Semantics-based platform for context-aware and personalized robot interaction in the Internet of Robotic Things

Christof Mahieu[a], Femke Ongenae[a], Femke De Backere[a], Pieter Bonte[a], Filip De Turck[a], Pieter Simoens[a]

[a] *IDLAB, Ghent University - imec, Technologiepark 15, 9052 Gent, Belgium*

Abstract

Robots are moving from well-controlled lab environments to the real world, where an increasing number of environments has been transformed into smart sensorized IoT spaces. Users will expect these robots to adapt to their preferences and needs, and even more so for social robots that engage in personal interactions. In this paper, we present declarative ontological models and a middleware platform for building services that generate interaction tasks for social robots in smart IoT environments. The platform implements a modular, data-driven workflow that allows developers of interaction services to determine the appropriate time, content and style of human-robot interaction tasks by reasoning on semantically enriched IoT sensor data. The platform also abstracts the complexities of scheduling, planning and execution of these tasks, and can automatically adjust parameters to the personal profile and current context. We present motivational scenarios in three environments: a smart home, a smart office and a smart nursing home, detail the interfaces and executional paths in our platform and present a proof-of-concept implementation.

Keywords: Internet of Things, personalization, semantics, ontology, context-aware systems, social robots

1. Introduction

The integration of service robots in our daily life is being envisioned in an increasing number of environments. Service robots are defined by the

Email address: `christof.mahieu@ugent.be` (Christof Mahieu)

ISO as “robots that perform useful tasks for humans or equipment excluding industrial automation applications” [1]. This definition spans a wide variety of form factors, from robots assisting in a single task like vacuum cleaning, to humanoid social companion robots. Users have high expectations in terms of usability of service robots, wishing for robots capable of handling a wide range of chores at home, at work, during leisure or that provide assistance to the handicapped and elderly [2].

However, this positive attitude towards service robots is highly dependent on the reliability and efficiency by which the robots execute their tasks [2]. When it comes to social companion robots, the desired robot behavior is highly dependent on the actual context and the personal preferences. For instance, a social robot thats adapts to the emotional state of a person leads to more enjoyable experiences [3], a domestic butler robot should not remind one to take medicines in the presence of guests, etc. Personalization also increases usability on the longer term [4]. For example, health conditions of people suffering from chronic diseases change over time, a family may welcome a baby, children grow up, etc. When users perceive the system as adapting to their changing needs, they will find it more useful and will be more accepting towards the system [5].

These requirements for contextualization and personalization in task execution pose significant challenges to robotic controllers, especially when considering real-world environments instead of well-controlled lab conditions. Social robots must be able to grasp the wider context, beyond the perception range of their on-board sensors. This can be realized by embedding the robot in a smart environment and creating an *Internet-of-Robotic-Things* (IoRT) [6], where knowledge on context and personal activities is captured via pervasive sensors and wearables.

However, developers of robotic services cannot anticipate the details of smart environments, such as the type of sensor and actuator devices, or the actual personal preferences and context of the humans in these environments. In this paper, we present a platform that facilitates developers to build services according to novel declarative ontological models for generating context-aware, personalized interaction tasks. The platform allows developers to (i) access semantically annotated data from sensors, robots and databases and observe relevant context changes, (ii) set the type and parameters of a human-robot interaction task according to the actual context, and (iii) dispatch these tasks to the robot at the appropriate time.

The remainder of this paper is structured as follows. In Section 2 we present motivational scenarios in three different smart environments. These scenarios outline the envisioned type of services enabled by our platform and highlight the scientific and technological challenges. In Section 3, we derive the requirements for our platform. In Section 4, we present ontologies that capture the problem domain of generating interaction tasks and that allow to declaratively define robot intervention tasks. In Section 5, we present the architectural details of our platform. A prototype implementation and quality attribute evaluation results are presented in Section 6. We discuss the limitations of our current platform and outline opportunities for future work in Section 7. In Section 8, we position our contributions in different strands of related work. We conclude our paper in Section 9.

2. Motivating scenarios

The aim of this research is to design a generic platform architecture that facilitates building context-aware services that automatically generate personalized interaction tasks for a robot deployed in a smart environment. Below, we sketch motivating scenarios in three types of smart environments that illustrate the scope and functionality of our platform and the services deployed on it. Aspects of these scenarios will also be used in the remainder of the paper to explain the internals of the platform.

2.1. Versatile helper in a smart home

The Jones family is a very busy household. They already extensively equipped their house with several sensor appliances, and recently they bought a humanoid robot to help with household chores. The robot interfaces with the smart home (SH) system for better context-awareness.

SH1 - Nutrition monitoring In the evening, after a long day at work, Mr. Jones wants to prepare a dish. His smart fridge detects that he takes out some chicken and a portion of spinach. The robot drives to the kitchen to advise Mr. Jones against preparing spinach since medical recommendations are to maintain a diet with a steady intake of vitamin K when taking blood thinners and Mr. Jones' intake of vitamin K earlier this week was already above average. Mr. Jones asks the robot for an alternative recipe. Given that Mr. Jones has an evening appointment scheduled in two hours, the robot suggests a sauté dish that can be prepared in less than half an hour and that doesn't require advanced cooking skills [7].

SH2 - Homework assistance Joshua, the 8-year old son asks the robot to help him with his homework. The robot invites Joshua to do his homework in his room instead of the living room, because Joshua's sister Olivia will enter in half an hour. The robot is able to track the engagement and attention span via sensors on the body of the child [8], and changes the role it plays accordingly, varying between a friend and a stricter tutor.

SH3 - Therapy monitoring[1] Olivia, 10 years old, was recently diagnosed with diabetes and she still needs to learn how to properly manage her glucose level by injecting an appropriate amount of insulin. Olivia's connected glucose sensor readings indicate a decreasing trend in her blood sugar level [10]. As the robot knows that Olivia is coming back from her tennis class, it infers that this may be due to physical activity. When Olivia enters, the robot kindly asks her how the tennis class went. As dinner will not be ready for another hour but Olivia needs to manage her glucose level, the robot advises her to eat an apple. Once dinner starts, the robot reminds Olivia to carefully count the carbs and to account for the fact that she already had an apple as well as a physical work out.

Challenges All three interaction scenarios leverage on the interpretation of sensor data, but each time the data is interpreted in a different knowledge domain: nutrition, cookery, education. The content (e.g. recipe) and style (e.g. tutor, friend) of interactions are tailored to contextual information that is derived from static sources (e.g. calendar, recipe database) as well as from interpretation of sensor data (e.g. person arriving at home, person starting to cook).

2.2. Companion supporting care staff in a smart nursing home

The cultural, personal and medical background of the residents in a smart nursing home (NH) is very diverse. In its aim to keep up with the highest standards of person-centric care giving, the nursing home has equipped its residents with wearables, has installed several sensors in rooms, corridors and common living rooms and has deployed a mobile companion robot [11].

NH1 - Activity Announcement Every morning, when Bob is sitting in his chair waiting for breakfast, the robot enters his room to announce a selection of today's activities in the nursing home and to read aloud a selection of news highlights. Usually, Bob enjoys the robot to have a more jovial style of greeting him [12], but the robot tones down his enthusiastic style since it detected via the bed sensors that Bob had a restless night. In the room next door, Elisabeth prefers the robot to greet her in a formal way as Mrs. Smith. Since the robot knows that Elisabeth doesn't like quizzes, it doesn't announce this activity to her. Instead, it gives some updates on celebrity news, a topic that wasn't of interest at all to Bob. The

[1]Motivational scenario SH3 was co-created with dietitians and pediatricians [9].

robot increases the speaker volume as Elisabeth is hard hearing and sends a temporarily mute command to the TV in the room.

NH2 - Behavioral Disturbance Management[2] Henri has been diagnosed with dementia. Over the past months, his cognitive capabilities have steadily declined and he has started to show behavioral disturbances typical for persons with dementia. From time to time, Henri starts wandering through the NH and sometimes enters the rooms of other residents. The wandering pattern is detected through IMU data of Henri's wearable [14] and the location tracking system in the nursing home. The robot is sent to Henri to distract him and guide him back to the common living room. Once Henri is seated, the robot starts playing his favorite Beatles song. These personal preferences were previously registered in the personal record of Henri by the care staff.

NH3 - Visitor Information After a fall earlier this week, Elisabeth is diagnosed with a mild concussion. As part of the treatment, the light and sound volume in the room must be kept moderate. On Sunday, her family visits her in her room and a sound sensor detects that the sound level in the room is too high. The companion robot is sent to the room, where it explains that Elisabeth needs sufficient rest and asks politely to keep the volume down.

Challenges The companion robot is shared between multiple services that create very diverse interaction tasks. This requires a component that arbitrates the priority. Note that the set of services changes over time: the service monitoring light and sound in each room is only active during the recovery period of Elisabeth. In each of the proposed applications, the interaction task is based on advanced processing algorithms from sensor data, e.g. activity recognition, detection of wandering, sleep quality assessment, etc. As in the smart home scenarios, all robot interactions are again completely personalized, i.e. both the content and the style (tone, volume) of the robot's verbal interactions are tailored to the resident at hand.

2.3. Receptionist in a smart office

SO1 - Visitor Reception An external participant to a meeting arrives in a smart office (SO) and scans a QR-code at the entrance kiosk. Since the local employee that organizes the meeting is stuck in traffic, a concierge robot [15] drives to the entrance, greets the visitor by name and invites the visitor in his native language to follow him to the meeting room. As the

[2]Motivational scenario NH2 was co-created with NH staff [13].

robot approaches, it detects that the visitor has broken his leg and sits in a wheelchair [16]. Hence, the robot deviates from the company's policy to stimulate physical exercise and will guide the visitor towards the elevator instead of showing him to the stairs. As the elevator arrives at the meeting room floor, the visitor is routed to the coffee corner, because the meeting room is still being cleaned after the previous meeting. After the cleaning staff has left the room, the visitor is asked to follow to the meeting room and is notified about the expected time of arrival of his host.

