Исследование подходов к разработке умных объектов

А.Г. Клименко, К.С. Зайцев

Аннотация **– Разрабатывать умные объекты можно по-разному. Целью настоящей работы является исследование преимуществ и недостатков основных подходов к разработке умных объектов. Последовательно анализируются особенности таких подходов, как модельно-ориентированный; подход на основе mashup; подход «функция как услуга»; программирование с ориентацией на узлы; на основе использования баз данных; на основе макропрограммирования. При рассмотрении модельно-ориентированного подхода анализируется связь между описанием системы на высоком уровне абстракции и готовым приложением IoT. При создании mashup-приложений рассматриваются разработка вручную и с помощью инструментальных средств. В подходе «Функция как услуга» выделяется его способности к интеграции с облачными базами данных, службами аутентификации и авторизации, службами обмена сообщениями и т.д. В подходе, основанном на макропрограммировании, фиксируется требование наличия методологии разработки полного жизненного цикла создания приложений. Результаты сравнения характеристик приведенных подходов, полученные авторами, оформлены в виде таблицы преимуществ и недостатков. Сравнение подходов позволяет сделать вывод о том, что для разработки умных объектов можно использовать различные подходы, которые определяются целями разработки, бюджетом и наличием специальных навыков разработчиков.**

Ключевые слова **– умный объект, интернет вещей, моделирование умных объектов, mashup-приложения, макропрограммирование.**

I. ВВЕДЕНИЕ

Внедрение цифровых технологий влечет за собой широкий спектр умных решений и умных объектов, которые применяются практически во всех отраслях экономики и социальной сферы [1]. Некоторые из этих решений известны давно, например, 3D проектирование зданий (Building Information Modelling, BIM) или промышленные роботы.

Другие, такие как умные счетчики для сбора данных о потреблении воды и электроэнергии; умные системы распределения ресурсов в топливно-энергетическом комплексе, умные дома и др. разрабатываются на наших глазах.

Результаты экспертных опросов показывают неравномерное развитие сегодняшнего спроса на умные объекты в разных секторах экономики и социальной сферы. Новый всплеск ускоренного создания и вывода на рынок умных устройств будет связан с комбинированием в них компонентов различных технологических направлений, таких как ИИ, Интернет вещей, беспроводная связь, сенсорика и др. Ежегодный прирост этого рынка с 2020 по 2026 г. оценивают ориентировочно в 58% [2].

Для надежной и комфортной разработки новых решений для умных объектов, комбинирующих различные технологии, нужно следовать определенной методике, т.е. последовательности действий, связанных с выбором компонентов. Классические подходы при реализации подобных проектов, как известно, начинают давать сбои по многим вопросам [3]. Например, при пассивном сборе данных, система управления зданием или система для автоматического контроля и сбора информации, которые запрашивают данные у программируемого логического контроллера, плохо сочетаются с особенностями мобильной сети. Также возникают проблемы с преобразованием сетевых адресов или нестабильными соединениями.

Поэтому приходится использовать специальные подходы, ориентированные на разработку умных объектов. Анализу таких подходов и посвящена настоящая работа.

II. ПОДХОДЫ К РАЗРАБОТКЕ

2.1. Модельно-ориентированный подход

Модельно-ориентированный подход, как следует из названия, основан на идее описания создаваемой системы на более высоком уровне абстракции [4]. Это можно сделать в UML и/или других языках с помощью диаграмм, моделирующих определенные аспекты или представления системы. Затем по полученным диаграммам генерируется код для разных платформ с учетом разных API.

На рис. 1 представлены ключевые элементы модельно-ориентированного подхода, а именно связь между описанием системы на высоком уровне абстракции и готовым приложением IoT.

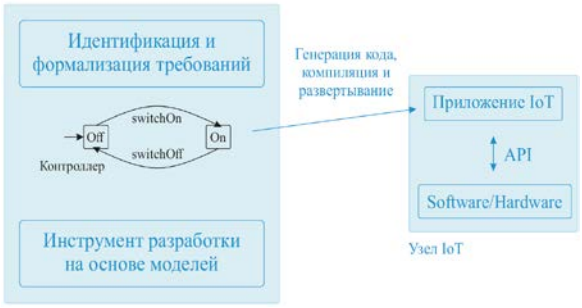

Рис. 1. Ключевые элементы модельно-ориентированного подхода.

