electronics

Article

An Agent-Based Approach to Realize Emergent Configurations in the Internet of Things

Fahed Alkhabbas [1,2,*], Romina Spalazzese [1,2] and Paul Davidsson [1,2]

1. Internet of Things and People Research Center, Malmö University, 21119 Malmö, Sweden; romina.spalazzese@mau.se (R.S.); paul.davidsson@mau.se (P.D.)
2. Department of Computer Science and Media Technology, Malmö University, 21119 Malmö, Sweden

* Correspondence: fahed.alkhabbas@mau.se

Received: 29 June 2020; Accepted: 17 August 2020; Published: 20 August 2020

Abstract: The Internet of Things (IoT) has enabled physical objects and devices, often referred to as things, to connect and communicate. This has opened up for the development of novel types of services that improve the quality of our daily lives. The dynamicity and uncertainty of IoT environments, including the mobility of users and devices, make it hard to foresee at design time available things and services. Further, users should be able to achieve their goals seamlessly in arbitrary environments. To address these challenges, we exploit Artificial Intelligence (AI) to engineer smart IoT systems that can achieve user goals and cope with the dynamicity and uncertainty of their environments. More specifically, the main contribution of this paper is an approach that leverages the notion of Belief-Desire-Intention agents and Machine Learning (ML) techniques to realize Emergent Configurations (ECs) in the IoT. An EC is an IoT system composed of a dynamic set of things that connect and cooperate temporarily to achieve a user goal. The approach enables the distributed formation, enactment, adaptation of ECs, and conflict resolution among them. We present a conceptual model of the entities of the approach, its underlying processes, and the guidelines for using it. Moreover, we report about the simulations conducted to validate the feasibility of the approach and evaluate its scalability.

Keywords: emergent configurations; artificial intelligence; self-adaptive IoT systems

1. Introduction

The Internet of Things (IoT) has enabled physical objects and devices such as sensors, actuators, and appliances, to connect and collaborate to achieve users' goals. By leveraging such things, the IoT enables novel types of services and applications in various domains such as transportation and logistics, health-care, and building automation [1].

IoT environments are often dynamic and uncertain. For instance, things are sometimes mobile and often resource-constrained with respect to energy. Thus, they can appear, be discovered, and join IoT systems at runtime. Likewise, they can become suddenly unavailable. Additionally, environmental conditions monitored and controlled through IoT devices, such as ambient temperature and oxygen levels, can change suddenly. Human users benefit from the IoT when they can achieve their goals seamlessly and effortlessly in arbitrary environments with different types of things. For instance, a user should be able to adjust the temperature in her bedroom, office, and a hotel room that she visits for the first time. After the requested temperature is reached, it should be maintained automatically. Further, users might compete on the things available in their environments, and sometimes their goals cannot be achieved simultaneously.

To address the aforementioned challenges, we exploit Artificial Intelligence (AI) to engineer smart IoT systems that can achieve users' goals and cope with the dynamicity and uncertainty of their environments. More specifically, we propose an approach that exploits the notion of Belief-Desire-Intention (BDI) agents [2] to realize Emergent Configurations (ECs) in the IoT. An EC is an IoT system composed of a dynamic set of things that connect and cooperate temporarily to achieve a user goal. A thing is any connected object or device, with its functionalities, services, or applications [3,4].

An *agent* is an autonomous computer system situated in a possibly dynamic and uncertain environment, in which it tries to achieve its goals [5,6]. There has been an increasing interest in exploiting agents' technology to engineer IoT systems [7]. One of the most common models for programming intelligent agents is the BDI model proposed by Rao and Georgeff [2]. BDI-agents make rational decisions about their actions by reasoning about their (i) beliefs about themselves and their environments; (ii) desires, which correspond to goals; (iii) intentions, which represent the desires that agents commit to achieving [2]. BDI-based agents are autonomous, goal-directed, context-aware, and can communicate and collaborate to form Multi-Agent Systems (MAS). These characteristics make them very suitable to be exploited for realizing ECs [7–9]. Our approach also employs the notion of roles to design resilient ECs that can cope with the dynamicity and uncertainty of IoT environments. A *role* represents a pattern of behavior(s) that is common among different classes of agents [10]. It defines the tasks that the playing agent(s) should be able to perform [11].

To summarize, the main *contribution* of this paper is an agent-based approach that enables the distributed formation, enactment, self-adaptation of ECs, and conflict resolution among them. A few approaches have been proposed to realize dynamic goal-driven IoT systems. However, the majority of those approaches adopt centralized architectures, resulting in limited scalability. Additionally, they support only reactive adaptations of the systems, typically in response to their constituents' sudden unavailability. Moreover, they require users to express their goals whenever they want the goals to be achieved. In contrast, our approach adopts a distributed architecture, enables both reactive and proactive adaptations of ECs, and contributes towards automating the achievement of users' goals by leveraging autonomous agents that act on behalf of the users. The proposed approach comprises a conceptual model that describes its key entities and their relationships and the processes applied to realize ECs. We also provide guidelines for developers on using the approach.

The remainder of this paper is organized as follows—Section 2 presents scenarios about ECs. Section 3 introduces the approach. Section 4 presents guidelines for developers on using the approach. Section 5 presents the experiments conducted to validate the feasibility of the approach and evaluate its scalability. Section 6 discusses the approach. Section 7 presents related work. Finally, Section 8 concludes the paper and outlines future work directions.

2. Emergent Configurations Scenarios

In this section, we present two scenarios: one focuses on realizing ECs in a smart meeting building and encompasses the self-adaptation aspect; the other concerns realizing ECs at the scale of a smart city.

2.1. The Smart Building Scenario

In a smart building, spaces such as meeting rooms and halls are set up automatically through mobile walls based on the planned activity and the number of attendees. Participants in a conference organized in the building express their interest in attending specific sessions using applications that run on their smart devices (e.g., smartphones). Accordingly, the existing spaces are divided or combined automatically, and users are given directions on the sessions' locations via their smart devices. When smart spaces are set up, the things available within a space might change. Also, attendees might bring (personal) things with them (e.g., smart devices).

In the building, several agents are programmed to support users' activities. The smart Lighting Agent (LA) configures the light level in spaces based on ongoing activities and available things, such as light sensors, curtains, and lamps actuators. Likewise, the Display Management Agent (DMA) configures the display tools, for example, laptops, cameras, screens, and projectors available in spaces automatically. Similarly, the Temperature Management Agent (TMA) configures the temperature levels in spaces based on available Heating, Ventilation, and Air conditioning (HVAC) devices such as air-conditioners, heating units, and cooling fans. Moreover, the TMA can forecast temperatures in spaces during sessions using Machine Learning (ML) techniques on the data collected from the things in the spaces such as humidity and temperature sensors, air-conditioning status (i.e., on/off), air-conditioning temperature, and underfloor heating temperature. For the participants' comfort, based on the foreseen temperature, the TMA adjusts the HVAC devices in the spaces proactively. Finally, the Fire Management Agent (FMA) detects and controls fire through smoke and flame sensors, fire alarms, and sprinklers.

When a space is set up, the TMA evaluates if the space's temperature is within a specified comfort range (e.g., 20°–22°). If the temperature is not within the range, the TMA forms an EC (EC_1 from now on) from available things and enacts it automatically. For instance, if the temperature in a space is above the range, the EC constituents could be two air-conditioners that are automatically turned on and set up to operate in cooling mode with a temperature degree specified by the TMA.

To present her/his research, using an application that runs on her/his smartphone, a user expresses her/his goal to "give a presentation". Available things are discovered, and a set of suitable things are chosen dynamically to form an EC (EC_2 from now on) that achieves the goal. For instance: the smartphone, the projector, the light sensor, and the curtains actuator. The DMA turns on the projector, which shows the presentation streamed by the smartphone. Meanwhile, the LA closes the curtains automatically due to the high light level detected by the light sensor.

During the presentation, the projector turns off suddenly. The event is automatically detected and analyzed by the DMA, which proposes the user to continue the presentation using the available screen. The user accepts the proposal and EC_2 is adapted automatically. Meanwhile, the TMA forecasts that the temperature in the space is expected to increase in the next ten minutes. Therefore, for the participants' comfort, it proactively adapts EC_1 by turning on a third air-conditioner and setting it up to work in the cooling mode with a specific temperature.

While the presentation is ongoing, EC_3 is dynamically formed and enacted to "evacuate the building" due to fire. For instance, the FMA activates the fire alarm. The DMA recognizes people using video cameras and guides them to exits by displaying instructions on screens. The TMA turns off HVAC devices, and the LA opens the curtains and turns on the lamps to facilitate the evacuation.

2.2. The Smart City Scenario

In a smart city, agents are programmed using rules to publish advertisements automatically. For instance, an agent advertises boat trips according to a set of rules, including the following. If available boats can carry out more trips than those needed to transport people queuing and rain is forecasted within an hour, the agent should advertise boat trips to people in the surrounding, providing a discounted price. Similarly, an agent is programmed to advertise the city museum. For instance, if the numbers of people in the museum and those queuing are above specific thresholds, then some people queued recently get advertisements to come back later with discounted prices.