Challenges Realizing this scenario requires a "visitor welcoming service" to have access to knowledge of the business policy on how visitors should be received (identification, offer something to drink, guide to meeting room). The service needs to reason on static (floor plan) and dynamic (i.e. provided by sensors, such as room occupancy and arrival time of the host) contextual information to determine i) that a robot should be tasked to receive the visitor; and ii) how the robot should complete this task. For the latter decision, personal information is taken into account as well (language, physical condition).

3. Requirements

In this section, we set the requirements for our middleware platform. We start by analyzing the challenges and conceptual flow that are common to all scenarios from the previous section. Then, we discuss functional and non-functional requirements.

3.1. Problem analysis

Realizing the application services outlined in section 2 requires combining technologies and scientific innovations from both the IoT and robotics domain.

From the IoT domain, the pervasive sensing technology and data processing algorithms allow overcoming the physical limitations of a social robot's perception range. A robot cannot observe the entire smart office, care facility or home. Also, body-mounted sensors provide biophysical data to monitor and account for a user's health condition or mood. Such biophysical parameters are close to impossible to measure with robot-mounted sensors.

From the robotics domain, social and assistive robots provide a novel interaction modality. These robots are also an actor with autonomous mobility and actuation capabilities. The scenarios described above demonstrate the various roles that robots can take when interacting with humans in smart (sensorized) environments: as a companion helping to manage chronic diseases (scenario SH3), as an educator explaining you new concepts or helping with homework (SH2), as a guide navigating you through an unknown building (SO1) or as a technological tool assisting caregivers in entertaining and looking after elderly (NH1-NH3).

Despite the diversity in application domains and services, all human-robot interaction scenarios follow an identical conceptual data-driven workflow [17] transforming IoT data streams into personalized interaction tasks for robots, as depicted in Figure 1. Interactions are triggered either directly from events signaled by the smart environment, for instance a user starting to cook (scenario SH1) or an elderly starting to wander (NH2); or from business logic, e.g. for education (SH2) or building facility management (SO1). The interactions are executed by a set of robotic and non-robotic actuators (e.g. elevator, TV, light). Contextual and personal information is used to determine the moment, type and parameters of interactions to be executed. For instance, the domestic robot will educate in the ideal room (SH2), visitors are routed to the coffee corner until the meeting room is vacated (SO1), or the robot's speaker volume are adjusted (NH3).

Realizing the flow of Figure 1 requires a complex integration of multiple technologies and services and proper management of sensitive data. In the IoT domain, this problem has been addressed by introducing middleware platforms that abstract vendor-specific syntax and offer supporting services for accessing IoT data and interoperability between applications [18]. In this paper, an IoRT platform is presented that hides the specific configuration of a smart environment (type of sensors and robot). Developers can use our platform to access contextual and personal information, and combine this information with domain-specific knowledge to determine which kind of task should be executed and when. The low-level logic to plan specific actions can be left to the platform.

People will likely invest in only a single robot for their smart home. Our platform mediates between multiple services and a single robot. Extension to multi-robot systems would require dealing with task allocation, resource reasoning, multi-robot planning and scheduling, see e.g. existing work on cloud-based task control [19] and constraint-based planners [20]. With this increased functionality, the added value of social robots will increase and help to motivate the investment cost of such a robot.

While numerous IoT platforms exist, the Internet-of-Robotic-Things is a nascent research domain and so far, integration between the two domains was mostly limited to focused applications, as outlined in the recent survey by Simoens et al. [21]. The IoRT vision is however grounded in past

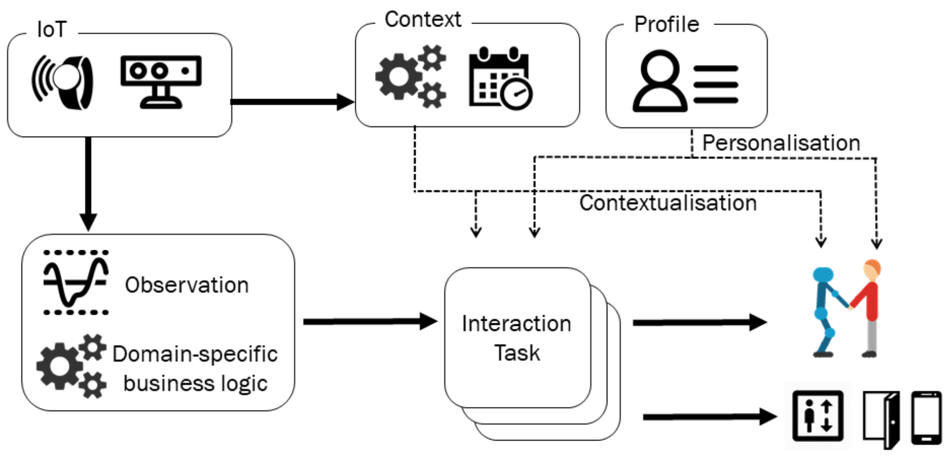


Figure 1: Interaction tasks originate from IoT observations or from business logic. The instantiation and execution of the interaction task both depend on contextual and personal information

research efforts and associated middleware platforms integrating robots and sensor devices have been presented mostly in the context of robot *ecologies*. However, these platforms were mostly oriented towards assistive services, whereas we focus on context and personalization to tailor the style and modality of human-robot interaction. These platforms are further discussed in section 8.

3.2. Functional requirements

To realize a middleware platform that supports the conceptual flow of Figure 1, we formulate the following functional requirements:

- The system should offer a declarative northbound interface to describe the observations and business logic that should be met to trigger a specific interaction task. In this paper, we refer to such a submitted set of rules as a Task Generating Service (TGS). By creating a rich declarative language, we can hide the configuration and low-level details of the sensors and actuators in the IoRT environment to developers of TGS. Developers only need to focus on the *what* and *when* of an interaction task, but not on the *how*.
- The system should internally convert a registered TGS into a data-driven workflow that converts IoT sensor input to actions to be executed by robotic and non-robotic actuators in the smart environment. Once a TGS

is submitted, the platform must start observing the relevant sensors. Actuators that are registered with our platform provide one or more actions. Example actions include moving the robot (SH1), having the robot play a song (NH2), sending a command to the smart home automation system to open a door or mute the TV (NH1).

- The system should be able to interpret raw data using operational context and application domain knowledge. The raw data provided by IoT sensors has no meaning on itself. The platform will thus have to semantically enrich the raw sensor data [22], i.e. make the properties and the device and context in which the data was gathered explicit, possibly link this to static information (e.g. activity schedule in the nursing home, calendar) and reason on the derived higher-level observations to determine where the contextual preconditions of an interaction task are fulfilled. For instance, if an IR sensor detects the presence of a person, this is only useful information if one knows whether this sensor is placed in a meeting room, an entrance hall or the kitchen of a smart home. If the sensor is placed in a kitchen, one needs contextual information (e.g. time of day) and personal information (e.g. age) to accurately conclude that the detected person is an adult who is likely to start cooking and can benefit from the robot giving advice.
- The system should be able to schedule multiple interaction tasks over time for a single robot. Since there can be several TGS running concurrently, the system will need to schedule tasks over time, accounting for deadlines and constraints such as the robot's battery. While the system supports multiple non-robotic actuators, only a single robot is assumed, as motivated in the previous section.
- The system should be able to fill in action parameters and evaluate action filters. The way tasks are converted into actions should be adapted to the person (personalization) and to the context (contextualization). By this we mean both which actions are chosen to fulfill the task, as well as the parameters of each action. For instance, the action "talk about topic" has two parameters: the language to be used and the topic. Similarly, the action "play music" should be avoided at night.

3.3. Non-functional requirements

Modifiability is the most important non-functional requirement for our platform. In this paper, we refer to modifiability as the effort that is required to alter the functionality of the designed platform to realize other scenarios in possibly other smart environments than the scenarios and environments that were put forth in section 2. Note that this modifiability requirement must hold at deployment time, i.e. when installed in a specific IoRT environment, as well as at runtime, i.e. when a additional sensor is installed or when additional Task Generation Services are deployed. An example of the latter is described in scenario NH3 of subsection 2.2: the behavior of the system that was deployed in the nursing home is extended with novel behavior at runtime after a resident suffered a concussion. Specifically we consider a deployed system to consist of two distinct parts: the system core, which does not change no matter the scenario or the smart environment, and the system plug-ins, listed below:

- Task Generating Services. These TGS encapsulate the most essential business logic of the application so the set of TGS will differ between application domains.
- (Sensor) input sources. Sensor input sources may differ between environments. When using a sensor adapter framework [23], especially one with many commercial sensors already integrated, new sensors have no impact on the system.
- Action Executors. It must also be possible to attach new types of actuators, each bringing a set of actions.
- Action filters. These filters serve to shape a task execution strategy by managing the dynamic availability of actions. For instance, loud actions such as playing a song should be avoided while people are sleeping.

To evaluate the modifiability, we propose the Quality Attribute Scenarios [24] below.

- **Source of stimulus:** Developer.
  Stimulus: Additional TGS are declared to the platform.
  Artifact: The system's reasoning engine.
  Environment: System under normal operation.
  Response: The declared TGS is converted to a data flow that produces interaction tasks.
  Response measure: New TGS is operational in less than 10 seconds.
- **Source of stimulus:** Developer.
  Stimulus: New Action Executor is added to the system.
  Artifact: The system's repository of action executors.
  Environment: System under normal operation (at runtime).
  Response: The actions of the new executor are listed in the Action Repository and available for usage by the planner.