Описание системы на высоком уровне абстракции, как правило, проводится в 2 этапа [5]:

1. Идентификация требований – сбор информации о функциональных возможностях системы и выявление нефункциональных требований. Этот процесс является итеративным, и функциональные требования могут быть представлены с помощью диаграмм UML, описывающих функции системы IoT.

2. Формализация требований – структурирование требований и разработка моделей описания взаимосвязи системных процессов. На этом этапе можно выделить подуровни с конкретными задачами, связанными с проектированием систем IoT, такими как координация, интеграция, управление большими данными и т. д.

При описании системы применяется базовый принцип разделения интересов как по вертикали, так и по горизонтали. Вертикальное разделение снижает сложность разработки приложений за счет абстракции, а горизонтальное разделение задач уменьшает ее за счет описания системы с использованием разных системных представлений, каждое из которых описывает определенный аспект системы [6]. Преимущества подхода исходят из основной идеи о том, что дробление системы на различные задачи в процессе разработки приложений может повысить производительность.

Выделяют два достаточно широких класса моделей [7]:

• модели разработки: это модели программного обеспечения на уровнях абстракции выше уровня кода. Примерами моделей разработки являются модели требований, архитектуры, реализации и развертывания. Исследования, как правило, сосредоточены на создании и использовании этих моделей.

• runtime-модели: эти модели представляют некоторые аспекты работающей системы и, таким образом, являются абстракциями runtime-явлений. Многие исследователи начали изучать, как эти модели могут использоваться для поддержки динамической адаптации программных систем.

Основным преимуществом подхода на основе моделей является независимое от платформы моделирование, которое позволяет генерировать код для различных конкретных платформ IoT. Существует множество инструментов генерации кода для разных языков программирования [8]. Вот некоторые из наиболее известных: Papyrus для C++ и Java [9], ThingML для C, C++, Java и JavaScript [10], Visual Paradigm для 17 языков, включая C#, Java, Python, PERL и Ruby.

Генераторы кода создают сопоставления между определенными структурами диаграмм и строками кода. Способность этих инструментов переводить одну структуру в другую варьируется; некоторые из них создают только скелетный код, в то время как другие могут соответствовать почти любой структуре.

Основным недостатком представленного подхода к разработке является то, что он занимает много времени [11]. Не только из-за множества уровней абстракции, на которых следует описывать системы, но и из-за детализации структур, которые можно использовать повторно, т. е. классов и фрагментов кода [12].

2.2. На основе mashup

При использовании этого подхода система разрабатывается путем объединения существующих сервисов, обычно с использованием знакомых инструментов и подходов веб-разработки (например, прототипирования). Таким образом, эта стратегия часто применима для разработки персонализированных, настраиваемых, недолговечных и некритичных для бизнеса приложений.

Можно выделить основные этапы методологии [5]:

1. Подбор инструментов с учетом цели проекта, когда предполагается, что хорошее решение должно:

• использовать наиболее распространенные платформы;

• иметь открытый исходный код и опираться на открытые стандарты;

• развертываться локально;

• поддерживать нескольких языков программирования.

2. Выбор платформы для интеграции между сервисами и удаленными платформами.
3. Стандартизация разработки.
4. Обработка данных и манипулирование ими.

Инструменты mashup обычно предоставляют графический редактор для взаимосвязи сервисов в одном приложении [6], а также для моделирования потока сообщений между компонентами (это могут быть датчики, шлюзы IoT, внешние веб-сервисы и проч.). В этом контексте методологию мэшапов можно рассматривать как конкретные формы программирования для конечного пользователя, ограниченные определенной моделью.

Инструменты mashup позволяют выполнять очень быстрое прототипирование и использовать преобразование, трансформацию и объединение данных из одного или нескольких сервисов для достижения целей проекта. Они также позволяют подключать различные сервисы для создания новых процессов.

Как упоминалось в [13], mashup-инструменты могут извлечь выгоду из концепций подходов, основанных на моделях. Кроме того, подходы, основанные на моделях, могут лучше соответствовать IoT и предоставлять простые в использовании инструменты.