To publish advertisements, ECs are formed and enacted automatically. For instance, the server of the company that organizes boat trips connects and streams videos to the users' smart devices –thanks to the smart city guide application that users install on their smart devices for tourism purposes. Advertisements can also be displayed on public displays or message boards with lights. Additionally, the current temperature in the area, the number of available boats, and the number of people queuing for trips are displayed alongside the advertisements. This information is provided by the city's IoT infrastructure, which is managed by the City Management Agent (CMA). The temperature can be provided, for instance, by a dynamically discovered sensor, while the other information can be provided by a camera close to the harbor or by two break beam sensors. Similarly, ECs are realized to advertise the museum. When multiple agents request to publish advertisements concurrently, ECs compete for displaying advertisements on users' devices or public displays.

In the city, police incident management agents are programmed to recognize incidents automatically, rank them according to their level of emergency, and take appropriate actions (e.g., dispatch patrols or request medical services). For this purpose, ECs are formed and enacted dynamically. For instance, an EC is formed dynamically to recognize an ongoing incident in the harbor area. The constituents are, for example, sound sensors and a camera that are instructed by the CMA to live stream sound and video to the police server. The camera also moves automatically to record the incident or as requested by the police officer in the operations room. The server can process the streamed data and classify the incident (e.g., shooting). In this case, the EC formed to analyze the incident competes for the use of the camera with the ECs formed to advertise boat trips, as when the camera moves, it cannot count people and boats at the harbor.

3. The Approach

3.1. Conceptual Model

Figure 1 shows a conceptual model that refines the one in a preliminary study [12]. An EC is formed and enacted to achieve a *user goal* in a specific *context*. This is represented by the *ternary relation* EGC . For this purpose, our approach enables collaboration among two types of *BDI-based agents*: (i) *Core agents*, involved in the realization of all ECs. Examples from Figure 2 are the User Agent (UA), the Emergent Configurations Manager (ECM), and the Agents Manager (AM). The UA is responsible for enabling users to interact with the system or for acting on behalf of those users. A user can interact with one or more UAs. The ECM is responsible for instantiating abstract schemas (see below) and orchestrating the application-level agents to realize ECs. The AM is responsible for registering the application-level agents and reporting changes in their availability to the ECM. (ii) *Application-level agents* (e.g., TMA in Figure 2) are responsible for instantiating concrete schemas (see below) and forming and enacting sub-teams to realize sub-goals of a user's goal.

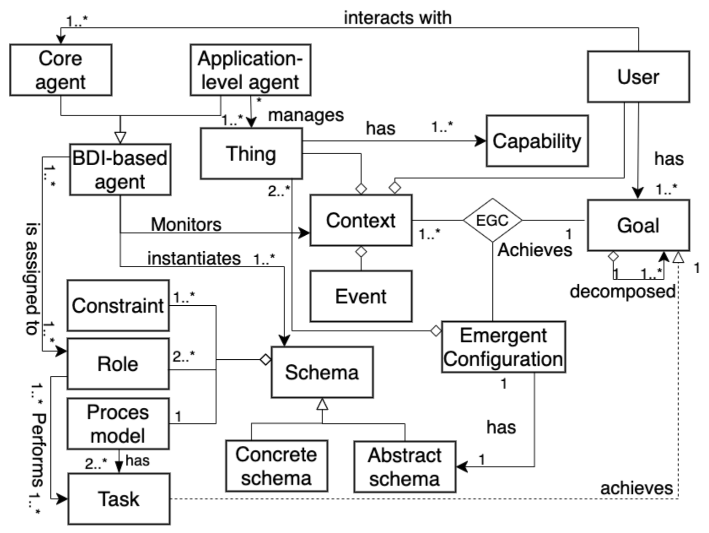


Figure 1. Conceptual model of the approach (described in UML).

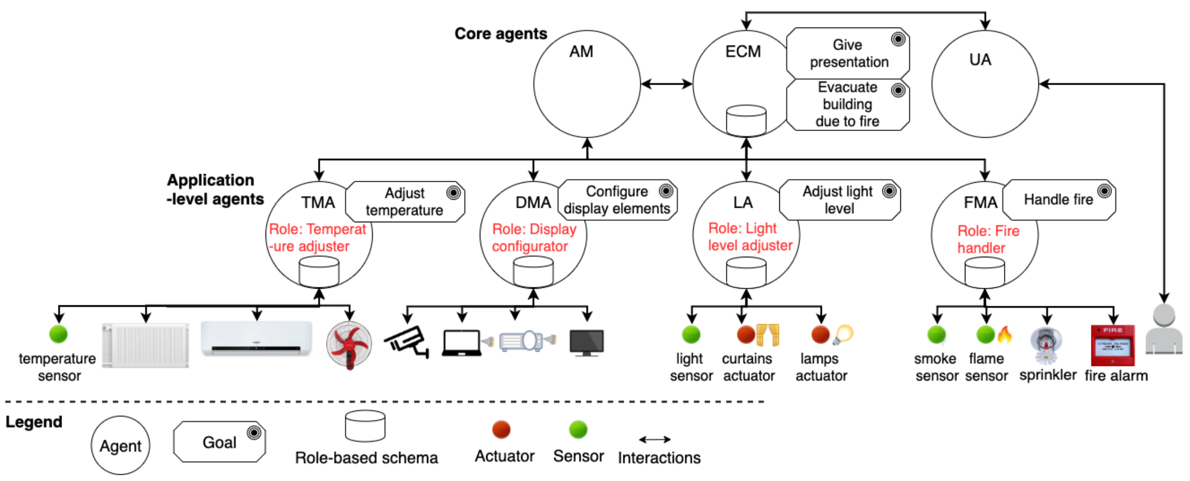


Figure 2. A snapshot of the runtime environment of the approach when realizing the smart meeting room scenario.

A *role* represents a stereotype of tasks that the assignee should perform. An abstract *process model* specifies the order of the execution of the different roles' tasks. *Constraints* specify the requirements that an assignee should meet to adopt a role. We have two types of *schemas*: *abstract* and *concrete schemas*. They both include a set of roles, a set of constraints, and an abstract process model, but they show the following differences. Within an *abstract schema*, roles are assigned to agents (e.g., TMA), and the process models comprise high-level tasks (e.g., adjust temperature). While within a *concrete schema*, roles are assigned to things (e.g., temperature sensor), and the process models comprise finer grained tasks (e.g., detect temperature level). Additionally, constraints within abstract schemas are defined at the agents' level. For example, the TMA can adopt the temperature adjuster role because it can autonomously adjust the temperature in the smart meeting room. While, in concrete schemas, the specified constraints are related to the things. For instance, only connected and operational things can adopt roles. Moreover, constraints can specify the number of assignees that can adopt the same role simultaneously. *Instantiating an abstract schema* means dynamically assigning the roles it comprises to available agents, which in turn *instantiate concrete schemas* by forming sub-teams and assigning the included roles to the things they manage. Although the enactment of ECs is driven by process models, we consider them emergent as their constituents are specified dynamically based on the things discovered at runtime. An *event* describes a dynamic change in the context. The considered types of events are (a) things disconnection; (b) things moving outside the spatial boundaries of their ECs; (c) things running out of battery; (d) application-dependant events: This category comprises two sub-types of events. Events of the first sub-type (d_1) are not related to the changes in the status of things but are triggered when users' goals are no longer achieved due to sudden changes in their environments. For instance, the temperature is higher than the level requested by the user. Events of the second sub-type (d_2) are triggered when changes in the users' environments are forecasted. For instance, the temperature in a room is forecasted not within the users' comfort zone.

For each *thing*, we model its type (e.g., temperature sensor), the capabilities supported by the type (e.g., sense light level), the thing's operational and connectivity status, and battery level (when applicable). Additional attributes can be modeled to represent the current state of things (e.g., if curtains are closed or opened). A *capability* might have pre-conditions that should be satisfied before it can be performed. For instance, light lamps can be turned on only if they were turned off and vice versa. Finally, an application-level agent can manage multiple things. But, each thing is managed only by one application-level agent at the same time.

3.2. Agents' Beliefs

Agents monitor their environments continuously and update their beliefs accordingly. A UA has beliefs about a user's current location and a set of spatial boundaries (i.e., locations). The AM has beliefs about application-level agents, their availability status, and the spatial boundaries of the things they manage. The ECM has beliefs about requested ECs that await enactment, active ECs that are being enacted, and ECs whose enactment is suspended due to conflicts. Moreover, the ECM belief set includes a priority-based hierarchy of roles used for resolving conflicts among ECs (see Section 3.4.3).

Application-level agents have beliefs about the things they manage, the sub-teams they formed, and the enactment status of those sub-teams. Additionally, application-level agents have beliefs about required domain knowledge (e.g., the suitable light level during a presentation). Further, their belief sets include information about the maximum number of times a thing can perform a capability simultaneously (when applicable). For instance, a screen can display one advertisement at a time. Moreover, they include information about the capabilities that cannot be performed simultaneously. For instance, it is not possible to perform the capabilities "open curtains" and "close curtains" by an actuator simultaneously. These beliefs are used for enabling the distributed resolution of conflicts among ECs, as explained in Section 3.4.3.