Response measure: A plan can include the new actions within 5 seconds.

- **Source of stimulus:** Developer.
  Stimulus: Action filter is added.
  Artifact: The system's reasoning engine.
  Environment: System under normal operation.
  Response: The new filter is added to the filter array.
  Response measure: The new filter starts filtering actions within 10 seconds.

4. Ontological Models for Contextualization and Personalization

We have conceived three ontological design patterns that offer a vocabulary for developers to *declaratively* define which robot interactions should be performed based on the observed IoRT context and how these interactions should be personalized according to the profile of the end-user.

An ontology defines the concepts within a domain, what their attributes are and how the different concepts are linked to each other. Axioms and rules in the ontology constrain the characteristics and possible interpretations of the concepts and relations defined in the ontology. In the next three subsections, we detail the three ontological models. To support the IoRT platform, the novel concepts defined in these three models, together with common concepts in IoRT applications are grouped in a suite of ontologies. This suite is presented in Section 4.4.

4.1. The observation pattern: capturing context data through sensors and linking to events & interaction strategy

This pattern captures reactive scenarios in which interaction tasks result from interesting events that are derived from the IoT sensor data. The interaction is an immediate response to the observation or event. For instance, a visitor scanned a QR code at the entrance (SO1), or an elderly with dementia has entered a wrong room (NH2). The observation pattern is visualized in Figure 2. The generic concepts of the pattern are shown in white, while a specific example instantiation of the pattern based on the NH2 scenario detailed in Section 2 is shown in gray.

The observation pattern consists of five classes, namely `Observation`, `Symptom`, `Fault`, `Solution` and `Task`. An `Observation` is any piece of information observed by a sensor, device or robot. A `Symptom` models specific phenomena that are detected in the `Observations` by taking into account the current context and situation. Queries or axioms can then be

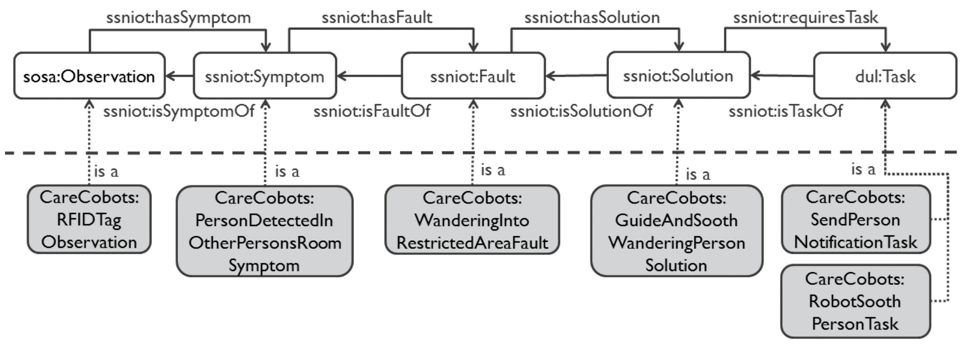

ssniot:hasSymptom
ssniot:hasFault
ssniot:hasSolution
ssniot:requiresTask
sosa:Observation
ssniot:Symptom
ssniot:Fault
ssniot:Solution
dul:Task
ssniot:isSymptomOf
ssniot:isFaultOf
ssniot:isSolutionOf
ssniot:isTaskOf
is a
is a
is a
is a
is a
CareCobots: RFIDTag Observation
CareCobots: PersonDetectedIn OtherPersonsRoom Symptom
CareCobots: WanderingInto RestrictedAreaFault
CareCobots: GuideAndSooth WanderingPerson Solution
CareCobots: SendPerson NotificationTask
CareCobots: RobotSooth PersonTask

Figure 2: Visualization of the ontological concepts and relationships that make up the observation pattern. Concepts are visualized as rectangles while relationships are visualized as arrows. The white concepts and solid arrows represent the pattern, while the gray concepts and dotted arrows below the dashed line visualize an example instantiation of the pattern. The example is based on the NH2 motivating scenario. The prefixes indicate the namespace of the ontology module the concepts and relationships originate from (see Section 4.4).

defined that detect undesirable combinations of `Symptom`s and classify them as `Faults`. These detected `Fault`s can then be coupled to `Solution`s that resolve them. Finally, a `Solution` can be mapped onto one or more `Task`s that need to be performed to reach this solution. As such, the various high-level events that are detected within the IoRT environment, can easily be linked to (robot) interactions that need to be performed. As all these concepts are linked together by the indicated relationships, e.g. `hasFault`, `hasSolution`, etc. It is always possible to trace the task back to the original observation that caused it.

We illustrate this pattern using the NH2 scenario. Every time the wearable of an elderly is located by a receiver in the nursing home, a `RFIDTag Observation` is registered. Out of the context it is derived that the receiver that made this `RFIDTagObservation` is located in a room that does not belong to this elderly. As such a `PersonDetectedInOtherPersonsRoom Symptom` is derived. As the person who lives in this room is not present and this elderly has a history of wandering into this room, a `WanderingInto RestrictedAreaFault` is derived. The defined solution for these types of faults is a `GuideAndSoothWanderingPersonSolution`, which means that the elderly should be lead to a safe area and should be soothed such that he might stop wandering. This solution triggers two tasks at first, namely a `SendPersonNotificationTask` to notify a nurse of what was detected and which solution is being undertaken by the robot and a `RobotSoothPerson`

`Task` to trigger the robot to guide the elderly into the common room and sooth him/her.

4.2. The personalization pattern: capturing profile and context data to personalize the (robotic) interactions

The concepts of this pattern can be used to declaratively define how the type, style and content of robot interactions must be adapted to the profile and preferences of a person. The personalization pattern is visualized in Figure 3 with the generic concepts in white and example instantiations for the NH2 scenario in gray.

The personalization pattern allows to associate each `Person` with a `Profile`. This `Profile` is split up into a `BasicProfile` and a `RiskProfile`. The first models administrative, biological, psychological, and sociological information. Our current ontology restricts the sociological profile to nationality and spoken language but can be extended in the future to capture other notions of a user's culture and hence become "culturally competent" [12]. Capturing cultural knowledge is the subject of ongoing work, e.g. in the CARESSES project [25], and once such models become established, they may help to adapt the robot's way of interacting (gestures, choice of phrases, etc.) to the user's cultural identity. The `RiskProfile` is defined by classification axioms and rules. This allows a reasoner to automatically obtain the risk profile of a person by reasoning on the information in the basic profile.

A `Person` is also correlated with his or her `Preferences`, e.g., `Language Preference` or `MusicalPreference`. On the one hand, these preferences can be explicitly stated. On the other hand, these preferences can be derived through reasoning on the profile information. Finally, the `Preferred Interactions` can be specified, which indicate which mode of interaction a person prefers. These interaction are then linked through the `requires Action` relationship to the specific `Actions` which can be performed by robots, devices and people and which satisfy the specified preferred interaction mode.

Each `Task` is associated with one or more `Actions` that fulfill its requirements. An `Action` can be associated with `Parameters`, which can be chosen based on the specified preferences. This is indicated through the `influencedBy` relationship. Finally, the `PersonalizationRequest` concept models a request to personalize a particular `Task` or `Action`.

To explain the application of the personalization pattern, Figure 3 visualizes an example instantiation of the pattern for the NH2 scenario where the robot sooths a wandering elderly after first guiding the elderly out of the wrong room. Previously, the care staff has entered profile information about

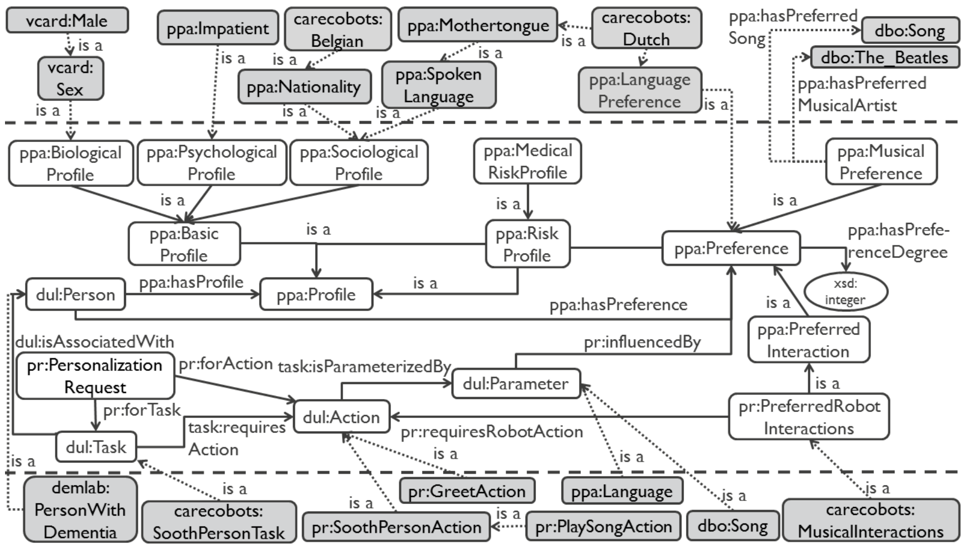


Figure 3: Visualization of the ontological concepts and relationships that make up the personalization pattern. Concepts are visualized as rectangles while relationships are visualized as arrows. The white concepts and solid arrows represent the pattern, while the gray concepts and dotted arrows below the dashed line visualize an example instantiation of the pattern. The example is based on the NH2 motivating scenario. The prefixes indicate the namespace of the ontology module the concepts and relationships originate from (see Section 4.4).

all residents in the system, i.e., administrative, e.g. date of birth, biological, e.g. `Sex`, psychological, e.g. `Impatient`, and sociological information, e.g., `Nationality` and `SpokenLanguage`. The staff also indicated which of the residents prefer `MusicalInteractions` with the robot, and which are their `MusicalPreferences`. This `MusicalPreference` is defined by linking to information contained in DBPedia about `MusicalArtists`, e.g. The Beatles, and `Songs`, e.g., Let it be.

