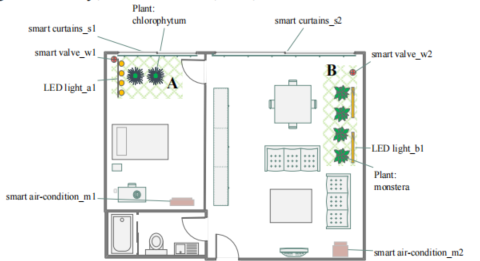
在智能家居应用的开发上，建模的方法被很多研究者所广泛使用。在软件工程领域，尤其对于这种被发明用于智能家居的工具和元数据而言，建模语言的选择是传统且严格的。有团队开发并提出一个智能家居系统来提高生活质量。他们应用一个简单的模型来表示住户的活动，和电器的操作，还有其他的开发者用各种方法验证了使用知识图谱给智能家居构建模型的可行性。然而，他们的工作并没有关注设备和使用者的需求之间的关系，也没有提供开发应用程序的方法。

在我们之前的工作中，我们提出了一个基于物联网应用开发的方法的运行模型。开发者仅仅需要构建传感器设备的运行模型和满足个性化的应用场景的定制模型。以下的段落将关注我们以往的在模型驱动工程领域上的工作。对于给定的元模型和一组给定的管理接口，SM@RT可以自动生成代码，这是为了将模型映射到有足够的运行性能的接口。如果用户改变元模型，SM@RT可以重新生成映射代码。更多的细节可以在之前的（支持运行时软件架构：基于双向转换的方法）这篇论文中找到，另外，对于不完整的形式化建模语言的情况而言，我们之前的工作提供元对象设施 (MOF)的 元模型支持向上兼容的扩展机制并自动生成用于模型集成的模型转换。但是，也很难从头开始来构建运行时的模型来满足智能家居应用的开发需求。本文中的方法建立在我们之前的工作上。

三、 方法概述

在本节中，我们将概述我们的方法。我们首先提出一个例子，它说明了智能家居发展中遇到的挑战场景。 然后我们简单介绍一个运行时基于知识图谱的智能家居应用开发。

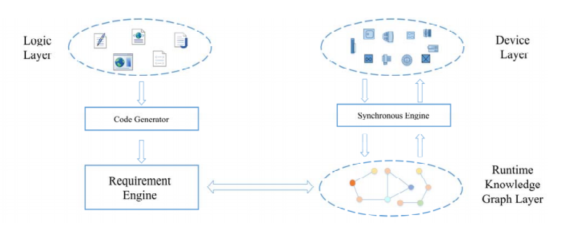
1. 示例



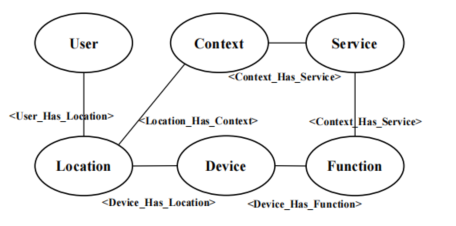
图一.示例

图1中的房子共有三个房间，每个房间有一些智能设备。其中，有一些智能种植的相关设备，如智能温度、控制装置、智能水阀、智能灯和智能窗帘。由这些智能设备提供植物生长的适宜的温度、湿度和光照强度。根据之前的智能家居场景开发方法，开发人员开发物联网系统是基于不同地理位置和不同的智能设备。一方面，应用场景可能包括需要协作的不同类型的设备管理。开发人员必须熟悉由智能设备提供的 API。另一方面，开发者需要自下而上地开发每个场景。开发人员必须编写许多不同的程序来管理类似的场景。除了编程困难、需求变化和设备变化可能会导致应用程序被重新开发。简单来说，就是可复用性比较差。以智能种植应用开发的场景为例，开发者面临以下三种应用程序的挑战：首先，对智能种植场景的需求是复杂的。开发人员需要考虑种植在 A 区的叶绿素的生长需要和B区种植的龟背竹。此外，不同的设备，需要考虑光照强度、土壤湿度等不同。开发人员并不容易查明所有参与智能种植场景应用的对象。其次，复杂的智能设备很多。开发人员需要熟悉智能设备的编程接口，例如智能水阀、智能灯、智能窗帘。应用程序开发是在低级别完成的，非常接近操作系统，需要开发者关注底层系统问题。底层 API 可以是非常复杂的，收集的数据量可能很大。最后，智能种植应用的业务逻辑是复杂多变的。当设备的位置发生变化时，应用程序也会发生变化或当植物类型发生变化时，开发者需要重新开发应用场景。这种不确定性造成大量不必要和重复地开发应用程序。