3.3. Agents' Desires and Intentions

As already mentioned, agents' intentions are the desires that agents commit to achieving based on their environments' status. User agents' intentions are specified either by users when they express goals or automatically by enacting rules. A rule has an antecedent and a consequent. An Agent evaluates continuously if the antecedents of its rules hold according to their beliefs and consequently commits to perform the rules' consequent. For instance, in Listing 1 below, we provide a representation of a rule that the agent that advertises boat trips (see Section 2.2) uses to specify its intentions. Below, `y` denotes the number of people that a boat can transport. For simplicity, we assumed that all boats are similar with respect to the number of people they can transport. `z` represents the number of extra available boats, `d` denotes the duration of the forecast (e.g., in the next hour), and `p` represents the percentage of the discount on the price of the trips to be advertised.

Listing 1: An agent's rule to advertise boat trips.

```
IF(NumberOfAvailableBoats − $\frac{1}{y}$ NumberOfQueuingPeople > z
AND RainForecastedWithin (d))
THEN AdvertiseTrips(p)
```

The ECM, AM, and application-level agents' intentions are specified according to the processes presented below.

3.4. Processes

This section presents the process for enabling the distributed formation, enactment, adaptation of ECs, and conflict resolution among them. Investigating the accuracy of ECs in achieving their goals is planned for future work.

3.4.1. ECs Formation

Figure 3 illustrates the process of enabling the *distributed formation* of ECs. The process starts with expressing a user goal. This sub-process can be performed either manually by the user or automatically by the user agent through executing rules. Expressing the goal means specifying at least the goal type (e.g., give a presentation) and the goal spatial boundaries (e.g., a room). ECs can also be formed and/or adapted automatically when changes in users' environments are forecasted (see more in Section 3.4.2). Then, the ECM loads the abstract schema that corresponds to the selected goal type. Afterward, the ECM queries the AM for the application-level agents ($A_1 ... A_n$ in Figure 3) that manage the things situated within the goal's spatial boundaries and requests them to form sub-teams for performing the high-level tasks in the schema. The sub-team formation process is presented in Algorithm 1.

Algorithm 1: Sub-teams formation algorithm

```
1  Function HandleFormationRequest(highLevelTask):
2      $schema \leftarrow getConcreteSchema(highLevelTask)$
3      if $schema != null$ then
4          $initialAssignment \leftarrow null$ $things \leftarrow getManagedThings()$
5          foreach task $t_i \in schema.tasks$ do
6              $r_i \leftarrow getRole(t_i, schema)$
7              $suitableThingAndCapability \leftarrow null$
8              $suitableThingAndCapability \leftarrow findThingAndCapability(t_i, things)$
9              if $suitableThingAndCapability == null$ then
10                 return
11             else
12                 $th \leftarrow suitableThingAndCapability.getTh()$; $ca \leftarrow suitableThingAndCapability.getCap()$;
                     $initialAssignment.add(t_i, r_i, th, ca)$
13             end
14         end
15         $commitAndWait(highLevelTask, initialAssignment)$
16     if $AgentIsAssignedRole()$ then
17         $assignRolesToThings(schema, initialAssignment)$
18     else
19         $terminateCommitment(highLevelTask)$
20     end
```

If an application-agent has a concrete schema that can be used to achieve a high-level task (line 2), it evaluates if the things it manages meet the constraints of adopting the roles in the schema, have the capabilities to perform the roles' tasks, and the preconditions of those capabilities are satisfied in the context (lines 5–8). If a task cannot be achieved, the process terminates (line 9). Things that meet the aforementioned conditions are assigned initially the corresponding role in the abstract schema (line 12). If all tasks in the concrete schema are achievable, the agent commits to the ECM that it can achieve the high-level task autonomously (line 15). The commitment is valid until the ECM notifies the agent whether it is assigned the role or not. The agent can also query the ECM concerning the assignment and terminate the commitment automatically if the ECM does not respond within a specified period (line 19).

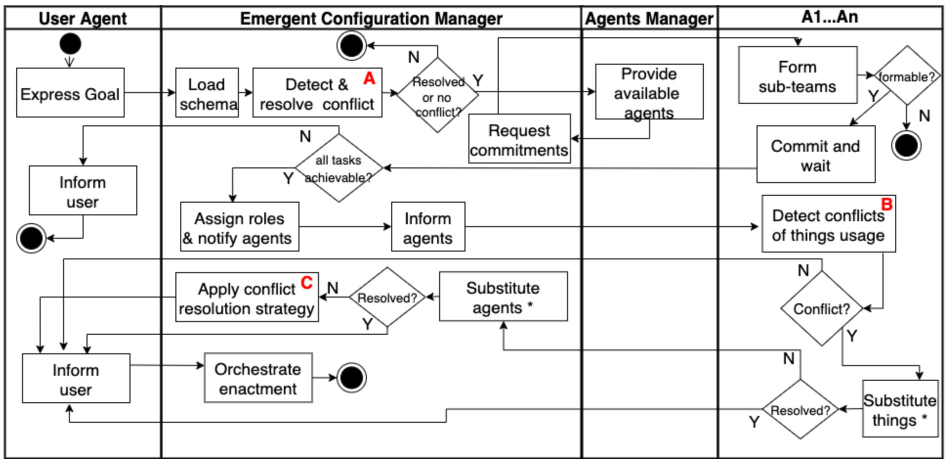


Figure 3. Emergent Configurations (ECs) formation process. The two sub-processes marked with * were not implemented in our prototype due to constraints imposed by the design of the used framework.

If all the tasks in the abstract process model are achievable, the ECM forms an EC by assigning the included roles to (some of) the application-level agents that committed to adopting them. The agents form sub-teams by assigning the roles in concrete schemas to chosen things based on the already performed analysis (line 17).

3.4.2. ECs Adaptation

Figure 4 illustrates the process that enables the *distributed adaptation* of ECs. It is an event-based process that executes continuously the Belief, Desire, Intention cycle to enable reactive and proactive adaptations, as explained in the following. To enable *reactive* adaptations, application-level agents ($A_1 ... A_n$ in Figure 4) monitor continuously the things they manage and the states of their environments. When changes are detected, application-level agents update their beliefs and trigger internal events of the types (a, b, c, and d_1) presented in Section 3.1. Similarly, to enable *proactive* adaptations, application-level agents periodically forecast changes in their environment and trigger events of the type d_2. Application-level agents analyze the events as follows. If an event is of type a,b, or c, the application-level that triggered the event identifies the sub-teams formed by the agent, whose enactment have not terminated yet, and comprise the thing that became unavailable suddenly. After that, the agent tries to adapt the sub-teams by re-assigning the role(s) assigned to the thing to other available things managed by the agent.

If the event is of types d_1, or d_2, the agent evaluates if ECs were formed to achieve goals that are or will become unachieved due to the event. This sub-process handles the case where a user requests to maintain the achievement of a goal that is already achieved in the environment during the request time. For instance, the user requests to maintain the temperature in her/his bedroom in the range of 20°–22°. When the goal was requested, the temperature in the room was already within the specified range. Thus, there was no need to form an EC. However, an EC should be formed automatically when changes in the environment are detected or forecasted. If an EC exists already, the application-level agent adapts it to maintain the achievement of the goal. This can be achieved by adjusting (the settings) of the EC constituents (e.g., to set the air-conditioner to operate at a lower temperature, or to turn on an additional air-conditioner). For this purpose, the agent executes a control loop [13], where the adaptation process is performed gradually until the goal is achieved. For instance, if the temperature degree in the room should be reduced, the responsible agent sets the air-conditioners to operate on lower temperatures gradually. Meanwhile, the agent monitors the environment continuously by analyzing temperature sensors data and adjusts the air-conditioners accordingly. If no EC was formed to achieve the goal, the agent tries to form an EC autonomously by instantiating the concrete schema that corresponds to the goal type, as described in Section 3.4.1.

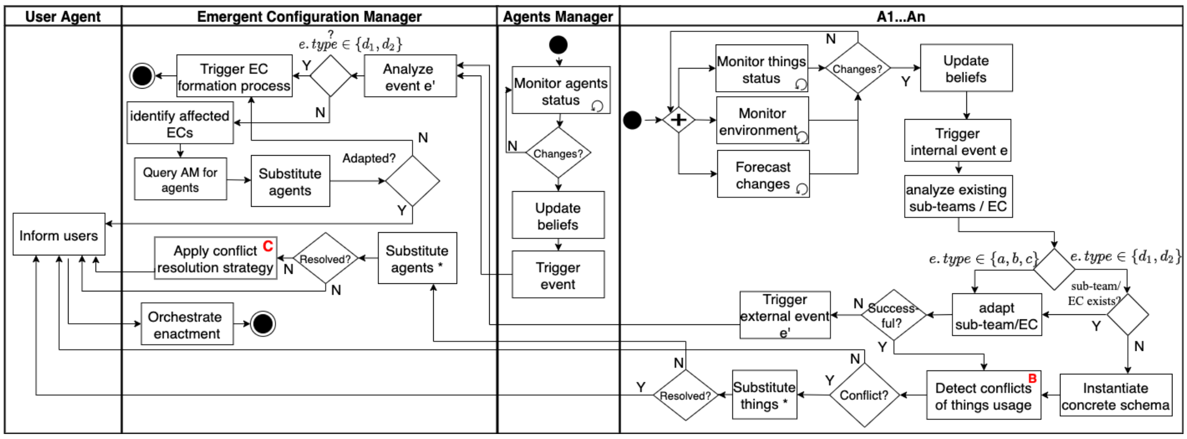


Figure 4. ECs adaptation process.