By querying and reasoning on the semantic information, i.e. following and interpreting the links, the `SoothPersonTask` is personalized in two ways.

First, there are several robot interactions possible to sooth a person. Based on the fact that `MusicalInteractions` are specified as preferred mode of interaction for the person in our example, we can derive that a `PlaySongAction` is an appropriate `SoothingAction`. To enforce this, the following semantic rule is specified in the domain-specific ontology:

```
hasProfile(?x,?y) ∧ PreferredRobotInteractions(?y) ∧
    MusicalInteractions(?y) ∧ isAssociatedWith(?t, ?x) ∧
       requiresAction(?t, ?a) ∧ SoothPersonAction(?a)
                                          → PlaySongAction(?a)
```

Second, the selected `PlaySongAction` is linked via the `parameterizedBy` relation to the parameter `Song`. This parameter is then in turn related to the `MusicalPreference` concept through the `influencedBy` property. The specific song can then be chosen based on the `MusicalPreference` defined for this person, e.g., in this case a song by The Beatles should be played. Additional application-specific algorithms can decide how to use this information, e.g. to randomly pick a song of the discography or to just go with one of the preferred songs, e.g. 'Let it be'.

4.3. The opportunity pattern: capturing context data to determine windows of opportunity

Next to the data-driven tasks that are derived directly from the incoming sensor data and captured by the observation pattern, a second type of tasks are interactions tasks that are desirable during a certain window of opportunity. Examples in our motivational scenarios are: the robot that needs to help a child with his homework (SH2) or the companion robot that needs to read today's activities aloud (NH1). The start of the window is typically a fixed moment in time, e.g., when the child gets home from school or when the elderly has woken up, whereas the end of the window can be more flexible.

These windows of opportunity depend on particular `ContextualPreconditions` that should ideally be fulfilled in order to be able to perform the task. The opportunity pattern was created to model these contextual dependencies and is shown in Figure 4, along with its relations to the two previous patterns. We adopt the context categorization proposed by Zainol, et al. [26]. It discerns between three types of context, i.e. `Extrinsic`, `Interface` and `Intrinsic Context`. The first refers to information derived from the physical world or environment. This information is often derived directly from sensors through observations, i.e., by using the observation pattern described in Section 4.1. `Intrinsic Context` denotes the attributes of a user or an agent, i.e., profile information, preferences and emotional state.

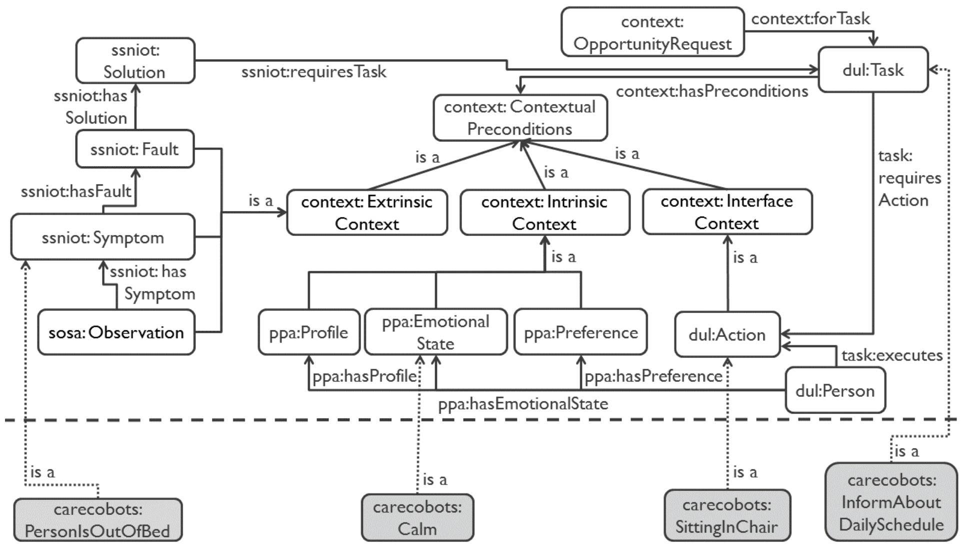


Figure 4: Visualization of the ontological concepts and relationships that make up the opportunity pattern. Concepts are visualized as rectangles while relationships are visualized as arrows. The white concepts and solid arrows represent the pattern, while the gray concepts and dotted arrows below the dashed line visualize an example instantiation of the pattern. The example is based on the NH1 motivating scenario. The prefixes indicate the namespace of the ontology module the concepts and relationships originate from (see Section 4.4).

This information is often directly provided by the user or derived from static data sources, i.e., by using the personalization pattern described in the previous section. Finally, the `Interface Context` refers to activities that the users are performing within the environment. This context thus forms the connection between the `Intrinsic` and `Extrinsic Context`. This division in context types allows to prioritize between context conditions that should be fulfilled in order to perform the task.

An example of the opportunity pattern based on the NH1 motivating scenario is visualized in Figure 4. The robot needs to find a window of opportunity to perform the task of reading today's activities to an elderly. The following `ContextualPreconditions` need to be fulfilled. First, the elderly needs to be out of bed, which is a `Symptom` derived based on observations from sensors by the observation pattern and is thus an instantiation of an `ExtrinsicContext` precondition. Second, the person needs to be `Calm`, which is an example of an `IntrinsicContext` precondition. Third, the elderly is `SittingInChair`, which is an `InterfaceContext` precondition. When the deadline for the action to be performed is drawing close, the

IoRT platform can use this division of preconditions to prioritize which one is more important to be fulfilled, while still performing the action within the time frame. For example, to read the daily activities, it is more important that the person is awake and out of bed (extrinsic) than the fact that he/she is currently not busy performing a task (interface) or calm (intrinsic).

4.4. The IoRT ontology

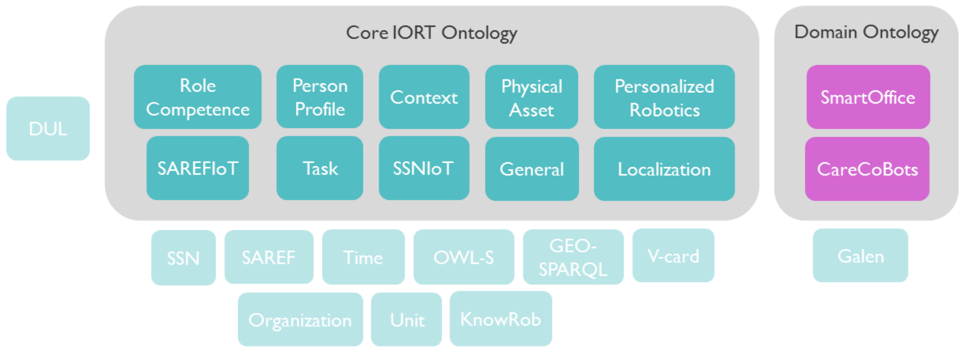


Figure 5: The different modules of the IoRT Ontology.

The IoRT ontology, available online[3], captures all concepts that are relevant for personalized, context-aware task generation. To ensure that the ontologies and patterns capture all the information and (implicit) knowledge and processes that the domain experts use on a daily basis, an intricate co-design process was set up with various stakeholders within representative application domains of the IoRT platform. In these workshops, domain experts needed to play out scenarios and resolve problems as if they were the intelligent IoRT platform. The information and knowledge they used to decide which actions to perform were captured in decision trees. The nodes in the decision trees formed the concepts and relationships for the ontology, while their structure was translated to rules and axioms. Concepts, relationships and structures that often re-occurred across different application domains and workshops, gave rise to the construction of the patterns and the concepts included in them. The adopted co-design process is detailed in Ongenae, et al. [27].

Instead of conceiving the ontology as one monolithic model, the IoRT ontology actually consists of a suite of ontological models that can be in-

[3]`https://github.com/IBCNServices/Accio-Ontology/tree/gh-pages`

corporated in IoRT applications depending on the required knowledge. The models are interlinked to each other by defining relationships between their respective concepts. By designing the ontology in a modular fashion, re-use is facilitated and (domain-specific) extensions can be easily achieved. An overview of the various modules is shown in Figure 5. The modules were devised by first making a distinction between the IoRT core and domain-specific ontologies.

The core ontologies model general information of interest to all applications built upon the IoRT platform, regardless of the use case domain. Adding too many axioms to the core ontologies that constrain the possible interpretations of concepts was especially avoided, unless there was very wide agreement about the constraint amongst the stakeholders involved in the co-creation process. This facilitates cross-domain applicability of the core ontologies and allows easy extension without contradicting with the knowledge already contained in these ontologies. The various core modules are also aligned to Dolce Ultralight upper ontology[4] to support broad semantic interoperability to other domain-specific ontologies.

Domain ontologies contain knowledge specific to a particular domain (e.g. office or medical), such as the specific profile information of a person relevant within a particular domain (e.g. medical parameters and diagnosis), robotic actions only of interest in a particular domain (e.g. soothing a person), etc. All concepts in the domain-specific ontologies are always subclasses of concepts in the core ontologies. Relationships to the information captured in the core ontologies are also defined. New domain ontologies can thus easily be defined by extending the core ontologies. The domain-specific ontologies can easily import and extend a specific core module instead of importing the whole suite of IoRT ontologies. This facilitates re-use and allows application components to only use a subset of the suite of IoRT ontologies to perform the domain-specific reasoning. A smaller, focused ontology is also easier to interpret and extend with new concepts, relations and definitions.