我们的方法使智能家居场景应用开发更简单、更高效。我们使用模型驱动的开发方法。图 2 展示了基于智能家居应用开发的运行时知识图谱的概览。首先，概念化设备层并自上而下抽象出智能家居场景中的对象。其次，构建智能设备的运行时的知识图谱来提供统一化的管理。第三，根据用户的需求，自动生成逻辑建模程序，并应用于运行时知识图谱，最后实现智能家居场景应用的开发。在我们的方法中，开发人员通过配置需求文件来开发智能家居场景应用，从而大大减少手动编码工作。



图二基于智能家居应用开发的知识图谱概览



图三智能家居应用场景概念模型

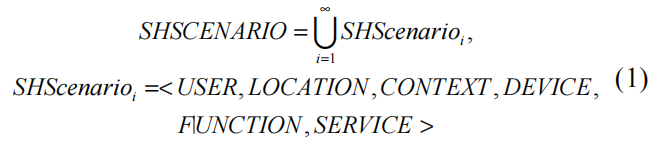
四、对于智能家居应用我们的方法

本节介绍基于智能家居应用程序开发方法的运行时知识图谱。智能家居概念模型是以构建智能的运行时知识图谱家庭场景为基础的。运行知识图谱的构建有利于智能家居场景应用的代码自动生成。

1. 智能家居概念模型的定义

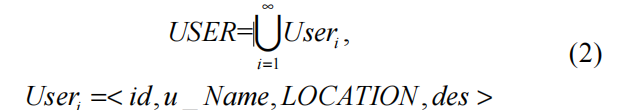
图 3 显示了智能家居场景的概念模型。图2抽象出复杂的智能家居应用场景，这将有助于开发人员更好地理解和开发智能家居场景应用。

定义1：智能家居场景的定义：



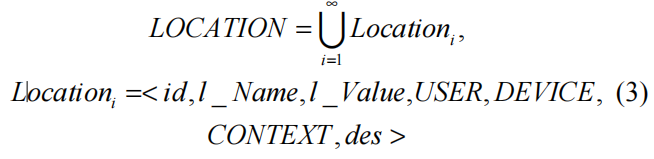
其中 SHScenarioi 表示智能家居应用场景编号为 i 。USER 代表所有的集合场景中的用户，LOCATION 代表一个集合场景中的所有位置，CONTEXT 代表一个场景中所有位置的集合，DEVICE代表场景中所有设备的集合，FUNCTION 表示所有函数的集合场景中，SERVICE 代表所有服务的集合场景。

定义2：智能家居场景中的用户定义：



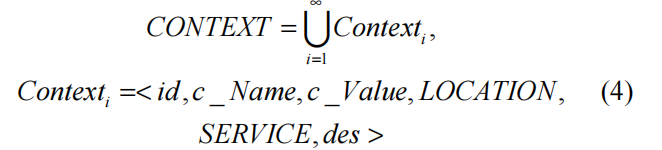
其中 Useri 表示智能家居中的用户应用场景编号为 i 。 id 代表用户的唯一标识符，\_ u Name 代表 Useri 的名称，LOCATION 代表user当前所在位置的集合，des是用户的一组描述。

定义3：智能家居的场景的位置定义：



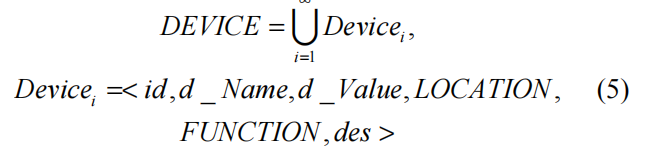
其中 Locationi 表示智能家居中应用场景的位置编号为 i 。 id 代表位置的唯一标识符，\_l Name 表示位置的名称Location , l V\_alue 表示Location的值，l V\_alue 定义为三元组。 USER 代表一个当前位置所有用户的集合，DEVICE代表当前位置所有设备的集合，CONTEXT 表示所有上下文的集合当前位置，设计一组该位置的描述。