Meanwhile, the AM monitors the availability of the application-level agents and updates its beliefs when their status changes. If an application-level agent fails to adapt an EC or a sub-team autonomously or when the application-level becomes unavailable, events are triggered and reported to the ECM, which analyzes them and takes actions based on their types. More specifically, if an event is of type d_1 or d_2, the ECM triggers the formation process to enable the collaboration of available agents to achieve the user goal. Given that the agent that triggered the event could not achieve or maintain the goal independently. If the triggered event is of type a, b, or c, the ECM identifies affected ECs (i.e., those whose enactment have not terminated yet) and tries to assign the role(s) assigned to the agent that triggered the event to other agents. If no substitute agent is available, the ECM triggers the formation process to reform the EC (if possible).

3.4.3. Conflict Resolution

Conflicts among ECs can occur at:

1. Goal level: Users might requests goals that can not be achieved simultaneously. For instance, two users request to set different light levels in a smart room.
2. Things level: A thing is assigned to multiple roles and requested to perform capabilities:
    (a) that cannot be provided concurrently. For instance, a curtain's actuator is requested to open and close the curtains simultaneously;
    (b) where the maximum number of times to perform it is reached. For instance, a screen cannot display two advertisements at a time.

The conflict resolution process is distributed among application-level agents and the ECM. It comprises the sub-processes A, B, and C shown in Figures 3 and 4. To resolve conflicts among ECs, our approach applies a priority-based conflict resolution strategy based on a hierarchy of roles. The hierarchy specifies the priorities of different users' roles. For instance, when an EC is realized to enable a user to a give a presentation, the user agent is assigned the role speaker. Similarly, the "system" is assigned the role emergency handler, when ECs are realized to evacuate a building due to fire. The hierarchy specifies, for example, that the role emergency handler has a higher priority than the role speaker. The hierarchy is created by developers and stored in the belief set of the ECM. In order to apply the conflict resolution strategy, we assume that a role-based access control model is developed to specify the users' roles and types of goals that each role can request. Note that a user's role might change dynamically when an EC is realized to achieve her/his goal.

For a specific type of goals such as set light level or temperature, conflicts occur when multiple ECs are realized concurrently within the same spatial boundaries. For instance, a conflict occurs when two ECs are realized to set different light levels in a room simultaneously. Consequently, there should be no more than one active EC that achieves a goal of those types within the same spatial boundaries. For this purpose, the abstract schema used to form ECs to achieve those goals should be flagged by developers as singleton schemas.

Conflicts at the level of goals might also occur when no common (singleton) schemas are instantiated to form ECs. For instance, a user requests the goal "prepare to sleep". Consequently, an EC is formed to make her/his room dark. In the same room, another user requests the goal "play music" and increases the volume of the speakers used to play the music. The enactment of both ECs should not proceed in parallel because the second goal's achievement negatively affects the first user. Detecting and resolving such conflicts are out of the scope of this paper and planned for future work. Table 1 summarizes the possible types of conflicts among ECs and specifies the supported ones.

Figure 5 shows the refinements of sub-processes A and C in Figures 3 and 4. When a goal is expressed, the ECM analyzes if an active EC is instantiated using an abstract schema that is or comprises a singleton schema, which will be instantiated to achieve the requested goal. If a common singleton schema is identified, the ECM evaluates the priorities of the roles of the active EC user and the one who requested the new EC—thanks to the role-based hierarchy and access control model. If the priority of the role of the latter is lower, the request to form an EC is declined and the user is informed. Otherwise, the ECM terminates the enactment of the active EC and instantiates the abstract schema to form as new EC, as presented in Section 3.4.1.

Table 1. The types of conflicts among ECs.

Level	Description	Support
Goal	Conflicts that occur when achieving a goal of one type prevents, or affects negatively, the achievement of a goal of another type.	Not supported
	Conflicts that occur when more than one instance of (singleton) goals of the same type are requested to be achieved concurrently.	Supported
Thing	Conflicts that occur when things are requested to perform capabilities that cannot be provided concurrently.	Supported
	Conflicts that occur when things are requested to perform capabilities where the maximum numbers of times to perform them are reached.	Supported

When sub-teams are formed to form an EC, application-level agents detect conflicts in the use of things (sub-process B in Figures 3 and 4) and try to resolve them autonomously by re-assigning the related roles to other available things they manage. For instance, an agent publishes advertisements to users mobiles when other advertisements are displayed on public screens available on users' environments. If no substitute things are available, the ECM tries to resolve the conflict by re-assigning the related role to another application-level agent. For instance, in a smart city, multiple agents manage advertisements in the city center. If no substitute agent commits to performing the task, the ECM applies the conflict resolution strategy (sub-process C in Figure 5). Mainly, the ECM orders the enactment of ECs based on the priorities of the roles of the users who requested them. When the roles have equal priorities, the first formed ECs are enacted first.

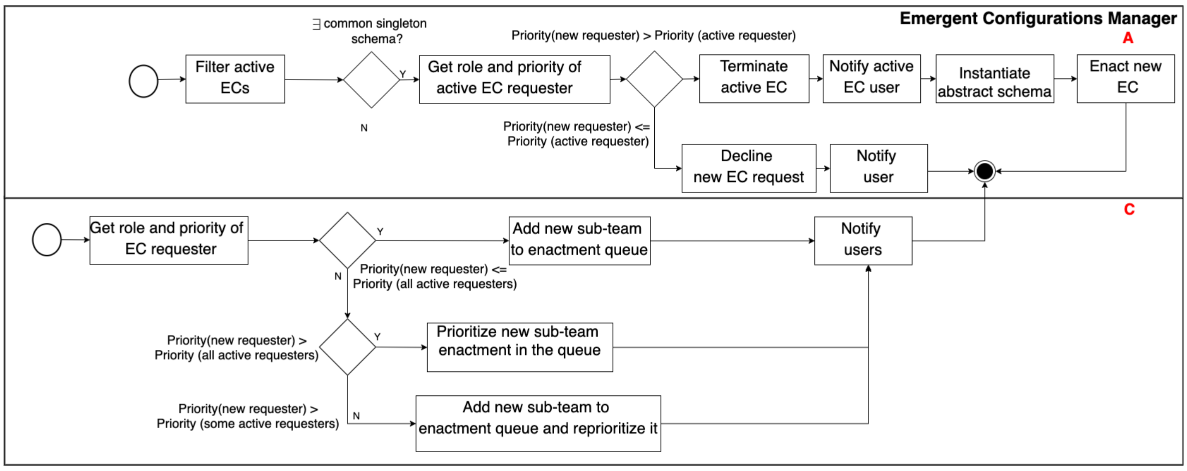

Emergent Configurations Manager
A
∃ common singleton schema?
Priority(new requester) > Priority (active requester)
Filter active ECs
Y
N
Get role and priority of active EC requester
Terminate active EC
Notify active EC user
Instantiate abstract schema
Enact new EC
Priority(new requester) <= Priority (active requester)
Decline new EC request
Notify user
C
Get role and priority of EC requester
Y
Add new sub-team to enactment queue
Notify users
N
Priority(new requester) <= Priority (all active requesters)
Priority(new requester) > Priority (all active requesters)
Y
Prioritize new sub-team enactment in the queue
Priority(new requester) > Priority (some active requesters)
N
Add new sub-team to enactment queue and reprioritize it

Figure 5. Part of the process applied to resolve conflicts among ECs.

3.4.4. ECs Enactment

The ECM orchestrates the enactment of abstract role-based schemas by instructing application-level agents to perform high-level tasks based on the order specified in process models. Similarly, application-level agents instruct things to perform the tasks specified in the concrete role-based schemas by consuming the things' functionalities that are wrapped as services (e.g., REST, SOAP) or exposed through Publish-Subscribe interfaces (e.g., using MQTT protocol).

4. Guidelines for Developers on Using the Approach

To enable the realization of ECs using our approach, developers should perform the following tasks.