The modules were further refined by identifying the different categories of knowledge within the core, e.g., devices, people and robotics. A module was created for each. The domain-specific ontologies are made ad hoc for particular use case domains.

The modules are linked to each other using the OWL import mechanism.

[4]`http://www.ontologydesignpatterns.org/ont/dul/DUL.owl`

Mapping languages, such as RML[5], can easily be used to enrich the available and incoming data with semantic information.

This modularization process resulted in the following core ontologies (the prefixes of the namespaces of these ontologies are mentioned between brackets):

- **General (general)** This ontology models generally re-usable concepts, e.g. IDs, names, priorities and statuses. It imports and extends the OWL Time[6] W3C recommended ontology to represent temporal information.
- **SSNIoT (ssniot)** The W3C Semantic Sensor Network (SSN)[7] ontology describes in a general way sensors and their observations, the involved procedures, the studied features of interest, the samples used to do so, and the observed properties, as well as actuators. Our SSNIoT ontology extends SSN with the observation pattern explained in Section 4.1. We also modeled sensors, actuators, observations and properties which commonly occur in smart environments, e.g., motion, door, presence, environmental (humidity, temperature, light intensity) and sound sensors.
- **SAREFIoT (sarefiot)**: This ontology imports and extends the Smart Appliances REFerence[8] standardized ontology. SAREF is a shared model of consensus that facilitates the matching of existing assets (standards, protocols, data models, etc.) in the smart appliances domain. Again this ontology was extended with devices, objects, device & appliance functions and device states & properties that are prevalent within the IoRT domain, e.g., televisions or interaction screens.
- **Localization (local)** This ontology models localization information, both from a conceptual (e.g. floors, buildings, rooms and hallways) as well as a geospatial viewpoint (e.g. geometries, surfaces, points and arcs). It imports the GeoSPARQL[9] ontology to model the latter and link it to conceptual entities to describe their geometry.
- **PhysicalAsset (asset)** This ontology captures physical objects in daily environment, such as items, furniture and vehicles.
- **PersonProfile (ppa)** This ontology builds further upon the ontology mapping of the vcard specification (RFC6350)[10] and the W3C Organi-

[5]`http://rml.io/`

[6]`https://www.w3.org/TR/owl-time/`

[7]`https://www.w3.org/TR/vocab-ssn/`

[8]`https://sites.google.com/site/smartappliancesproject/ontologies/reference-ontology`

[9]`http://www.opengeospatial.org/standards/geosparql`

[10]`https://www.w3.org/TR/vcard-rdf/`

zation ontology[11] to model people, their relationships to each other, the organizations they work for or are connected with and their profile information. These models were extended with the personalization pattern introduced in Section 4.2. Our ontology also contains concepts and relationships that allow to model the (trust) relationship between two people, e.g. personal, colleague, therapeutic or family.

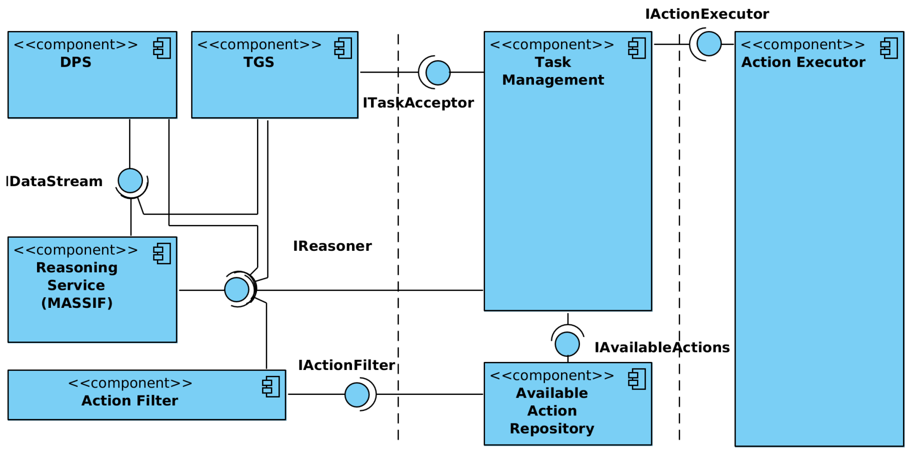
- **RoleCompetence (rolcomp)** This ontology associates e ach person with his or her roles and competences. A role is defined as a collection of needed competences. This allows reasoning about capabilities and abilities of people in order to assign tasks.
- **Context (context)** The context ontology mainly models correlations between concepts of the above ontologies, e.g. it associates devices and people with their location, devices and items with their owners and people with the events they participate in. It also contains the opportunity pattern explained in Section 4.3.
- **Task (task)** This ontology imports the OWL-S Process ontology[12] to model processes with their associated pre- & post-conditions, input, output and effects. A task is introduced as a sub-concept of an OWL-S Process. The task can be a potential one or an actual one. The latter is further split up into planned & unplanned tasks. Several relationships to correlate people with tasks are also defined, e.g., `executedBy` or `assignedTo`. Tasks can also define which (robotic) capabilities or competences are required to execute it.
- **PersonalizedRobotics (pr)** This ontology builds upon the KnowRob ontology suite[13] to model robots, their capabilities and the action they perform. It was extended with concepts and relations to model the affinity of a person towards robots (e.g. neutral, agressive, positive), robot interaction scenarios that occur within an IoRT environment (e.g. playing music, telling a story, guiding a person, helping with homework) and the possible reactions people can have to them. Relations to correlate the robotic systems to all the concepts introduced in the previous ontologies, e.g. sensors, items and tasks, were also introduced.

[11]`https://www.w3.org/TR/vocab-org/`
[12]`https://www.w3.org/Submission/OWL-S/`
[13]`http://www.knowrob.org/ontologies`

5. Implementation

5.1. Platform components

The architecture of our platform is shown in Figure 6. The platform is conceived as a middleware onto which declared Task Generation Services are deployed.



<<component>> DPS
<<component>> TGS
<<component>> Task Management
<<component>> Action Executor
IActionExecutor
ITaskAcceptor
IDataStream
IReasoner
<<component>> Reasoning Service (MASSIF)
IAvailableActions
IActionFilter
<<component>> Action Filter
<<component>> Available Action Repository

Figure 6: Component diagram of our middleware platform.

The dashed vertical lines in Figure 6 demarcate which components are responsible for task generation, task management and task execution respectively.

The semantic Reasoning Service supports all ontological reasoning. It provides the declarative interface in terms of the IoRT ontology that was introduced in section 4.4. The Reasoning Service takes as input data streams that are produced by Data Processing Services (DPS). These DPS transform sensor data to higher-level information. For instance, a localization DPS may combine readings from a wearable with the information from a presence sensor to provide an accurate location of a resident in a nursing home.

Task Generation Services use semantic reasoning and these enriched data streams to submit interaction tasks to the Task Management component. The task is scheduled and a planner generates a list of actions to be executed by robotic and non-robotic Action Executors. The planner avails of an Available Action Repository, which filters the set of actions that is available in the current context.

In the next sections, we provide more details on each of the three parts in Figure 6. Task generation is detailed in section 5.2, task management and the available action repository are detailed in section 5.3 and we conclude with a discussion on the action executors in section 5.4.

5.2. Definition and deployment of a semantic Task Generation Service

Developers of TGS first instantiate one or more of the ontological patterns discussed in Section 4. To register a semantic service, the developer must provide the following information:

1. The parts of the IoRT ontology that will be used by the service. By limiting the size of the model employed in the service, more performant semantic reasoning can be performed.
2. Filter rules that indicate the type of semantically enriched data that should be used as input for the TGS. These filter rules can range from simple indications of ontological concepts to intricate axioms. For example, if the service wants to receive all the observed RFID tags by the system as input, it registers a filter rule consisting of one concept, i.e., `RFIDTagObservation`. An example of a more complicated filter rule indicating that a TGS is interested in all data observed in the rooms of the elderly is shown in Listing 1.
3. The application logic of the TGS, defined by SPARQL Queries. An example instantiation of the observation pattern for the NH2 scenario is given in Appendix A.2.

```
Observation and (ObservedBy some (hasLocation some Room))
```

Listing 1: Example of a rule that filters all observation coming from a particular room

The implementation required for the services is minimized by providing a GUI to register the above information to the platform. A screenshot of the GUI can be found in Appendix A.

By employing this methodology, domain experts and analysts can focus on defining the background knowledge and specifics of the context and use case domain that the IoRT Platform is being deployed in by instantiating the ontology design patterns. The developers of the services can then use these specifications as input to define the task generation services in an easy fashion. As such, an effective separation of operational logic and domain knowledge is achieved. In the following section, we discuss the operational logic of the platform to set up a registered TGS.

5.2.1. Modular data-driven workflow for TGS

To execute the registered TGS, the IoRT platform first needs to semantically enrich data and then perform semantic reasoning on it. Both static data stored in repositories as well as sensor data must be transformed to semantic data that adheres to the concepts in the IoRT ontology.

For static knowledge, the Stardog[14] graph database is integrated. This graph database uses the created IoRT ontology as model and contains profile information of the users, information about the deployed sensors and devices, and descriptions of the available robots. Using RML[15], we create mappings that automatically transform the data in existing databases to semantic data adhering to the IoRT ontology.

To easily integrate, interact with and discover the multiple sensors and non-robotic actuators in the environment, the IoT sensor integration platform DYAMAND[16] [23] is used. A plug-in for the DYAMAND platform was written that maps the internal model of DYAMAND on the SSN ontology. This enables a standardized representation of the data outputted by DYAMAND in JSON-LD format.

We use the MASSIF platform (ModulAr, Service, SemantIc & Flexible) Platform [28] to realize scalable semantic reasoning on this enriched data. MASSIF allows the development & deployment of modular semantic reasoning services which collaborate in order to allow scalable & performant semantic reasoning on the enriched data. The heart of MASSIF is the Semantic Communication Bus (SCB) that facilitates data dissemination and coordination between the various semantic reasoning services. The principle is illustrated in Figure 7.