定义4：智能家居应用中的上下文场景定义为：



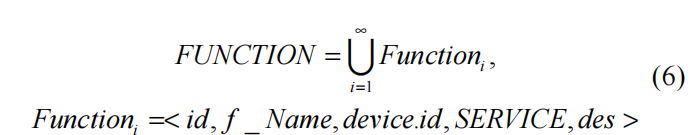
其中 Contexti 表示智能家居中的上下文应用场景编号为 i 。 id 代表上下文的唯一标识符，\_c Name 代表的名称, c V\_alue 表示 Contexti 的值，LOCATION 表示当前上下文位置的集合。 SERVICE 代表一组可以监视和控制上下文的服务，des 是一个一组上下文的描述，并建立上下文和服务之间的关系。

定义5：智能家居应用场景的设备定义：



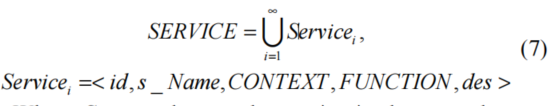
其中 Device 表示智能家居中的设备应用场景编号为 i 。 id 代表设备的唯一标识符，\_d Name 代表Device的名称, LOCATION 表示要其中 Device的位置，des 描述了设备的所有功能函数 id ，将通过它建立设备和设备之间的关系功能。

定义6：智能家居场景的功能定义：



其中Functioni表示智能家居中的功能应用场景编号为 i 。 id 代表函数的唯一标识符，f \_ Name 代表名称的功能。设备 id 表示设备的 id Functioni 属于哪个。 SERVICE 代表一组当前功能可以提供的服务，des表示一组函数的描述。 这函数和服务之间的关系将是通过它建立，同时关系函数和设备之间将建立。

定义7：智能家居应用场景中的设备的定义：



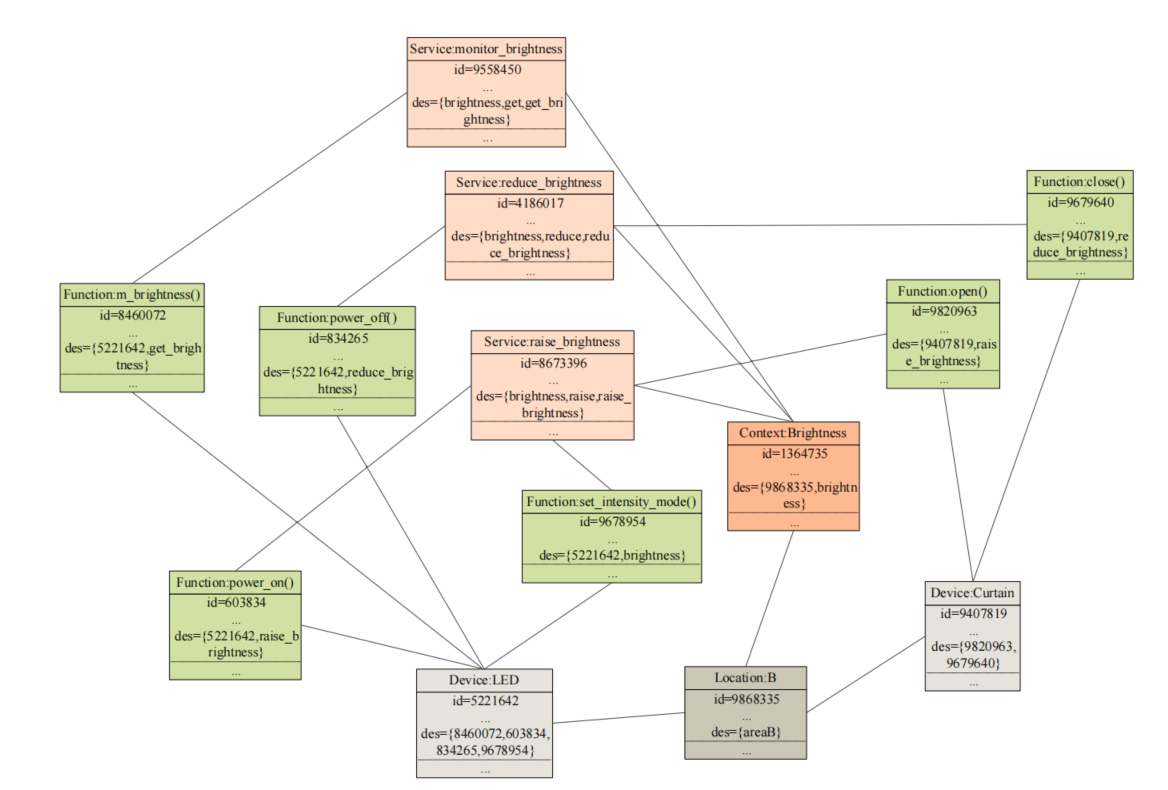
其中Servicei表示智能家居中的服务应用场景编号为 i。id代表服务的唯一标识符，s \_ Name 代表的名称Servicei , CONTEXT 表示当前服务可以监控和控制的上下文的集合。 功能代表所有功能的集合，可以提供当前服务，des代表一组描述服务。将通过它建立函数和函数之间的关系服务，同时服务和上下文之间的关系是确立的。