1. *Program role-based schemas*: Developers should specify and program the schemas components presented in Section 3.1. Figure 6 illustrates the abstract schema that models the "give a presentation" goal type. The process model of the schema specifies a parallel execution of two loops; each of them comprises a high-level task. The loops are executed until the task "end presentation" is achieved. The same Figure shows also the abstract and concrete schemas for proactively adjusting the temperature. Moreover, developers should specify abstract singleton schemas (e.g., using flags).
2. *Connect things and expose their functionalities*: For instance, things can be connected by registering them in a Cloud platform or through resources at the Edge of the network (e.g., gateways). Things' functionalities can be exposed, for example, as (web) services or Publish-Subscribe interfaces. This requires things' manufacturers to make their functionalities accessible and consumable.
3. *Develop application-level agents*: Developers should program application-dependent agents (e.g., LA). More specifically, they need to populate the agents' beliefs (see Section 3.2). Additionally, they should program the agents' capabilities to use the functionalities of the things they manage. Moreover, to enable (end) users to program (user) agents, developers should program constructs that can be composed into rules.
4. *Deploy agents*: Developers should deploy both core and application-level agents. The user agent can be deployed in one or more of a user's smart devices (e.g., smartphone). The other agents' deployment decisions can be driven by multiple factors such as reliability, performance, security, privacy, and cost. In general, the agents can be deployed in the Cloud or in Edge nodes with suitable processing and storage capabilities. When deploying the ECM, developers should consider the spatial boundaries where the ECM supports the realization of ECs. For instance, more processing and storage capabilities should be dedicated to deploying an ECM that supports the realization of ECs in a smart building than those needed to perform the same task in a smart home. Other factors that influence the placement of the ECM include the number of abstract schemas that it has and the number of users. When deploying the AM, developers should consider the number of available application-level agents registered and monitored by the agent. For instance, in a big smart building, multiple AM instances can be deployed in Edge nodes, where each instance is responsible for one or more floors. When deploying application-level agents, developers should consider the number of concrete schemas the agents have and the things they manage. For instance, in the smart meeting room scenario, application-level agents can be deployed in gateways (e.g., Raspberry Pi).

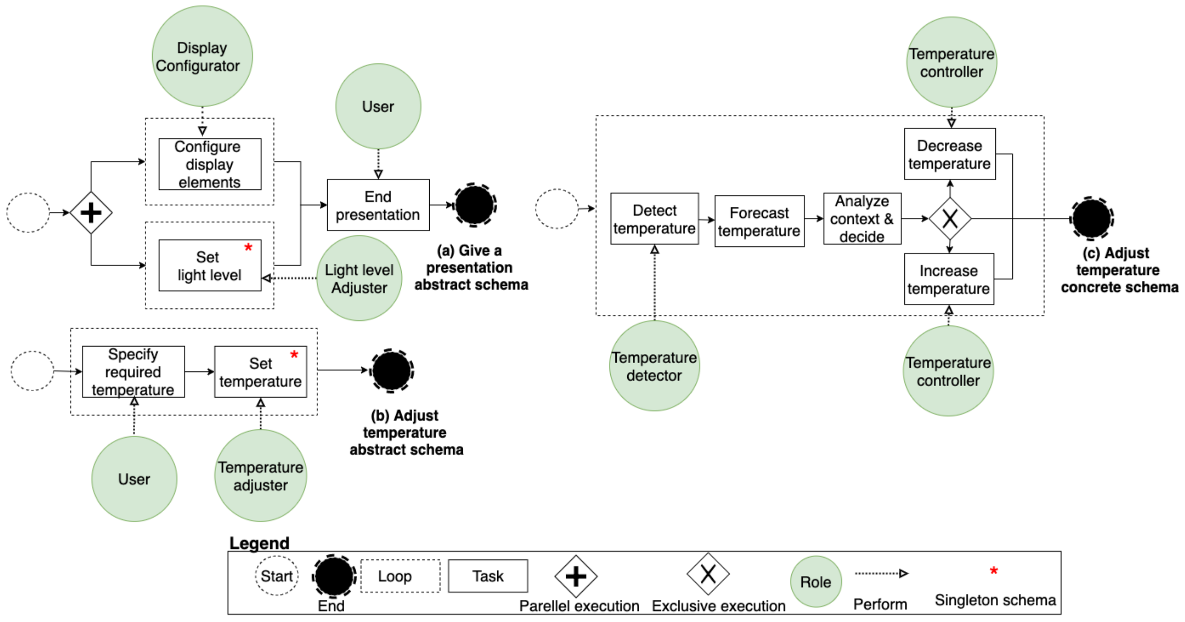


Figure 6. A representation of an abstract and concrete schemas in the smart building scenario.

5. Validation

We used, refined, and extended the Goal-Oriented Teams (GORITE) framework [14] to implement a prototype realizing the scenarios described in Section 3.1. The source code of our prototype is available at https://github.com/iotap-center/rb-ecs. We chose GORITE because it is open source and supports the realization of role-based and goal-driven Multi-Agent Systems (MAS). We conducted a set of experiments to validate the feasibility of the approach in realizing and managing concurrent ECs, and to evaluate its scalability. The experiments were conducted on a dual-core CPU running at 2.7 GHz, with 16 GB memory.

5.1. Smart Building Experiments

In this experiment, we aimed at evaluating the feasibility of the approach in forming and adapting ECs by simulating the smart building scenario presented in Section 3.1. To simulate the dynamicity in spaces with respect to available things, we generated 20 spaces. Each space was programmed to have two things that can be assigned each role in the concrete schemas used for realizing ECs to achieve the goal "give a presentation". We programmed user agents to request the goals simultaneously. To simulate the adaptation of ECs, we designed the experiment to simulate turning a constituent of each EC off. The agents ECM, AM, LA, and DMA collaborated and formed 20 ECs and enacted them successfully. We repeated this experiment for 15 times. The average time for forming, statelessly enacting, and adapting ECs was 42 milliseconds. The standard deviation was 1.2, and the standard error of the mean was 0.33.

Moreover, to forecast temperature in dynamic settings and adapt ECs proactively, we developed a ML model to forecast temperature in spaces. For this purpose, we used the eXtreme Gradient Boosting (XGBoost) algorithm [15] and the dataset used in Reference [16]. The dataset comprises 39,404 patterns structured in ten features, including humidity and temperature levels, air-conditioning status (i.e., on/off), air-conditioning temperature, and underfloor heating temperature. Unlike Reference [16], we simulated dynamic environments where the types of things in spaces are specified randomly.

We chose the (XGBoost) algorithm because it is scalable and deals well with missing data that result from the sudden unavailability of things [15]. Figure 7 shows the results of three experiments performed to evaluate the accuracy of the model in forecasting temperature in dynamic settings. More specifically, in the experiments, we randomly removed two, four, and six features from 20% of the data used for testing the model. The accuracy of the temperature forecast when all the features were provided was 91%. Table 2 shows the results of forecasting the temperature when features are removed randomly.

Table 2. The results of forecasting temperature in dynamic settings.

Number of Features	Accuracy	Mean Squared Error
Eight	88%	0.12
Six	84%	0.16
Four	79%	0.22

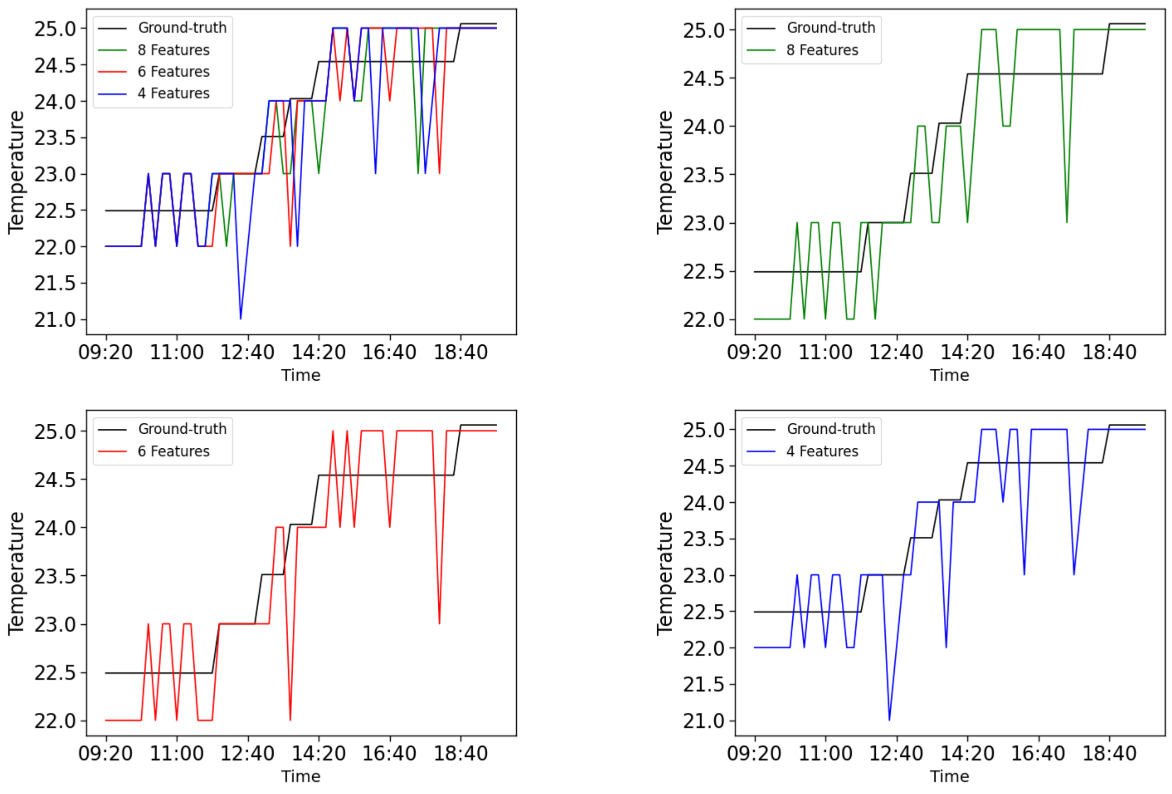


Figure 7. The accuracy of forecasting temperature in dynamic settings.