Mashup-приложения представляют собой наборы существующих ресурсов, которые взаимодействуют друг с другом для доставки нового контента с сокращением времени производства, сложности и стоимости. Разработчики ищут все существующие ресурсы, которые можно интегрировать для создания нового приложения. Важно отметить, что гибридные приложения не обязательно являются конечными продуктами и не всегда имеют пользовательский интерфейс [14]. Компоненты представлены в виде черных ящиков с входами и выходами, указанными в API. Разработчикам может потребоваться встроить некоторые функции в новый блок или выполнить адаптацию выбранных API. После подключения компонентов приложение можно протестировать и открыть для использования. Затем новые приложения могут быть построены поверх этого, и цикл разработки может начаться снова. Этот процесс описан Майклом Огринцем в [14] как круг мэшапов (рис. 2).

Разработка mashup-приложений может осуществляться вручную или с помощью инструментов [15, 16]. Ручной процесс требует значительного уровня знаний о задействованных технологиях. Это связано с тем, что mashup-приложения могут предоставлять разные форматы для извлечения данных. Следовательно, их необходимо будет проанализировать и преобразовать. Кроме того, если требуется пользовательский интерфейс, разработчик должен заботиться о его функциональности.

Рис. 2. Круг мэшапов, описанный Майклом Огринцем [14].

Сегодня mashup-инструменты предлагают различные схемы программирования и уровни автоматизации в соответствии с целевой аудиторией. Конечным продуктом может быть неисполняемый дизайн, прототип с фиктивными элементами или развертываемое приложение в зависимости от инструмента [9].

Типичные функции, поддерживаемые mashup-инструментами:

• создание, настройка и подключение узлов в удобном графическом редакторе;
• сбор, объединение и преобразование данных;
• создание сценариев, кодирование.

Платформы IoT обычно предлагают расширенный набор функций, таких как предметно-ориентированный язык и средства кодирования для поддержки процесса разработки.

2.3. «Функция как услуга»

Авторами статьи [11] был выделен подход к разработке умных систем, основанный на использовании бессерверных платформ «Функция как услуга» (Function-as-a-Service, FaaS). Вместо того, чтобы разрабатывать логику приложения в виде сервисов и управлять необходимыми ресурсами, разработчик приложения реализует детализированные функции, связанные в управляемом событиями приложении, и развертывает их на платформе FaaS. Платформа отвечает за предоставление ресурсов для вызова

функций и выполняет автоматическое масштабирование в зависимости от рабочей нагрузки.

Функции могут быть тесно интегрированы с другими службами, например, облачными базами данных, службами аутентификации и авторизации, службами обмена сообщениями. Эти услуги иногда называют Backend-as-a-Service (BaaS) – это сторонние сервисы, которые позволяют пользователям сосредоточиться только на логике приложения. Особенности платформ BaaS описаны в [17], а в [18] представлена модель облачных вычислений, реализованная за счет «бессерверных» архитектур и поддерживающих их платформ. Разработчики запускают код приложения событиями или HTTP-запросами и управляют им с помощью функций. Небольшие блоки кода развертываются в FaaS и выполняются по мере необходимости как отдельные действия.

На рис. 3 показаны ключевые элементы решения FaaS:

Рис. 3. Ключевые элементы решения Function-as-a-Service.

• источники событий – запуск или передача событий в один или несколько экземпляров функции;

• экземпляры функций – одна функция/микрослужба, которую можно масштабировать по мере необходимости;

• контроллер FaaS – развертывание, управление и мониторинг экземпляров функций и их источников;

• платформенные службы – общие кластерные или облачные службы, используемые решением FaaS (иногда называемые BaaS).

Бессерверный продукт FaaS может практически мгновенно и точно масштабироваться для обработки каждого отдельного входящего запроса. Бессерверные платформы не имеют понятия «заранее запланированная мощность». Масштабирование происходит автоматически без вмешательства разработчика и не требует настройки триггеров или правил «автоматического масштабирования». По завершении обработки запроса бессерверная система FaaS автоматически уменьшает вычислительные ресурсы.

Состояние приложения хранится в базах данных. Создание безопасного, масштабируемого, производительного и управляемого приложения для системы IoT кажется сложной задачей, однако наблюдается рост разработки для запуска функций, основанных на FaaS, для устройств IoT.