我们使用OWL2.0构建智能家居的概念模型场景。我们定义的概念模型是智能家居场景，这是基于运行时知识图构建的智能家居场景。

1. 运行知识图谱的构建

智能家居场景中的设备类型及编程接口复杂。 我们需要统一的方式管理底层设备。 我们构造运行智能设备模型，以便对它们进行统一管理方式。 这在 SM@RT 的帮助下很容易完成。

SM@RT（supporting models at run time）是一个模型驱动的框架，支持基于模型的运行时系统开发。 运行时的构建过程的知识图谱如下：第一步：建立对象和关系，根据对象配置文件。第二步：使用SM@RT获取和更新设备和用户的运行时模型的信息。第三步:通过对象信息更新对象之间的关系；跳到第 二步。



图四：通过对象之间的属性描述建立关系

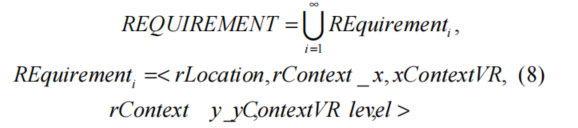
如图2所示，设备层和运行时知识图层是双向的。同步的引擎不仅可以使智能设备的状态由运行的模型来监控，而且也能够对要应用于智能设备的运行时模型的任何更改。这样我们就可以通过对设备的操作来管理设备运行时模型，这些操作将影响到底层系统。

如图4所示，有一个服务（id=4186017）可以降低光照强度，其描述属性为 des={亮度，reduce\_brightness}。有一个上下文(id=4186017) 关于亮度及其描述属性是des={9868335ˈbrightness}。同元素{brightness}在两个对象的 des 中。因此，我们建立服务 (id=4186017) 和上下文 (id=4186017)。

基于设备运行时模型和属性des，我们建立用户、位置、服务、功能、设备和上下文之间的关系，最后构建智能家居场景的运行时知识图谱。

c. 代码生成的方式

定义8:在智能家居应用的场景下的需求被定义：



其中 REquirementi 表示智能中的要求家庭应用场景编号为 i 。每个要求可由场景开发者管理。 r位置表示要控制的位置，\_rContext x表示要监控的上下文，xContextVR表示 \_ rContext x 的 \_ c 值的范围。\_ rContext y 表示要控制的上下文，yContextVR 表示 rContext \_ y 的 \_c 值的范围。

用户可以编写需求配置文件，如下所示图 5. 我们解析需求配置文件并自动生成相应的应用程序代码。自动生成的代码可以直接在之前构建的运行时知识图谱上运行，运行时模型可以与底层设备同步，并且最终开发应用程序。图 6 显示了一个为 QVT 建模自动生成的模板代码。

根据需求配置文件，代码生成器找到需要控制的运行时知识图谱的上下文对象并可以通过运行知识图谱的模型来搜索函数集。这需求文件的参数和通过搜索被填入模板代码的函数，最后得到生成的代码可用于开发智能家居应用程序。