5.2. Smart City Experiments

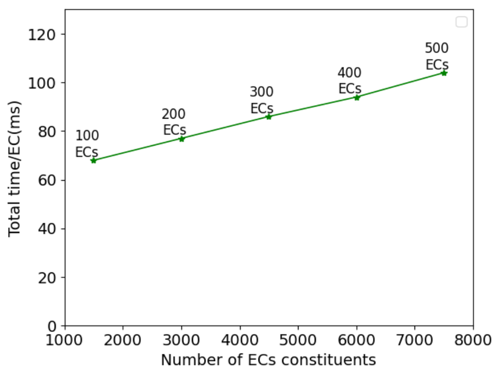
In this experiment, we aimed at evaluating the scalability of the approach when realizing conflicting ECs. We simulated the realization of conflicting ECs in two cases and measured the average time for enabling their distributed formation, enactment orchestration, and conflict resolution over 15 runs. In the *first case* (Figure 8a), we simulated the part of the smart city scenario, where the following concurrent ECs were realized: (i) An EC formed to “analyze an incident” competes for a camera with all ECs formed to “advertise boat trips”. (ii) At the same time, ECs formed to “advertise boat trips” compete for being displayed on smart screens with ECs formed to “advertise the city’s museum”.

We simulated an increasing number of users, agents, and ECs constituents. More specifically, we generated smart spaces automatically. Each space has 15 smart screens and 15 Beacons used for viewing advertisements and sending discount codes, respectively. An agent is responsible for managing advertisements in each space. To ensure that conflicts occur, we specified that only a camera is available, and that except for smart screens, there are no things (e.g., smartphones) that can display advertisements. The ECs formed to analyze the incident and publish advertisements about the museum were not conflicting; thus, they could be enacted concurrently. While the enactment of the ECs formed to advertise boat trips were suspended until the incident was analyzed, given that the police have higher priority when analyzing ongoing incidents than the advertising company. The enactment of the other ECs were prioritized based on their request time. Table 3 provides more information about the results of this experiment.

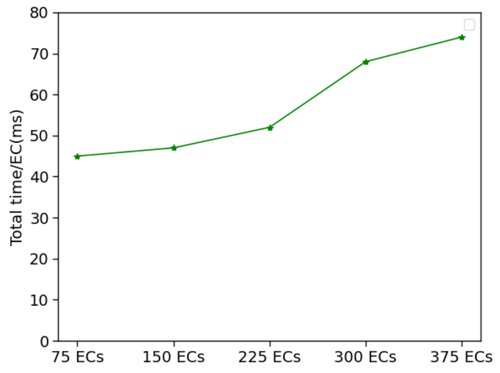
Table 3. The results of case 1 experiment.

Number of Agents	Number of Things	Number of ECs	Average Time / EC (ms)	Standard Deviation	Standard Error of the Mean
53	1506	101	67.8	3.9	1
103	3006	201	77.4	3.85	0.99
153	4506	301	85.8	7.75	2
253	6006	401	94.4	9.3	2.4
353	7506	501	104	13.22	3.41

In the *second case* (Figure 8b), we aimed at evaluating the scalability of the approach when increasing the number of concurrent ECs that compete for available things. We simulated an increasing number of parties that request to "publish advertisements" concurrently in 15 smart spaces that comprise 225 screens across a city. Note that the number of involved things in the two cases are different. This explains why the 300 ECs in the first case consumed more time to be realized than the 300 ECs in the second case. Table 4 provides more information about the results of this experiment.



(**a**) Case 1



(**b**) Case 2

Figure 8. The results of the smart city experiments.

Table 4. The results of case 2 experiment.

Number of Agents	Number of Constituents	Number of ECs	Average Time/EC (ms)	Standard Deviation	Standard Error of the Mean
21	232	75	45	3.7	11.7
26	237	150	47	4	12
31	242	225	52	2	13
36	247	300	68	7.5	17.5
41	252	375	74	9	19

As can be noted in Tables 3 and 4, the standard deviation and standard error of the mean increase in most of the cases when the number of conflicting ECs increase. This can be clarified by the following. In GoRITE, in multi-threaded settings, the formation process is a sub-process of the enactment process. Consequently, conflicts can not be detected before the enactment process is started.

Therefore, ECs enactment could be suspended when they are formed. Moreover, in the experiments, ECs were requested using Java threads created concurrently and managed by the Java Virtual Machine (JVM). We noted that the JVM sometimes tries to enact the ECs that were requested lately. However, due to conflicts, those ECs enactment cannot be proceeded. This slightly increases the time needed to enact the ECs that were requested earlier. This issue can be addressed by decoupling the formation and enactment processes and enabling the enactment only of ECs that have the highest priority.

6. Discussion

Our approach enables the realization of ECs by exploiting AI techniques and the notions of roles and events. Autonomous agents that are empowered with ML techniques are suitable for realizing ECs where users (e.g., janitors) rely on agents to maintain the achievement of several goals automatically (e.g., to maintain the temperature in a smart building). Also, when agents request goals on behalf of users (e.g., to publish advertisements), or where agents detect events and deal with them autonomously (e.g., to evacuate a building due to fire). ECs are realized within well-defined spatial boundaries. Thus, we expect the number of available things to be on a scale that ranges from dozens to few thousands. This notably alleviates the well-known IoT scalability problem [1]. At the smart city scale, the numbers of things and agents are large [17–20]. Thus, several instances of the approach should operate concurrently. Each of those instances is responsible for realizing ECs within well-specified and non-overlapping spatial boundaries. The number of the required instances can be influenced by multiple factors, including the number of things, the capabilities of the available deployment infrastructures, and the expected number of users. The results of the performed experiments validate the feasibility of the approach and indicate that it scales well.

Although core agents are involved in realizing all ECs, the efforts to concretely form, enact, self-adapt ECs, and resolve resulting conflicts are distributed among core agents and application-level agents. This distribution of tasks enables the efficient consumption of available resources. GORITE is designed to run only on one Java virtual machine and enables agents to communicate by accessing shared memory. This imposed a limitation on the deployment of the core and application-level agents in our prototype. More specifically, all the agents had to be deployed on a single machine. However, the processes of our approach can be applied when agents are deployed in distributed resources (e.g., servers and gateways) across the Edge-Cloud continuum where they can interact by exploiting, for example, Contract-Net protocol [21], commitment-based interactions [22], or integrating Interaction-oriented Software Engineering techniques [23]. This would improve more the scalability and performance of the approach.

Based on our experience, developing agents is not complex but depends on the level of their autonomy, the number of things they manage, and the developer's knowledge of the used framework. The complexity of developing schemas depends on the number of the included roles and the complexity of the included process models.

On the one hand, the main advantage of exploiting AI techniques to engineer IoT systems is enabling the engineering of smart IoT systems that can achieve user goals in arbitrary environments, and cope with dynamicity and uncertainty in their environments. These aspects are crucial for the adoption of IoT systems in our everyday life activities. On the other hand, IoT developers should learn agent-based programming languages, which might affect the applicability of the approach. Also, novel approaches are needed to develop explainable AI and enable the interaction of end users with agents.

The limitations of the approach include the following. ECs cannot be realized to achieve goals of types unknown to the ECM. Additionally, currently, when multiple application-level agents can perform the tasks of a role, the role is assigned to one (or more) of them randomly without considering quality-of-service aspects (e.g., security and performance). We plan to address these limitations in our future work.

7. Related Work

Several approaches have been proposed to enable MAS to self-adapt to uncertain environments. However, more efforts are needed to concretely address the problem of engineering self-adaptive MAS in a goal-oriented manner [24]. Ciortea et al. [25] proposed an approach where things are wrapped as agents or artifacts. When an agent cannot achieve a goal individually, agents collaborate in a framework called Socio-Technical Network to compose a goal-driven IoT mashup.

Other approaches exploit agents for modeling ambient assisted living environments [26,27], or for designing manufacturing systems [28]. Unlike our approach, none of the approaches discussed above supports IoT systems to cope with dynamic changes in their environments. Our approach does not require the use of a specific framework compared to the approach by Ciortea et al. [25]. Bures et al. [29] proposed DECCO, a formal approach and framework for enabling agents' coalitions in collective adaptive systems using autonomic component ensembles. For this purpose, the authors proposed and exploited the notion of Invariant Refinement Method (IRM), which supports the iterative refinement of system goals to generate system architectures (i.e., components and ensembles). However, the authors state that it is hard to use the framework to develop real-life systems where ensembles are dynamically formed in a distributed manner. To address this issue, the authors proposed TCOEL, a specification and implementation language based on Scala internal DSL [30].