MASSIF constructs a semantic service from the SPARQL queries declared by the developer. This service automatically contains the indicated modules of the IoRT ontology and loads all the available background knowledge that pertains to these ontologies from the Stardog graph database. The provided filter rules are registered on the SCB. The semantically enriched data is picked up by MASSIF and pushed towards the SCB. The SCB will then perform semantic reasoning to decide whether the data fulfills any of the registered filter rules. When a match is found, the data is forwarded to the services that have registered this rule. When new data is forwarded to a service by the SCB, the defined SPARQL queries are executed. By pro-

[14]`http://www.stardog.com/`

[15]`http://rml.io/`

[16]`https://dyamand.ilabt.imec.be/public`

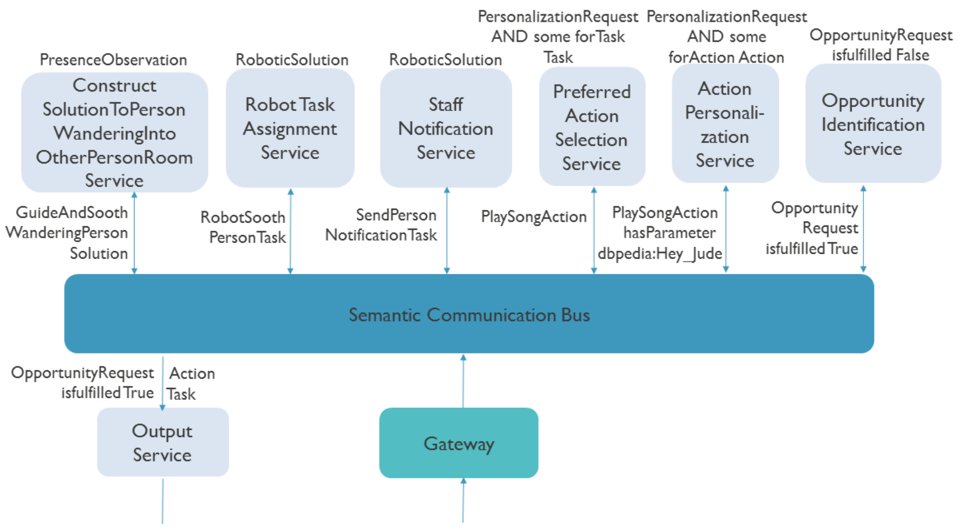

PresenceObservation
Construct SolutionToPerson WanderingInto OtherPersonRoom Service
RoboticSolution
Robot Task Assignment Service
RoboticSolution
Staff Notification Service
PersonalizationRequest AND some forTask Task
Preferred Action Selection Service
PersonalizationRequest AND some forAction Action
Action Personalization Service
OpportunityRequest isfulfilled False
Opportunity Identification Service
GuideAndSooth WanderingPerson Solution
RobotSooth PersonTask
SendPerson NotificationTask
PlaySongAction
PlaySongAction hasParameter dbpedia:Hey_Jude
Opportunity Request isfulfilled True
Semantic Communication Bus
OpportunityRequest isfulfilled True
Action Task
Output Service
Gateway

Figure 7: Illustration of how the MASSIF Platform and design patterns are exploited to easily create personalized robot interactions. The light blue squares represent services. Above the services, their filter rules are denoted. Below an example output is shown.

cessing one published event at a time, the performance of the SCB can be guaranteed and it also allows duplication and scalability of the SCB. Moreover, the MASSIF Platform also contains several algorithms to optimize the reasoning performance in the individual semantic services. A service is also able to push its conclusions back on the SCB. As such, collaboration between the services can be realized through loose coupling.

Besides the application-specific TGS, our IoRT platform comes with four generic services that also communicate over the SCB:

- `PreferredActionSelectionService`: This service takes as input a `PersonalizationRequest` for a particular `Task`. This request can for example be sent by the Available Action Repository as explained in Section 5.3.2. The SPARQL query that makes up the business logic of the `PreferredActionSelectionService` is shown in Appendix A.3. This SPARQL query triggers reasoning over the instantiations of the personalization pattern by the developers, as explained in Section 4.2. The reasoning selects the most appropriate action to fulfill a particular task based on the current context and profile of the person for whom the task is executed and returns it as result of the service.

- `ActionPersonalization`: This service also takes a `Personalization Request` as input, but for a particular `Action`. It uses a similar

SPARQL query to personalize the parameters of the action according to specified instantiations of the personalization pattern. As a result, the service outputs an `Action` with personalized parameters.

- `OpportunityIdentificationService`: This service takes as input an `OpportunityRequest`, which is not fulfilled yet. It analyzes the contextual preconditions of the task that is associated with this request as specified in the opportunity pattern instantiated by the developers. As a result, it registers filter rules with the SCB that indicate that the service is interested in this context. Each time, new context arrives in the service, a SPARQL query is executed to derive whether the contextual dependencies of the task have been fulfilled. When this is the case, an `OpportunityRequest` is outputted, which has been fulfilled.

- `OutputService`: This service takes as input all the generated `Actions` and fulfilled `OpportunityRequests`, and submits a task to the Task Management part of our platform in the JSON format detailed in Section 5.3.1.

5.2.2. Instantiations for the Smart Nursing Home motivational scenario

To illustrate how TGS, MASSIF and the design patterns can be used to easily create tasks based on the observed context, consider the smart nursing home NH2 use case outlined in Section 2.2. The resulting services are shown in Figure 7.

To execute the example **observation pattern** specified in Figure 2, the user first creates a TGS `ConstructSolutionToPersonWanderingIntoOther PersonRoomService` which is interested in presence (e.g. the RFID tag being worn by the elderly) information registered in the rooms of elderly, as expressed by filter rule 1. The SPARQL query for this TGS is shown in Appendix A.2. By performing semantic reasoning on the incoming observations and the designed pattern, the service automatically derives that a `WanderingInto RestrictedAreaFault` has occurred, because an elderly has wandered into the room of another elderly. The service outputs a `GuideAndSoothWandering PersonSolution` to resolve the situation. By studying the defined pattern, the developer creates two additional services to create the tasks that achieve this solution, i.e., `RobotTaskAssignmentService` and `StaffNotificationService`.

As explained previously, two generic services were created, i.e. `Preferred ActionSelectionService` and `ActionPersonalizationService` to trigger the reasoning over the instantiations of the **personalization pattern** by the developers. If we consider that the `PersonalizationRequest` pertains to a `SoothPersonTask` and the personalization pattern example specified in Figure 3 is used, it can be derived that these two services will result in the generation of a `PlaySongAction` with as parameter a Beatles Song, e.g., Hey Jude.

5.3. Task Management

In this section, we detail how the platform handles the execution and monitoring of interaction tasks. The main responsibility of the Task Management is to transform these tasks into a set of personalized actions to be executed by the Action Executors, the southbound interface of our platform that will be described in section 5.4. Since IoRT environments are dynamic due to the presence of humans, our platform includes the necessary measures for replanning and updating a list of available actions.

5.3.1. Task description

Tasks are received in JSON format that is produced either directly by a non-semantic TGS or indirectly from a semantic TGS of which the output is converted by the semantic OutputService. An example JSON object is shown in Appendix B.1.

The Task JSON object has only two mandatory fields: a task ID and a goal state (list of predicates) to be reached. The goal state is provided in Problem Domain Definition Language (PDDL), a popular domain-independent logic-based formalism [29]. The goal must be defined in terms of the predicates provided at design time. These predicates are specific to the scenario. This concept is explained in more detail in section 5.3.4. Optional fields depend on the nature of the task and may include a start time before which the task should not be executed, a deadline or a priority. By allowing TGS to submit tasks with a start time in the future, the scheduler is able to better forecast the demand for the robotic actuators and schedule battery charging cycles accordingly.

A TGS can update the task objects it created. This is typically the case when tasks are generated according to the opportunity pattern of Section 4.3. The TGS may use historical knowledge to initially submit a task with the expected (future) start time of the window of opportunity. This start time may be revised, even multiple times, as new information becomes available. During the window of opportunity the priority may be gradually increased as time elapses to ensure the task gets executed.

5.3.2. Personalized actions with dynamic availability

Each Action Executor announces to the platform a JSON with the set of actions that it can execute. This message contains a PDDL-formatted definition of its parameters, pre-conditions and results. A PDDL example of an action is shown in Listing 6.

Besides the description in PDDL syntax, needed by the planner, the announcement message also contains information about the functional and personalization parameters. This information binds the names of the parameters to their ontological concepts which is required in order to do the action personalization as well as action filtering. Personalization parameters are parameters that change the behavior of the action but do not affect the preconditions and effects of the action. Examples are the specific song to be played by a *PlayMusicAction* (see Listing 6), the recipe to propose to the user who wants to prepare food or the language in which to greet a visitor.

The Available Action Repository (AAR) listens to the actions that are announced by the Action Executors. When contacted by the planner, the AAR will provide a filtered list of available actions in a given context. This is achieved through an array of filters. Third party filter plug-ins are also supported here.

```
{ "ActionName":"PlayMusicAction",
  "PDDL": "...",
  "PersonalizationParameters": {
    "song" : {
      "URI": "http://dbpedia.org/resource/Song",
      "doc": "Optional documentation useful for debugging"},
    "language" : {
      "URI":"http://dbpedia.org/ontology/language"},
    "soundLevel" : {
      "URI":"http://www.w3.org/2001/XMLSchema#integer"}
  },
  "FunctionalParameters": {
    "Patient" : {
      "URI" : "http://dbpedia.org/ontology/person",
     }
   }
 }
```

Listing 2: Example of the JSON string used to announce the *PlayMusicAction*. The PDDL field has been omitted for brevity but a full example can be found in Appendix B.2.