逻辑建模程序根据用户需求自动生成，应用于运行时知识图层。在我们的方法中，领域专家或开发人员只需要聚焦智能家居场景需求，那么应用程序的逻辑开发可以是自动生成。

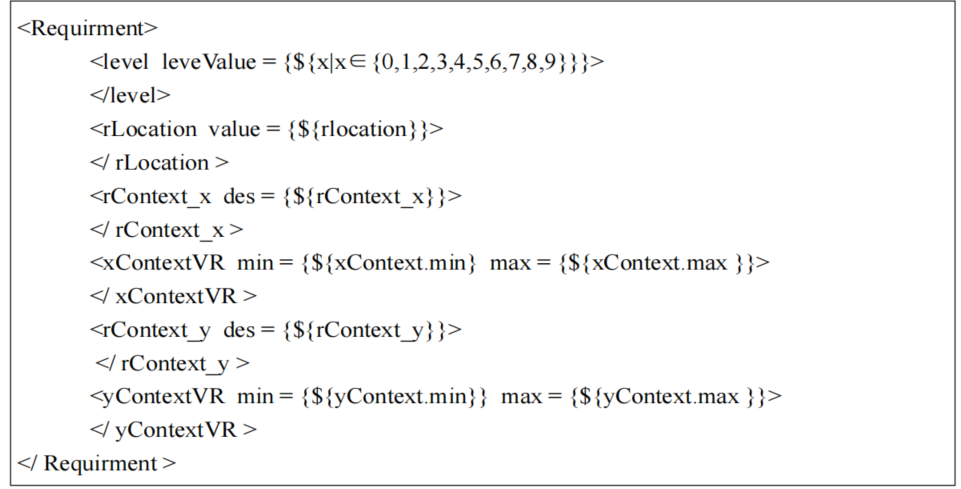


图5：需求的配置文件模板

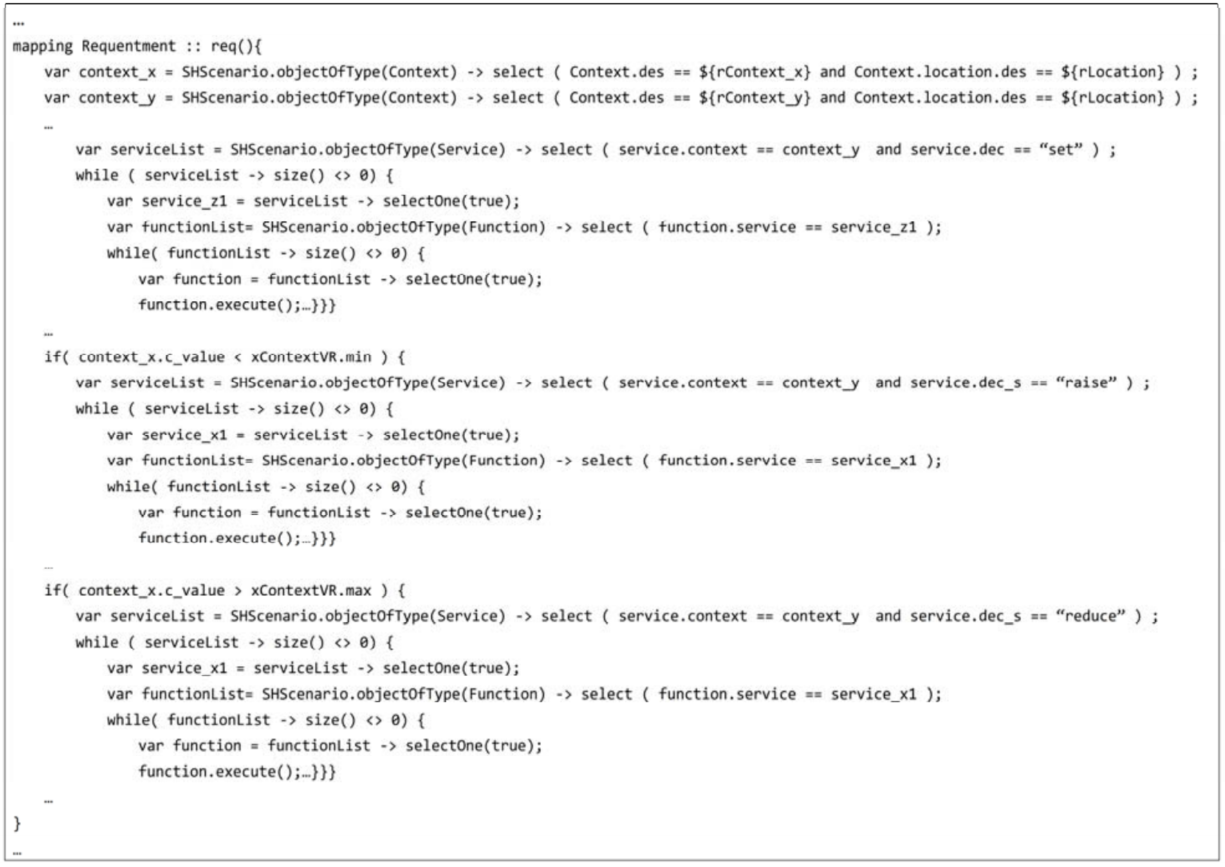


图6：自动生成的代码的QVT模板

五、案例

智能种植系统在物联网领域发挥重要作用。然而，由于智能设备和应用场景的多样性，智能种植的领域应用开发成本非常高。为了验证我们方法的可行性和效率，我们基于运行时的知识图构建智能种植系统。