There has been an increasing interest in developing MAS-based IoT systems by exploiting the Interaction-oriented Programming (IoP) paradigm. The paradigm is concerned with the languages of interactions among agents, the semantics of those interactions, and the tools and techniques for realizing MAS [31]. In Reference [23], the authors motivated the use of decentralized MAS to realize IoT systems. Further, the authors discussed several related challenges, including the need for IoP paradigms for MAS-based IoT systems. Christie et al. [32] proposed Protocols over Things (PoT), a decentralized approach that exploits the notion of roles and agents to realize IoT systems. The approach compiles protocols into interactions' constraints on communications among roles. To adopt roles, agents perform the decision-making required to enact the specifications of those roles. A rich evaluation of communication protocol languages that can be exploited to engineer MAS is presented in Reference [33]. Such protocols can be exploited to fully decentralize our approach, and enable application-level agents to negotiate and collaborate to form, adapt, enact ECs, and resolve conflicts among them.

Other approaches adopt the Web of Things (WoT) standards to enable the dynamic formation of IoT service mashups. Mayer et al. [34] proposed a centralized approach where things, which are modeled as semantically annotated web services, are dynamically composed to form goal-driven IoT mashups. The approach also supports the automated adaptation of the composed mashups apropos the availability of services. Our approach is more distributed, and it considers additional intrinsic contextual properties of the things (e.g., connectivity and operational status). Sohrabi et al. [35] proposed a similar approach that leverages a logic programming language to enable the composition of web services. In general, in approaches that adopt WoT standards, services are annotated to model things properties. Static properties such as things' capabilities can be annotated easily. However, annotating dynamic properties (e.g., battery levels) is more complex and has not been investigated well in the literature.

Chen et al. [36] proposed GoCoMo, a goal-driven and self-organizing model for enabling the decentralized automated composition and adaptation of services in mobile and pervasive environments. The model employs heuristic to prevent the flooding of networks by service requests. It also exploits a backward-chaining AI planning technique to perform reliable service composition and adaptation. AI planning techniques are well-known for being computationally complex, and do not scale well [37].

In previous studies, we proposed the *IoT-FED* approach [38] for enabling the dynamic formation and enactment of ECs. We also proposed the *ECo-IoT* [3] approach for enabling the dynamic formation and adaptation of ECs. The processes presented in Section 3.4 and those presented in References [3,38] share at an abstract level two components, the ECM and the UA. However, the tasks and the process flows in this paper are novel. Table 5 shows the main differences between those approaches and the approach proposed in this paper. ECs were realized in both approaches in a fully centralized manner by leveraging different techniques that rely on AI planning. Unlike both approaches, the approach proposed in this paper enables the realization of ECs in a distributed manner and exploits the notions of BDI-agents and role-based schemas. Additionally, it supports refining abstract schemas into concrete schemas at runtime. These techniques scale better compared to both approaches while preserving flexibility. Indeed, although the approach proposed in this paper supports more functionalities than both approaches, the average time for realizing ECs is in the scale of milliseconds. In contrast, those approaches realize ECs in the scale of few seconds even with less number of things. Moreover, the approach in Reference [3] supports the reactive adaptation of ECs in response to events of the types a, b, and c (see Section 3.1), while the approach proposed in this paper also supports reactive adaptations of the type d_1 and proactive adaptation as well. Supporting these adaptations and exploiting the notion of rules enable the approach in this paper to automate partially users environments. We plan to extend the approach to fully automate user environments in our future work.

Table 5. A comparison with the approaches proposed in References [3,38].

Approach	EC Formation	EC Adaptation	EC Enactment	Conflict Resolution	Environment Automation
ECo-IoT [3]	Centralized	- Centralized - Reactive	Not supported	Not supported	Not supported
IoT-FED [38]	Centralized	Not supported	Centralized	Not supported	Not supported
The agent-based approach	Distributed	- Distributed - Reactive - Proactive	Distributed	Distributed	Partially

The subject of conflict-resolution among agents has been well-researched [39,40]. Liu et al. [41] classified conflicts among agents into three categories: beliefs conflicts, goals conflicts, and plans conflicts. Conflicts can also occur when agents communicate (e.g., when an agent sends two conflicting messages) [42]. Several strategies have been proposed to resolve conflicts. Examples of such strategies include priority agreement, negotiation, voting, and arbitration [43–46]. The selection of an appropriate strategy to resolve conflicts depends on conflicts' types, agents' preferences, and application domains [41]. Few approaches have been proposed to resolve conflicts among IoT systems. Meiyi et al. [47] proposed an architecture that supports the runtime detection and resolution of conflicts among services in smart cities. Unlike our approach, the dynamic formation, enactment, and adaptation of goal-driven IoT systems are not supported. However, we have a similar perspective about conflicts at the level of things. Suri et al. [48] proposed a semantic-based framework that enables the development of IoT-aware business processes. Unlike our approach, the framework requires developers to specify the things that can substitute each other if any of them becomes unavailable.

In summary, by using agents and ML technologies to engineer goal-driven self-adaptive IoT systems, we contribute to a promising unexplored research direction in the MAS and IoT field [7,8,24].

8. Conclusions and Future Work

The majority of existing approaches to engineer IoT systems specify systems' constituents at design time. Such systems have shortcomings in supporting users to achieve their goals in dynamic and uncertain IoT environments. To address this challenge, we proposed an approach that exploits AI techniques to enable the dynamic formation, enactment, and adaptation of ECs. Additionally, the approach enables the detection and resolution of conflicts among ECs. Moreover, we presented guidelines for using the approach.

As future work, we plan to integrate a rule-based engine and perform a more extensive evaluation of the approach. We also plan to enable the approach to predict users' goals, learn their preferences, and form ECs accordingly and automatically (machine learning-based approaches will be exploited for this purpose). Further, we plan to investigate how to support users to construct new abstract schemas for new types of goals. The approach can also be evolved to consider non-functional aspects, for example, privacy, security, performance, energy consumption, and reliability when realizing ECs. Moreover, it could be realized in decentralized settings where core agents are not used. Instead, application-level agents negotiate and collaborate to realize ECs and resolve conflicts among them. Furthermore, we plan to investigate the realization of optimal ECs and the accuracy in achieving their goals. Finally, we assumed that the location and status of things are provided by services automatically. We plan to relax this assumption in our future work.

Author Contributions: Conceptualization, F.A., R.S. and P.D.; methodology, F.A., R.S. and P.D.; software, F.A.; validation, F.A., R.S. and P.D.; investigation, F.A., R.S. and P.D.; writing—original draft preparation, F.A.; writing—review and editing, R.S. and P.D.; visualization, F.A., R.S. and P.D.; supervision, R.S. and P.D.; project administration, F.A., R.S. and P.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the Knowledge Foundation (Stiftelsen för Kunskaps- och Kompetensutveckling), grant number 20140035.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
2. Rao, A.S.; Georgeff, M.P. Modeling Rational Agents within a BDI-architecture. *KR* **1991**, *91*, 473–484.
3. Alkhabbas, F.; Spalazzese, R.; Davidsson, P. ECo-IoT: An Architectural Approach for Realizing Emergent Configurations in the Internet of Things. In Proceedings of the 12th European Conference on Software Architecture, Madrid, Spain, 24–28 September 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 86–102.
4. Ciccozzi, F.; Spalazzese, R. MDE4IoT: Supporting the Internet of Things with Model-driven Engineering. In Proceedings of the International Symposium on Intelligent and Distributed Computing, Paris, France, 10–12 October 2020; Springer: Berlin/Heidelberg, Germany, 2016; pp. 67–76.
5. Wooldridge, M. Agent-based Software Engineering. *IEE Proc. Softw.* **1997**, *144*, 26–37. [CrossRef]
6. Maes, P. Modeling Adaptive Autonomous Agents. *Artif. Life* **1993**, *1*, 135–162. [CrossRef]
7. Savaglio, C.; Ganzha, M.; Paprzycki, M.; Bădică, C.; Ivanović, M.; Fortino, G. Agent-based Internet of Things: State-of-the-art and research challenges. *Future Gener. Comput. Syst.* **2020**, *102*, 1038–1053. [CrossRef]
8. Trentin, I.F.; Boissier, O.; Ramparany, F. Insights about User-Centric Contextual Online Adaptation of Coordinated Multi-Agent Systems in Smart Homes. In Proceedings of the Rencontres des Jeunes Chercheurs en Intelligence Artificielle 2019, Toulouse, France, 2–4 July 2019; HAL: Bengaluru, India, 2019; pp. 35–42.
9. Sirqueira, T.; Viana, M.L.; Cunha, F.J.; Nunes, I.; Lucena, C.J. Data provenance in Multi-agent Systems: Relevance, benefits and research opportunities. *Intern. J. Metadata Semant. Ontol.* **2018**, *13*, 9–19. [CrossRef]
10. Kendall, E.A. Role modelling for agent system analysis, design, and implementation. In Proceedings of the First and Third International Symposium on Agent Systems Applications, and Mobile Agents, Palm Springs, CA, USA, 6 October 1999; IEEE: Piscataway, NJ, USA, 1999; pp. 204–218.
11. Zhu, H.; Zhou, M. Roles in Information Systems: A Survey. *IEEE Trans. Syst. Man Cybern.* **2008**, *38*, 377–396.
12. Mihailescu, R.; Spalazzese, R.; Heyer, C.; Davidsson, P. A Role-based Approach for Orchestrating Emergent Configurations inthe Internet of Things. *arXiv* **2018**, arXiv:1809.09870.
13. Kephart, J.O.; Chess, D.M. The Vision of Autonomic Computing. *Computer* **2003**, *36*, 41–50. [CrossRef]
14. Jarvis, D.; Jarvis, J.; Rönnquist, R.; Jain, L.C. Getting Started with GORITE. In *Multiagent Systems and Applications*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 13–29.
15. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd Acm Sigkdd International Conference On Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.
16. Alawadi, S.; Mera, D.; Fernández-Delgado, M.; Alkhabbas, F.; Olsson, C.M.; Davidsson, P. A comparison of machine learning algorithms for forecasting indoor temperature in smart buildings. *Energy Syst.* **2020**, 1–17. [CrossRef]