Этот подход имеет ряд преимуществ:

• снижение эксплуатационных расходов и расходов на системное администрирование;

• снижение затрат на разработку и развертывание, обеспечение более быстрого выхода на рынок;

• высокая масштабируемость и отказоустойчивость.

Однако этот подход ограничен отдельными приложениями и статическим распределением вычислений. Интеграция систем IoT для общих приложений FaaS потребует расширения платформы FaaS на гетерогенные устройства.

2.4. С ориентацией на узлы

Подход к программированию, ориентированному на узлы [19] позволяет разрабатывать системы, основанные на полном контроле над отдельными устройствами.

Разработчики системы пишут программу, которая считывает данные с датчиков соответствующих устройств. Программа связывает узлы между собой и выдает им команды [4], а также собирает данные, относящиеся к некоторым внешним событиям, решает, куда их отправить, и при необходимости связывается с исполнительными механизмами по идентификатору или местоположению [7].

При разработке используются так называемые языки программирования общего назначения (General Purpose Language), например, C, C++, Java, и конкретный API промежуточного программного обеспечения или службы уровня узла.

Однако подход нелегко использовать для приложений IoT из-за большого количества разнородных устройств [20].

2.5. На основе использования баз данных

Этот подход устраняет ограничения ориентированного на узлы программирования и рассматривает всю сеть датчиков как виртуальную систему баз данных [7]. Он

предоставляет фреймворк с простым в использовании интерфейсом, который позволяет заинтересованным сторонам отправлять запросы к сети датчиков в SQL-подобном виде или через API на шлюзе.

Однако эти системы в значительной степени не обеспечивают гибкости для внедрения логики приложения. Например, заинтересованным сторонам требуются значительные изменения в синтаксическом анализаторе TinyDB и механизме запросов для реализации новых операторов запросов. Модель, основанная на использовании баз данных, не поддерживает логику приложения на этом уровне, что делает ее малопригодной для разработки полноценных приложений IoT [4].

2.6. На основе макропрограммирования

Этот подход преодолевает ограничения подхода с использованием баз данных, предоставляя возможность указать пользовательскую логику приложения. Здесь предусмотрены абстракции для возможности задать высокоуровневое поведение при совместной работе, при этом скрывая от заинтересованных сторон низкоуровневые детали, такие как передача сообщений или мониторинг состояния. Разработчики описывают свое приложение, используя эти абстракции, которые затем компилируются в код уровня узла.

Для устранения ограничений программирования, ориентированных на узлы, были предложены различные системы макропрограммирования, например, [21]. Основные преимущества подхода исходят из идеи о том, что путем разделения различных задач системы на определенном уровне абстракции и предоставления механизмов преобразования этих абстракций в целевой код производительность (например, возможность повторного использования, ремонтопригодность) в процессе разработки приложений может повыситься.

Макропрограммирование является жизнеспособным подходом по сравнению с программированием, ориентированным на узлы, и подходом, основанным на использовании баз данных. Однако отсутствие сложившейся методологии разработки программного обеспечения для поддержки всего жизненного цикла создания приложений обычно приводит к очень сложным в обслуживании, повторному использованию и платформенно-зависимому проектированию, с которыми можно справиться с помощью подхода, основанного на моделях. При этом большинство систем макропрограммирования в основном сосредоточены на этапе разработки, игнорируя тот факт, что он представляет собой лишь малую часть жизненного цикла создания приложения.

III. РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ

В настоящем исследовании был проведен анализ существующих подходов к разработке умных систем, и рассмотрены их основные особенности. Резюмируя результаты исследования, преимущества и недостатки рассмотренных подходов к разработке умных систем можно представить в виде таблицы 1.

Таблица 1. Преимущества и недостатки подходов к разработке умных объектов.

№	Название подхода	Преимущества	Недостатки
1	Модельно-ориентированный подход	Независимое от платформы моделирование. Возможность генерации кода. Разнообразие инструментов для разных языков программирования. Дробление системы на различные задачи может повысить производительность.	Трудоемкий процесс, который занимает много времени из-за детализации структур.
2	На основе mashup	Используются знакомые, простые в использовании инструменты и подходы к веб-разработке. Разработка персонализированных и настраиваемых приложений. Возможность выполнять очень быстрое прототипирование.	Разработка недолговечных и некритичных для бизнеса приложений. Ручной процесс разработки требует значительного уровня знаний о задействованных технологиях.
3	«Функция как услуга»	Мгновенное, точное, автоматическое масштабирование приложений для обработки каждого отдельного входящего запроса, отказоустойчивость. Снижение эксплуатационных расходов и расходов на системное администрирование. Снижение затрат на разработку и развертывание,	Подход ограничен отдельными приложениями и статическим распределением вычислений. Интеграция систем IoT потребует расширения платформы на гетерогенные устройства.