1. 智能种植场景的元模型

以图 1中的智能种植场景为例。例如，我们根据4.1 节中定义的模型的概念描述场景。 图 7 显示了场景的元模型。 从图 7 中我们可以看到有四种类型和三种类型的上下文场景，对于每种类型的上下文，都有一些相应类型的服务。

然而设备、功能、上下文和服务尚不能客观地实时反映当前的情况。我们也需要合并设备和功能，以及它们之间的关系。

1. 运行时知识图谱的构建

对于给定的元模型和给定的管理集接口，SM@RT [12] 可以自动生成代码，并将模型映射到具有足够运行时间的接口。为了确保设备和功能在知识图谱正确同步，使用设备运行时模型，我们定义映射模型和运行时知识之间的关系如图 8 所示。

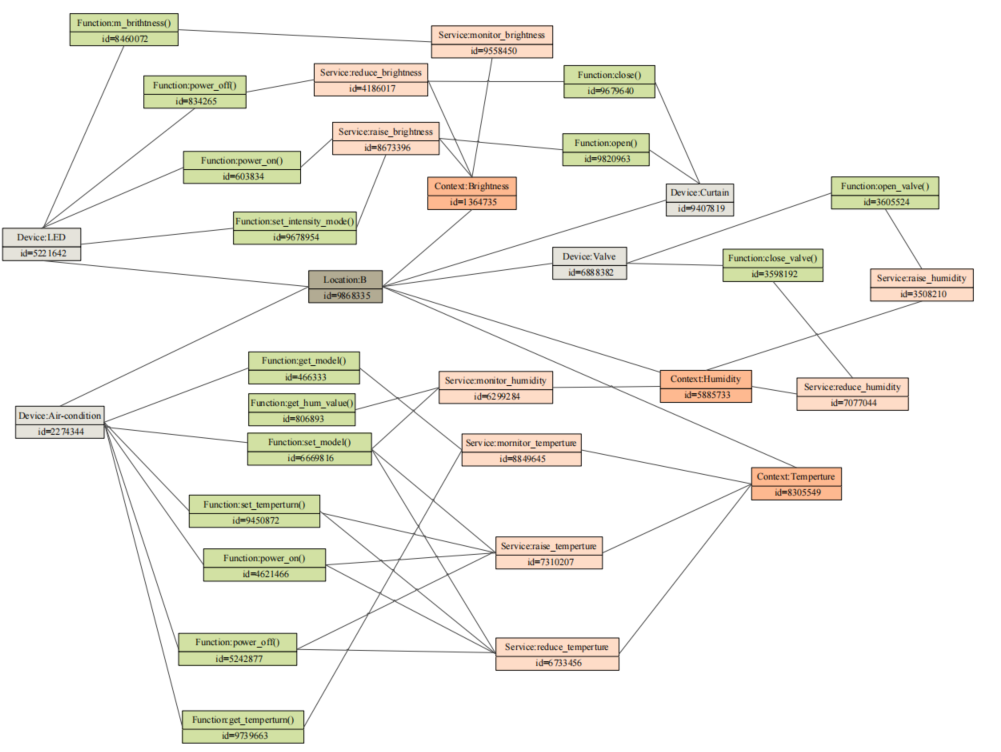
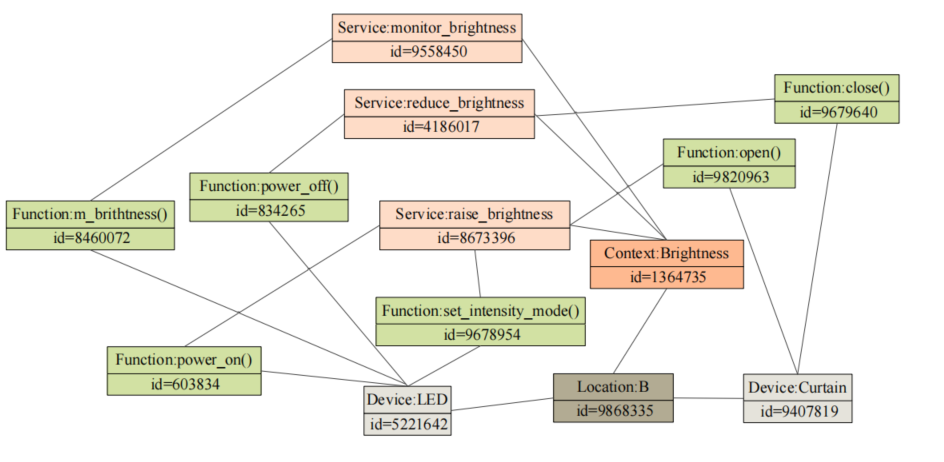
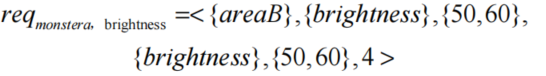