17. Blair, G.; Bromberg, Y.; Coulson, G.; Elkhatib, Y.; Réveillère, L.; Ribeiro, H.B.; Rivière, E.; Taïani, F. Holons: Towards a systematic approach to composing systems of systems. In Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware, Vancouver, BC, Canada, 7–11 December 2015; pp. 1–6.
18. Pico-Valencia, P.; Holgado-Terriza, J.A. Agentification of the Internet of Things: A systematic literature review. *Int. J. Distrib. Sens. Netw.* **2018**, *14*, 1550147718805945. [CrossRef]
19. Arellanes, D.; Lau, K. Evaluating IoT service composition mechanisms for the scalability of IoT systems. *Future Gener. Comput. Syst.* **2020**, *108*, 827–848. [CrossRef]
20. Khanouche, M.E.; Atmani, N.; Cherifi, A.; Chibani, A.; Matson, E.T.; Amirat, Y. QoS-Aware Agent Capabilities Composition in HARMS Multi-agent Systems. In Proceedings of the International Conference on Practical Applications of Agents and Multi-Agent Systems, Avila, Spain, 26–28 June 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 127–138.
21. Smith, R.G. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.* **1980**, *29*, 1104–1113. [CrossRef]
22. Boukadi, K.; Faci, N.; Maamar, Z.; Ugljanin, E.; Sellami, M.; Baker, T.; Al-Khafajiy, M. Norm-based and Commitment-driven Agentification of the Internet of Things. *Internet Things* **2019**, *6*, 100042. [CrossRef]
23. Singh, M.P.; Chopra, A.K. The Internet of Things and Multiagent Systems: Decentralized Intelligence in Distributed Computing. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1738–1747.
24. Hrabia, C.; Lützenberger, M.; Albayrak, S. Towards adaptive multi-robot systems: Self-organization and self-adaptation. *Knowl. Eng. Rev.* **2018**, *33*, e16. [CrossRef]
25. Ciortea, A.; Boissier, O.; Zimmermann, A.; Florea, A.M. Responsive Decentralized Composition of Service Mashups for the Internet of Things. In Proceedings of the 6th International Conference on the Internet of Things, Stuttgart, Germany, 7–9 November 2016; ACM: New York, NY, USA, 2016; pp. 53–61.
26. Sernani, P.; Claudi, A.; Palazzo, L.; Dolcini, G.; Dragoni, A.F. Home Care Expert Systems for Ambient Assisted Living: A Multi-Agent Approach. In Proceedings of the Workshop on The Challenge of Ageing Society: Technological Roles and Opportunities for Artificial Intelligence; CEUR-WS, Turin, Italy, 6 December 2013.
27. Abras, S.; Ploix, S.; Pesty, S.; Jacomino, M. A Multi-agent Home Automation System for Power Management. In *Informatics in Control Automation and Robotics*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 59–68.
28. Ciortea, A.; Mayer, S.; Michahelles, F. Repurposing manufacturing lines on the fly with multi-agent systems for the web of things. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, Stockholm, Sweden, 10–15 July 2018; pp. 813–822.
29. Bures, T.; Gerostathopoulos, I.; Hnetynka, P.; Keznikl, J.; Kit, M.; Plasil, F. DEECO: An ensemble-based component system. In Proceedings of the 16th International ACM Sigsoft Symposium On Component-Based Software Engineering, Vancouver, BC, Canada, 17–21 June 2013; ACM: New York, NY, USA, 2013; pp. 81–90.
30. Bures, T.; Gerostathopoulos, I.; Hnetynka, P.; Plasil, F.; Krijt, F.; Vinarek, J.; Kofron, J. A language and framework for dynamic component ensembles in smart systems. *Int. J. Softw. Tools Technol. Transf.* **2020**, *22*, 1–13. [CrossRef]
31. Singh, M.P. *Toward Interaction-Oriented Programming*; North Carolina State University at Raleigh: Raleigh, NC, USA, 1996.
32. Christie, S.H.; Smirnova, D.; Chopra, A.K.; Munindar, P.S. Decentralized Programming for the Internet of Things. In Proceedings of the 8th International Workshop on Engineering Multi-Agent Systems, Auckland, New Zealand, 9–10 May 2020.
33. Chopra, A.K.; Christie, V.S.H.; Singh, M.P. An Evaluation of Communication Protocol Languages for Engineering Multiagent Systems. *arXiv* **2019**, arXiv:1901.08441.
34. Mayer, S.; Verborgh, R.; Kovatsch, M.; Mattern, F. Smart Configuration of Smart Environments. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 1247–1255. [CrossRef]
35. Sohrabi, S.; Prokoshyna, N.; McIlraith, S.A. Web service composition via the customization of Golog programs with user preferences. In *Conceptual Modeling: Foundations and Applications*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5600, pp. 319–334.
36. Chen, N.; Cardozo, N.; Clarke, S. Goal-driven service composition in mobile and pervasive computing. *IEEE Trans. Serv. Comput.* **2016**, *11*, 49–62. [CrossRef]

37. Ghallab, M.; Nau, D.; Traverso, P. *Automated Planning: Theory and Practice*; Elsevier: Amsterdam, The Netherlands, 2004.
38. Alkhabbas, F.; De Sanctis, M.; Spalazzese, R.; Bucchiarone, A.; Davidsson, P.; Marconi, A. Enacting Emergent Configurations in the IoT Through Domain Objects. In Proceedings of the International Conference on Service-Oriented Computing, Hangzhou, China, 12–15 November 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 279–294.
39. Huang, P. An application of quantitative techniques to conflict resolution in a multi-agent system. *Comput. Electr. Eng.* **2003**, *29*, 757–779. [CrossRef]
40. Alshabi, W.; Ramaswamy, S.; Itmi, M.; Abdulrab, H. Coordination, cooperation and conflict resolution in multi-agent systems. In *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*; Springer: Dordrecht, The Netherlands, 2007; pp. 495–500.
41. Liu, T.; Goel, A.; Martin, C.; Barber, K. *Classification and Representation of Conflict in Multi-Agent Systems*; Technical report; The Laboratory for Intelligent Processes and Systems, University of Texas at Austin: Austin, TX, USA, 1998.
42. Singh, M.P. Semantics and verification of information-based protocols. In *AAMAS*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 1149–1156.
43. Wai Tsang, C.; Kin Ho, T. Conflict resolution through negotiation in a railway open access market: A multi-agent system approach. *Transp. Plan. Technol.* **2006**, *29*, 157–182. [CrossRef]
44. Liu, Q.; Cui, X.; Hu, X. Conflict resolution within multi-agent system in collaborative design. In Proceedings of the 2008 International Conference on Computer Science and Software Engineering, Wuhan, China, 2–14 December 2008; IEEE: Piscataway, NJ, USA, 2008; Volume 1, pp. 520–523.
45. Adler, M.R.; Davis, A.B.; Weihmayer, R.; Worrest, R.W. Conflict-resolution strategies for nonhierarchical distributed agents. In *Distributed Artificial Intelligence*; Elsevier: Amsterdam, The Netherlands, 1989; pp. 139–161.
46. Barber, K.; Liu, T.; Han, D. Strategic decision-making for conflict resolution in dynamic organized multi-agent systems. *Spec. Issue CERA J.* **2000**, *9*, 1–18.
47. Ma, M.; Preum, S.M.; Tarneberg, W.; Ahmed, M.; Ruiters, M.; Stankovic, J. Detection of Runtime Conflicts among Services in Smart Cities. In Proceedings of the 2016 IEEE International Conference on Smart Computing (SMARTCOMP), St Louis, MO, USA, 18–20 May 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–10.
48. Suri, K.; Gaaloul, W.; Cuccuru, A.; Gerard, S. Semantic Framework for Internet of Things-aware Business Process Development. In Proceedings of the 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Poznan, Poland, 21–23 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 214–219.