		обеспечение более быстрого выхода на рынок.	
4	С ориентацией на узлы	Полный контроль над отдельными устройствами.	Сложно использовать для приложений IoT из-за большого количества разнородных устройств.
5	На основе использования баз данных	Вся сеть датчиков рассматривается как виртуальная система баз данных.	Отсутствует возможность указать пользовательскую логику приложения.
6	На основе макропрограммирования	Возможность указать пользовательскую логику приложения. Повышенная производительность за счет разделения различных задач системы на определенном уровне абстракции и предоставления механизмов преобразования этих абстракций в целевой код.	Отсутствие методологии разработки ПО для поддержки всего ЖЦ создания приложений. Сложное в обслуживании, повторном использовании и платформенно-зависимое проектирование. Большинство систем в основном сосредоточены на этапе разработки.

IV. ЗАКЛЮЧЕНИЕ

В статье, выполненной в рамках выпускной квалификационной работы магистра, проводилось исследование преимуществ и недостатков основных подходов к разработке умных объектов. Последовательно проанализированы особенности таких подходов, как модельно-ориентированный; подход на основе mashup; функция как услуга; программирование, ориентированное на узлы; на основе использования баз данных; на основе макропрограммирования.

В результате исследования выявлено, что:

1) при модельно-ориентированном подходе описание абстрактной системы проводится в 2 этапа: сначала идентификации требований (часто с помощью диаграмм UML) и потом формализации требований, т.е. разработки моделей описания взаимосвязи системных процессов;

2) при использовании подхода на основе mashup система разрабатывается путем объединения существующих сервисов, обычно с использованием знакомых инструментов и подходов веб-разработки (например, прототипирования);

3) при использовании подхода «Функция как услуга» вместо того, чтобы разрабатывать логику приложения в виде сервисов и управлять необходимыми ресурсами, разработчику приложения необходимо реализовать детализированные функции, связанные в управляемом событиями приложении, и развернуть их на платформе FaaS;

4) при использовании подхода основе макропрограммирования требуется методология разработки программного обеспечения для поддержки всего жизненного цикла создания приложений, отсутствие которой приводит к сложному обслуживанию и платформенно-зависимому проектированию.

Подводя итог, можно сказать, что для разработки различных умных объектов можно использовать различные подходы, руководствуясь целями разработки, бюджетом и наличием специальных навыков разработчиков.

БЛАГОДАРНОСТИ

Авторы выражают благодарность Высшей инжиниринговой школе НИЯУ МИФИ за помощь в возможности опубликовать результаты выполненной работы.

БИБЛИОГРАФИЯ

[1] Цифровая трансформация отраслей: стартовые условия и приоритеты. Доклад ВШЭ, 2021 https://conf.hse.ru/mirror/pubs/share/463148459.pdf

[2] Digital Twin Market by Technology, Type (Product, Process, and System), Application (predictive maintenance), Industry (Aerospace & Defense, Automotive & Transportation, Healthcare), and Geography - Global Forecast to 2026 - Markets and Markets, 2020 https://www.marketsandmarkets.com/Market-Reports/digital-twin-market-225269522.html?gclid=EAIaIQobChMIsuDJh5ei9gIVGhF7Ch1ZCgAXEAAYASAAEgJsu_D_BwE .

[3] Романченко П. Умные здания: как внедряют smart-решения в старые здания и какую экономию это дает - Центр инноваций «Инфосистемы Джет», 2020 https://hightech.fm/2020/01/08/smart-buildings

[4] Rajkumar Buyya and A.V. Dastjerdi. Internet of Things: Principles and Paradigms. 05 2016.