5.3.3. Scheduler

The scheduler orchestrates the allocation of resources over time to maximize the number of tasks successfully completed, weighted according to the task priorities.

The received tasks are first sorted according to their priority and for each task the planner (discussed in the next section) is contacted to produce a tentative plan that allows to make an estimate of the task duration. Then, earliest deadline first (EDF) with task priorities scheduling is applied, with backfilling of tasks where possible. The scheduler component can be easily configured with other strategies to include more advanced scheduling approaches.

If a task cannot be scheduled, the scheduler will notify the corresponding Task Generation Service, which may decide to cancel the task and submit an alternative task.

5.3.4. Planner

The planner uses a modified version of Fast Downward [30]. Like any typical PDDL based planner it requires a domain and problem, both PDDL files, which describe the world, the actions and the goal. The current planner supports PDDL 2.2 level 1 and various parts of higher versions. The set of PDDL predicates, types and objects are defined at design time. Actions are also defined at design time but are disabled unless an Action Executor is available to execute it.

A PDDL based planner has a declarative model of the world defined by predicates. By chaining actions together it is possible to realize the requested goal while meeting the preconditions of each of the preceding actions. When a planning request is made the Available Action Repository (AAR) is queried for a list of available actions. Combining the resulting action list with the relevant domain knowledge and with the goal string received from the TGS, the domain and problem PDDL files are generated. Using these files the planner is able to generate a plan which is outputted as an ordered list of actions.

Since the personalization parameters of the action definition have no effect on the planning process, we are still able to use highly optimized planners proposed in other research. The planner, which ignores the personalization parameters, produces a plan in the form of an ordered list of actions which is then passed to the semantic Action Personalization Service.

In order to be able to provide timing estimates for tentative plans, which are often requested by the scheduler, a simple timing estimation component is built into the planner service. This timing component can provide a timing estimate for a parameterized action (e.g. (MOVE ROBOT KITCHEN HALL)). It does this by consulting the history of executed actions. If this action, with these parameters, has been executed before it will return the average historical execution time. If the action has been executed but with different parameters the average execution time of this action across all parameters is returned. Otherwise a default value is returned. This simple implementation allows for progressively better timing estimates. We leave it as future work to implement a more advanced duration estimator that accounts for up-to-date context information.

5.3.5. Action Dispatcher

The Action list produced by the planner is sent to an action dispatcher that calls the appropriate action executor modules while traversing it. In a dynamic environment, it is likely that the assumptions made at planning time are no longer valid at execution time, especially for plan of multiple long actions. For instance, by the time a robot arrives at the room of a nursing home resident, the resident may have entered the bathroom or left his room. Moreover, actions may fail during their execution.

The main responsibility of the Action Dispatcher is thus to monitor task progress and trigger a replanning if needed. If an action is completed, the dispatcher will check if the preconditions of the next action in the list are all fulfilled and if not, stop the plan and trigger a replanning. In case an action fails, the dispatcher might retry the action or mark the action as failed. The dispatcher contacts the AAR to temporarily remove the failed action from the available action pool and then requests a replan from the planning service. Since the offending action is not in the list of available actions we are guaranteed that it will not be in the plan. If no plan can be found the platform will inform the TGS about that the task failed.

5.3.6. Sequence diagrams

In this section we will demonstrate the interaction between the platform components using a concrete example. We will use the NH1 and NH2 scenarios described in section 2.2, where a companion robot is used to announce activities to the elderly of a nursing home and to sooth a person with dementia who is exhibiting a behavioral disturbance. We assume two separate TGS are deployed, for generating announcement and soothing tasks respectively. Two Action Executors are connected to our platform: one to send messages to the nurse's pager, and one for the social robot.

Activity announcement. The sleep sensors in Bob's mattress not only measure his sleep quality, they also detect when he wakes up. The sensors publish the event and it is picked up by the TGS for activity announcement. Upon receiving this event the TGS will publish the Announce task (see Listing 5). The platform then executes the sequence diagram of Figure 8. The task is submitted with a deadline of 7.20 AM, ten minutes before breakfast. The scheduler requests a tentative plan (steps 2-8 in Figure 8). When the scheduled execution time approaches the scheduler will ask the planner for a plan that is up-to-date with the current context (steps 11-17). It will then push this plan to the dispatcher which will execute it by calling the appropriate (robotic) Action Executor (steps 18 and 19).

Updating the plan. While the robot is moving to the elderly person's room, it is detected that he moves into the bathroom. For obvious privacy reasons the robot is not allowed in the bathroom so the *moveToPersonInRoom* action can no longer be completed. The fall-back strategy has been set to *early fail* so the task is marked as failed. The accompanying sequence diagram can be seen in Figure 9.

5.4. Action Executors

The southbound interfaces of our platform are the *Action Executors*, which encapsulate lower-level control logic. The Action Executors are responsible for managing and executing actions. They announce the actions they support to the platform using the format of Listing 2. Upon request of the dispatcher they can start or stop an action. Typically an Action Executor manages actions for a single actuator, but multiple executors may exist for one actuator. Since synchronization is done by the dispatcher this is not a problem. The executors can be added and removed at runtime so an executor going down does not crash the system.

There are no restrictions on the granularity of the actions provided by an Action Executor. However we tend towards coarser actions since this reduces the computational cost of the planner. Examples of actions are: move to a location, announce activities, assist with homework... Once an action is started control of the actuator is handed over to the action executor which is responsible for execution. When an action is finished, either because it completed or because it failed, its progress is communicated to the Dispatcher, which can then respond appropriately.

There is an Action Executor wrapped around DYAMAND to send commands to non-robotic actuators (light, door). We have also developed Action Executors for the Pepper and Nao robot. These Action Executors send commands to the robot's on-board NaoQi stack. The Action Executor for navigation actions internally uses ROS. These robotic actuators run on the

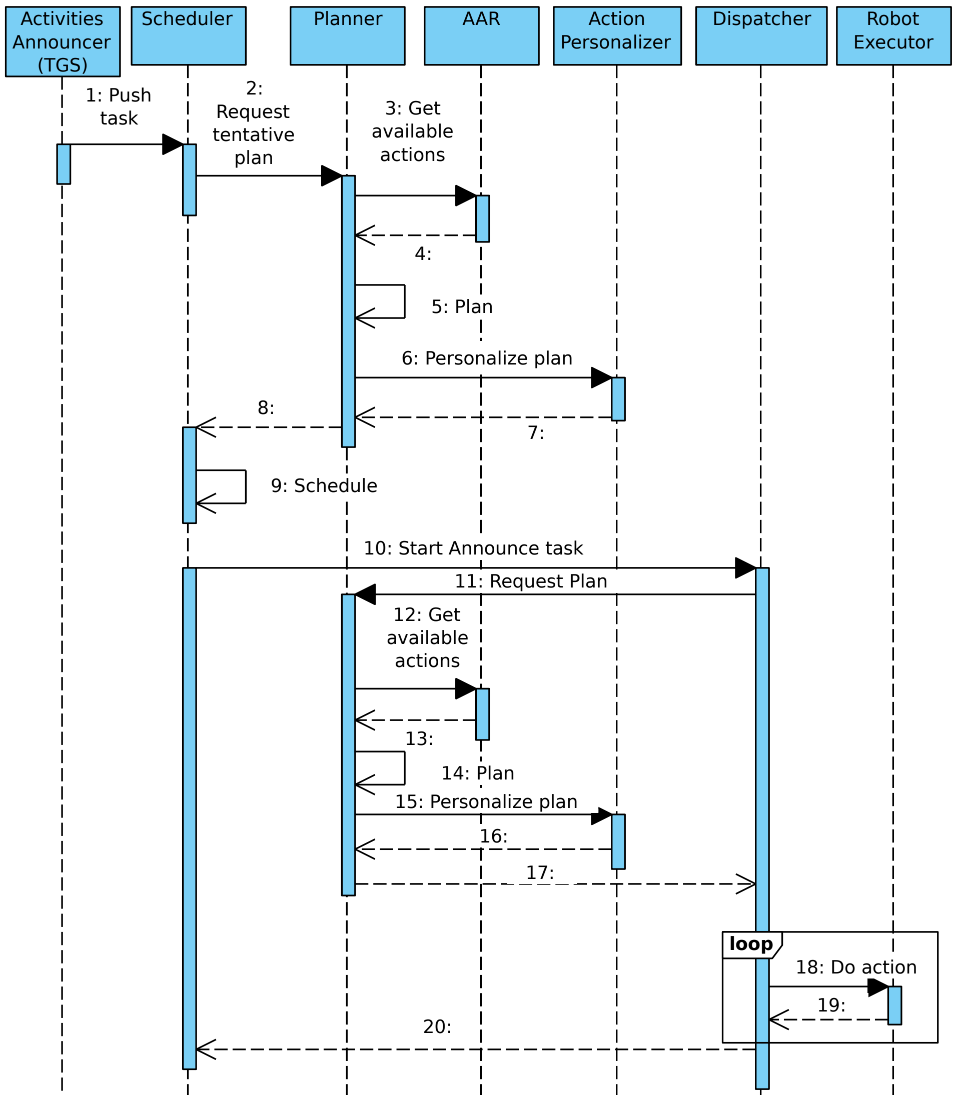


Figure 8: Sequence diagram depicting the control flow from the moment a Task is pushed until it has finished executing.