图8.B区域的运行时知识图谱



1. 代码的自动生成

比如在B区种植龟背竹，需要繁殖时的光照强度如下：



根据req，我们在图8中搜索，然后结果如图9所示。这个过程如下：第一步：搜索location.des={areaB}所在的位置，然后获取位置（id=5221642）作为回报。第 2 步：搜索 location.context where 和 context.des={brightness}，然后获取上下文（ id=1364735 ）为返回。第 3 步：搜索 context.service where service.des = {raise}，获取服务（ id=8763396 ）作为回报。第四步：搜索service.function，得到两个函数（id=603834, id=9820963 ) 作为回报。根据图 6 中的 QVT 模板，一个应用程序生成代码以作用于运行时知识图图 8 最后，当光强小于 50 时，打开智能LED灯，拉开窗帘增加光照强度。

D．评价

从三个方面评价我们的方法：

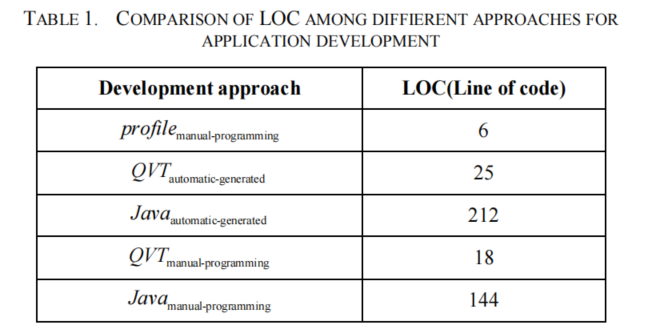
1) 构建基于运行时知识图谱的智能物联网系统

用于构建智能设备的运行时知识图谱，开发人员只需要定义这些对象之间的关系，定义基于架构的元模型和Eclipse 建模框架 (EMF) 上的访问模型。所有的应用逻辑都可以通过在运行时的知识图谱上运行程序，无需处理不同类型的底层管理。我们开发了基于运行时知识图谱的智能家居系统的种植系统为如图 8 所示。

运行时的每个模型元素知识图代表现实世界中的一个实体。管理员可以在架构上管理智能设备和运行时知识图谱上的操作是转化为管理接口的调用底层系统。此外，所有应用的程序逻辑都可以通过在运行时执行程序来执行知识图谱，这可能受益于许多以模型为中心的分析或规划方法和机制，如模型跳棋。

在运行时构建我们的知识图谱的过程我们的方法的时间是一次性的，这个时间成本是可以接受。运行时知识图可以重用于不同的智能家居应用场景。

2）对比管理接口和使用基于架构的运行时知识图谱对底层 API的编程难度，重用数据访问代码。然后，开发人员可以专注于管理任务逻辑，而无需处理不同类型的底层系统的接口。



举一个关于智能种植系统的亮度的的例子。 如表 1 所示，配置文件代码是智能亮度系统只是 6 LOC，生成的 QVT 代码大约是 25 LOC，Java 代码生成大约 212 LOC。 相比之下，QVT 代码手工写的是18 LOC，手工写的Java代码是 144 LOC。 但总的来说，我们的方法可以将 LOC 减少约 85%。

六、结论和未来的工作

智能家居应用开发通常需要开发人员处理底层的系统问题，了解场景并处理复杂和变化的需要。本文提出了一种基于运行时知识图谱的智能家居应用开发方法。开发人员只需要根据概念模型和构建基于运行时知识图谱的智能设备。开发人员可以管理需求的场景。需求文件自动生成建模语言，这些建模语言是直接跑在运行时的知识图谱上。关于运行时知识图谱的代码自动映射到智能设备和所有的应用逻辑可以通过执行在运行时知识图谱上生成的代码来执行。因此，开发人员可以专注于管理逻辑的核心。我们的方法可以帮助开发人员处理底层 API，复杂的应用逻辑和收集到的底层数据，大大减少了手工编码的工作量。

未来的工作，我们计划为了开发人员改进对智能设备的编程。我们计划执行进一步分析，例如在运行时知识图谱和底层系统之间，检查模型以确保更深入

生成的因果关系的正确性和完整性。而且，我们计划借助模型技术添加更多高级功能来简化开发任务。

参考文献

[1] D. C. Schmidt, “Model-Driven Engineering,” Computer Languages

Systems and Structures vol. 43, 2006, pp. 139-155.