[5] Alex Drozd, Oleg Illiashenko, Vyacheslav Kharchenko, Maryna Kolisnyk, Yuriy Kondratenko, Galyna Kondratenko, Olena Maevska, Oleksande Martynuyk, Denis Mazur, Maksym Nesterov, Anatoliy Plakhtyeyev, Vadym Shkarupilo, Ievgen Sidenko, Inna Skarga-Bandurova, Vladimir Sklyar, Galyna Tabunshchyk, Mykyta Taranov, Artem Velikzhanin, Dmytro Uzun, and Heorhii Zemlianko. Internet of Things for Industry and Human Application. In Volumes 1-3. Volume 2. Modelling and Development. 08 2019.

[6] Christian Prehofer and Luca Chiarabini. From internet of things mashups to model-based development. In 2015 IEEE 39th Annual Computer Software and Applications Conference, volume 3, pages 499–504, 2015.

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. Computer Networks, pages 2787–2805, 10 2010.

[8] Vinay Kulkarni and Sreedhar Reddy. Separation of concerns in model-driven development. Software, IEEE, 20:64 – 69, 10 2003.

[9] Saeed Aghaee, Marcin Nowak, and Cesare Pautasso. Reusable decision space for mashup tool design. In Proceedings of the

4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '12, page 211–220, New York, NY, USA, 2012. Association for Computing Machinery.
[10] Marco Brambilla, Eric Umuhoza, and Roberto Acerbis. Model-driven development of user interfaces for iot systems via domain-specific components and patterns. Journal of Internet Services and Applications, 8, 12 2017.
[11] Saitel Sanabria and Anshul Jindal. The ifs and buts of the development approaches for iot applications. 01 2021.
[12] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. pages 37–54, 06 2007.
[13] Alessio Botta, Walter Donato, Valerio Persico, and Antonio Pescap`e. Integration of cloud computing and internet of things: A survey. Future Generation Computer Systems, 56, 10 2015.
[14] Michael Ogrinz. Mashup patterns: Designs and examples for the modern enterprise. 2009.
[15] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. IEEE Internet Computing, 12(5):44–52, 2008.
[16] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. 11 2015.
[17] Kin Lane. Overview of the backend as a service (baas) space, 2013.
[18] Sarah Allen, Ben Browning, Lee Calcote, Amir Chaudhry, Doug Davis, Louis Fourie, Antonio Gulli, Yaron Haviv, Daniel Krook, Orit Nissan Messing, Chris Munns, Ken Owens, Mark Peek, Cathy Zhang, and Chris A. Cncf wg-serverless whitepaper v1.0. Cloud Native Computing Foundation, 2017.
[19] Kamin Whitehouse, Cory Sharp, David Culler, and Eric Brewer. Hood: A neighborhood abstraction for sensor networks. 01 2004.
[20] Pankesh Patel and D. Cassou. Enabling high-level application development for the internet of things. Journal of Systems and Software, 103:62–84, 01 2015.
[21] Animesh Pathak, Luca Mottola, Amol Bakshi, V. Prasanna, and Gian Picco. A compilation framework for macroprogramming networked sensors. pages 189–204, 06 2007

Статья получена 28 февраля 2022.
А.Г. Клименко, Национальный Исследовательский Ядерный Университет МИФИ, магистрант, (e-mail: Klimenko.an11@gmail.com)
К.С. Зайцев, Национальный Исследовательский Ядерный Университет МИФИ, профессор, (e-mail: KSZajtsev@mephi.ru)

Study of approaches to the development of smart objects

A.G. Klimenko, K.S. Zaytsev

***Abstract*. Smart objects can be developed in different ways. The purpose of this work is to study the advantages and disadvantages of the main approaches to the development of smart objects. The features of such approaches as model-based; mashup-based approach; "function as a service" approach; node-oriented programming; based on the use of databases; based on macro programming. When considering a model-based approach, the relationship between the description of the system at a high level of abstraction and the finished IoT application is analyzed. When creating mashup applications, development by hand and with the help of tools is considered. The Function as a Service approach highlights its ability to integrate with cloud databases, authentication and authorization services, messaging services, and more. In the approach based on macro programming, the requirement for a development methodology for the full life cycle of application creation is fixed. The results of comparing the characteristics of the above approaches, obtained by the authors, are presented in the form of a table of advantages and disadvantages. Comparison of approaches allows us to conclude that different approaches can be used to develop smart objects, which are determined by the development goals, budget, and the availability of special skills of developers.**

***Keywords* - Smart object, Internet of things, Smart object modeling, Mashup applications, Macro programming**

REFERENCES

[1] Digital transformation of industries: starting conditions and priorities. HSE report, 2021, https://conf.hse.ru/mirror/pubs/share/463148459.pdf

[2] Digital Twin Market by Technology, Type (Product, Process, and System), Application (predictive maintenance), Industry (Aerospace & Defense, Automotive & Transportation, Healthcare), and Geography - Global Forecast to 2026 - Markets and Markets, 2020 https://www.marketsandmarkets.com/Market-Reports/digital-twin-market-225269522.html?gclid=EAIaIQobChMIsuDJh5ei9gIVGhF7Ch1ZCgAXEAAYASAAEgJsu_D_BwE .

[3] Romanchenko P. Smart buildings: how smart solutions are implemented in old buildings and what savings it gives - Jet Infosystems Innovation Center, 2020 https://hightech.fm/2020/01/08/smart-buildings

[4] Rajkumar Buyya and A.V. Dastjerdi. Internet of Things: Principles and Paradigms. 05 2016.

[5] Alex Drozd, Oleg Illiashenko, Vyacheslav Kharchenko, Maryna Kolisnyk, Yuriy Kondratenko, Galyna Kondratenko, Olena Maevska, Oleksande Martynuyk, Denis Mazur, Maksym Nesterov, Anatoliy Plakhtyeyev, Vadym Shkarupilo, Ievgen Sidenko, Inna Skarga-Bandurova, Vladimir Sklyar, Galyna Tabunshchyk, Mykyta Taranov, Artem Velikzhanin, Dmytro Uzun, and Heorhii Zemlianko. Internet of Things for Industry and Human Application. In Volumes 1-3. Volume 2. Modelling and Development. 08 2019.

[6] Christian Prehofer and Luca Chiarabini. From internet of things mashups to model-based development. In 2015 IEEE 39th Annual Computer Software and Applications Conference, volume 3, pages 499–504, 2015.

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. Computer Networks, pages 2787–2805, 10 2010.

[8] Vinay Kulkarni and Sreedhar Reddy. Separation of concerns in model-driven development. Software, IEEE, 20:64 – 69, 10 2003.

[9] Saeed Aghaee, Marcin Nowak, and Cesare Pautasso. Reusable decision space for mashup tool design. In Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '12, page 211–220, New York, NY, USA, 2012. Association for Computing Machinery.

[10] Marco Brambilla, Eric Umuhoza, and Roberto Acerbis. Model-driven development of user interfaces for iot systems via domain-specific components and patterns. Journal of Internet Services and Applications, 8, 12 2017.

[11] Saitel Sanabria and Anshul Jindal. The ifs and buts of the development approaches for iot applications. 01 2021.

[12] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. pages 37–54, 06 2007.

[13] Alessio Botta, Walter Donato, Valerio Persico, and Antonio Pescap`e. Integration of cloud computing and internet of things: A survey. Future Generation Computer Systems, 56, 10 2015.

[14] Michael Ogrinz. Mashup patterns: Designs and examples for the modern enterprise. 2009.

[15] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. IEEE Internet Computing, 12(5):44–52, 2008.

[16] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. 11 2015.

[17] Kin Lane. Overview of the backend as a service (baas) space, 2013.

[18] Sarah Allen, Ben Browning, Lee Calcote, Amir Chaudhry, Doug Davis, Louis Fourie, Antonio Gulli, Yaron Haviv, Daniel Krook, Orit Nissan Messing, Chris Munns, Ken Owens, Mark Peek, Cathy Zhang, and Chris A. Cncf wg-serverless whitepaper v1.0. Cloud Native Computing Foundation, 2017.

[19] Kamin Whitehouse, Cory Sharp, David Culler, and Eric Brewer. Hood: A neighborhood abstraction for sensor networks. 01 2004.

[20] Pankesh Patel and D. Cassou. Enabling high-level application development for the internet of things. Journal of Systems and Software, 103:62–84, 01 2015.

[21] Animesh Pathak, Luca Mottola, Amol Bakshi, V. Prasanna, and Gian Picco. A compilation framework for macroprogramming networked sensors. pages 189–204, 06 2007