[2] S. Sicard, F. Boye and N. D. Palma, “Using components for

architecturebased management: the self-repair case,” International

Conference on Software Engineering, ACM, May. 2008, pp. 101-110,

doi: 10.1145/1368088.1368103.

[3] B. Morin, O. Barais, G. Nain and J. Jezequel, “Taming dynamically

adaptive systems using models and aspects,” IEEE International

Conference on Software Engineering, IEEE, May. 2009, pp. 122-132,

doi: 10.1109/ICSE.2009.5070514.

[4] J. Yang, G. Huang, W. H. Zhu, X. F. Cui and H. Mei, “Quality

attribute trade-o­ through adaptive architectures at runtime,” Journal

of Systems and Software, vol. 82, 2009, pp. 319-332, doi:

10.1016/j.jss.2008.06.039.

[5] H. Mei, G. Huang, L. Lan and J. G. Li, “A software architecture

centric selfadaptation approach for internet-ware,” Science in China,

vol. 51, 2008, pp. 722–742, ISSN: 1009-2757.

[6] S. Nakamura, S. Shigaki, A. Hiromori, H. Yamaguchi and T.

Higashino, “A model-based approach to support smart and social

home living,” ACM International Joint Conference on Pervasive and

Ubiquitous Computing, ACM, Sept. 2015, pp. 1101-1105, doi:

10.1145/2750858.2805835.

[7] D. Lüddecke, N. Bergmann and I. Schaefer, “Ontology-Based

Modeling of Context-Aware Systems,” Model-Driven Engineering

Languages and Systems, Springer International Publishing, LNCS,

vol. 8767, 2014, pp.484-500, doi: 10.1007/978-3-319-11653-2\_30.

[8] D. Ernadote, “Ontology-Based Pattern for System Engineering,” 2017

ACM/IEEE 20th International Conference on Model Driven

Engineering Languages and Systems (MODELS). IEEE, Sept. 2017,

pp. 248-258. doi:10.1109/MODELS.2017.4.

[9] D. Ernadote, “An ontology mindset for system engineering,” IEEE

International Symposium on Systems Engineering, IEEE, Sept. 2015,

pp. 454-460, doi: 10.1109/SysEng.2015.7302797.

[10] B. Iordanov, A. Alexandrova, S. Abbas, T. Hilpold and P. Upadrasta,

“The Semantic Web as a Software Modeling Tool: An Application to

Citizen Relationship Management,” Model-Driven Engineering

Languages and Systems, Springer Berlin Heidelberg, LNCS, vol.

8107, 2013, pp. 589-603, doi: 10.1007/978-3-642-41533-3\_36.

[11] X. Chen, A. Li, X. Zeng, W. Guo and G. Huang, “Runtime model

based approach to iot application development,” Frontiers of

Computer Science, vol. 4, 2015, pp. 540-553, doi: 10.1007/s11704-

015-4362-0.

[12] B. Motik, “Representing and Querying Validity Time in RDF and

OWL: A Logic-Based Approach,” International Semantic Web

Conference, Springer Berlin Heidelberg, LNCS, vol. 6496, 2013, pp.

550-565, doi: 10.1007/978-3-642-17746-0\_35.

[13] H. Song, et al, “Supporting runtime software architecture: a

bidirectional-transformation-based approach,” Journal of Systems and

Software, vol. 84, May. 2011, pp. 711-723, doi:

10.1016/j.jss.2010.12.009.

[14] J. Li, X. Chen, G. Huang, H. Mei and Chauvel, F, “Selecting fault

tolerant styles for third-party components with model checking

support,” Lecture Notes in Computer Science, vol. 5582, 2009, pp.

69-86, doi: 10.1007/978-3-642-02414-6\_5.

[15] X. Chen, G. Huang, F. Chauvel, Y. Sun and H. Mei, “A framework

for the integration of MOF-compliant analysis methods,” Asia-Pacific

Symposium on Internetware, ACM, Nov. 2010, pp. 1, doi:

10.1145/2020723.2020724.

[16] O. M. Group, Meta object facility (mof) 2.0 core specification.

http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-15.pdf, 2013.