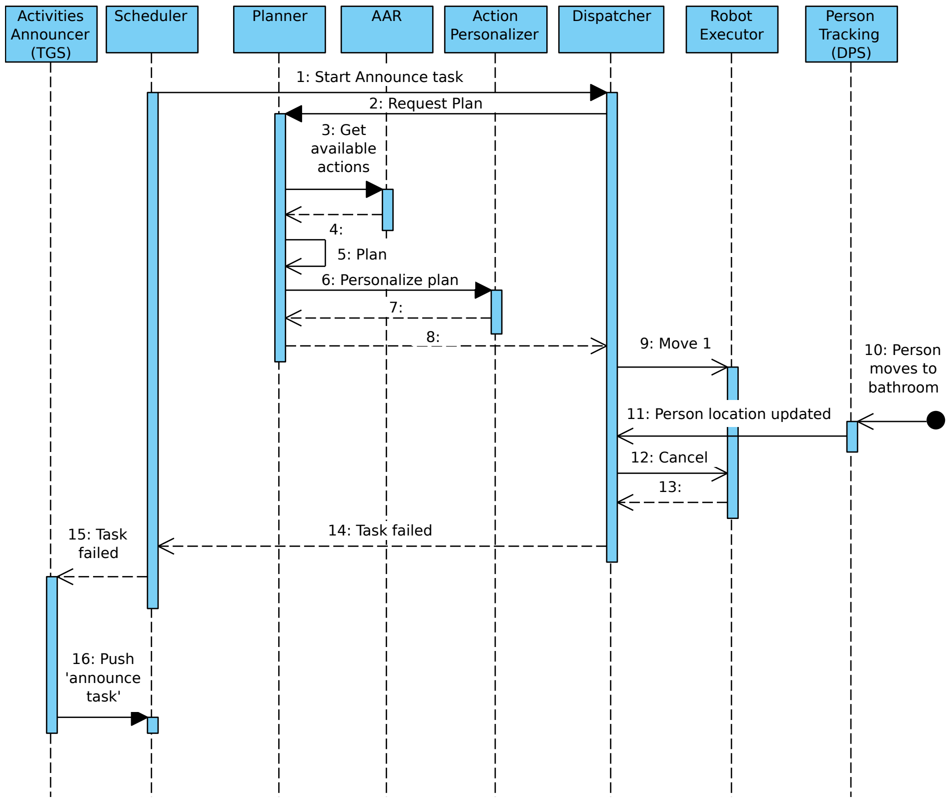


Figure 9: Sequence diagram depicting the control flow in case one of the Action fails. The Task fall back strategy has been set to early fail.

robot and their coarser definition leaves sufficient autonomy at the robot to ensure (physical) safety and privacy.

6. Evaluation

In this section, we detail a prototype we have built and evaluate the Quality Attribute Scenarios.

6.1. Prototype

We have deployed the platform in the imec HomeLab, a two-story house (600m^2) with technical corridors in every room to flexibly install sensors and an open home automation system[17]. A demo implementation is realized for visitors of the HomeLab, of the NH3 scenario where noise and light levels must be kept low in one room of the smart home. When the light in the patient's room is turned on, the monitoring TGS submits a task to be executed by a Pepper robot. The intervention task makes the robot enter the room to educate the patient and his possible visitors. Some screenshots of the demo can be seen in Figure 10. Full videos can be found online[18].

Figure 11 shows the different technologies and platform components that are involved in this proof-of-concept.

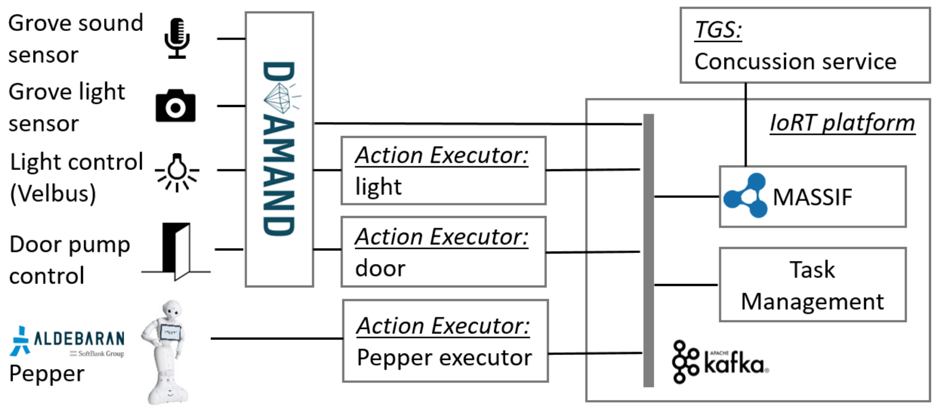
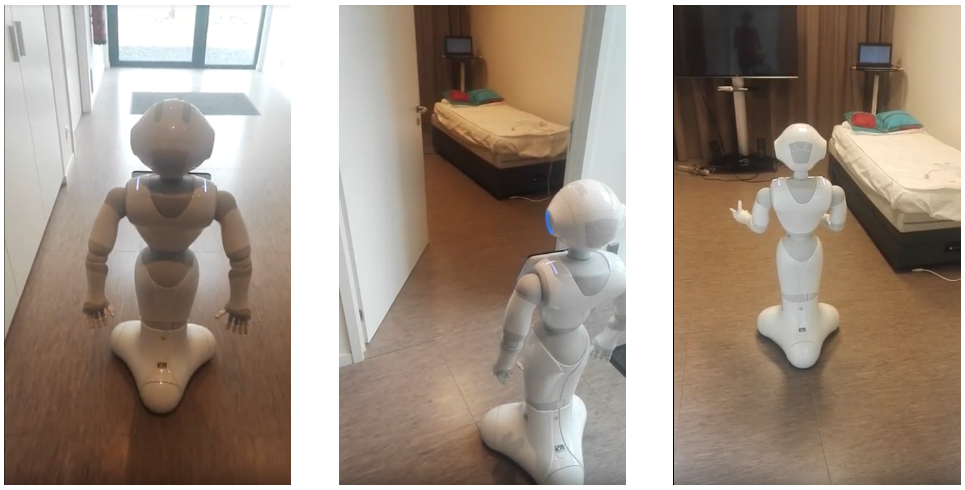


Figure 11: Different components and technologies used for the proof-of-concept.

[17]https://www.imec-int.com/nl/homelab
[18]https://goo.gl/Kn9hkx and https://goo.gl/zHsqJN



(a) Robot driving to the patient's room

(b) Robot entering the room

(c) Robot explaining light sensitivity to the patient

Figure 10: Robot responding to light trigger in concussed patient's room. (The requirement that the person should be in the room has been disabled for the purpose of this video.)

6.2. Evaluation of Quality Attribute Scenarios

In section 3.3 we put forward three quality attribute scenarios that we wanted to achieve as part of our non-functional requirements. The response measure of each scenarios was measured and the results are shown in Table 1. All evaluations were done using a laptop with an Intel(R) Core(TM) i5-5300U CPU running at 2.30GHz with 8GB RAM running Linux Ubuntu 16.04 LTS.

In order to measure the time for a TGS to become operational we measured the time between a TGS being booted, it being deployed on the MASSIF Platform and being accepted by the scheduler. The deployment of the TGS on the MASSIF Platform mostly depends on the time it takes to register the filter rules on the SCB. In the measurement we did not include the time needed for this registration as this heavily depends on the complexity of the filter rule, the amount of filter rules already present in the SCB and the size of the ontology used by the SCB. This was already thoroughly evaluated in previous research [31], which has shown that in most realistic scenarios this time is under 1 second.

To measure the time for a new Action Executor to be available we measured the time it took between the service being booted and all the new actions being available in the system. The used service was not connected to a robot or any other system to able to fairly asses the time till acceptance. 5 runs were deemed sufficient since the variance on the acceptance times was very low (0.025).

To measure the time required for an Action Filter to become operational we again isolate the service boot time from the loading of the semantic services in the reasoning engine for the same reasons as for the TGS. The average boot time was 1.91 seconds. 10 runs were used to establish the average instead of the 5 for the previous QAS since the variance was much higher (2.88).

QAS	Response measure	Target	Actual
Adding TGS	TGS operational	10 sec	1.18 sec
Adding Action Executor	New actions available	5 sec	1.38 sec
Adding Action Filter	Filter operational	10 sec	1.91 sec

Table 1: Evaluation of all Quality Attribute Scenarios put forth in section 3.3

7. Platform limitations

7.1. Other considerations

In this section, we discuss several other non-functional requirements that are important in an IoRT environment, but are currently not included in our platform architecture. We nevertheless propose several tactics that could be used to realize these requirements.

- **Privacy, security and trust** Privacy, security and trust are key requirements for any IoT platform [32, 33], yet the inclusion of robots in the IoT brings novel aspects to all three requirements. With respect to privacy, this novelty arises from the robot mobility and from the personal and contextual data stored in the platform. First, the robot mobility expands the potential spatial coverage of sensors to any location in a smart environment. In particular, the robot's camera and microphone may collect sensitive rich data. Second, adequate personalization and contextualization, the key goals of our work, can only be achieved if the services deployed on top of our platform have access to private information related to medical

history, activities, routines and preferences. Traditional privacy preservation mechanisms such as anonymization or delayed access to sensor data are not compatible with our goals. Instead, a user-friendly authoring tool for configuring data access control policies might be needed [34], that is able to translate these user-specified policy into a machine-readable format for the platform [35].
With respect to security, the main novelty arises from how robots increase the attack vector of IoT platforms: a malicious partner could hijack services or robots, in steering them to gather additional information (e.g. is a person at home) or, worse, deliberately cause physical harm. Therefore, appropriate mechanisms must be put in place to prevent unauthorized intrusion, e.g. by using authentication and role-based permissions.
These privacy preservation and security mechanisms should foster trust of users in our platform. Trust is nevertheless a multi-faceted attribute, so the trust of users in our platform is also impacted by other non-functional requirements discussed in this section, such as safety or timeliness.

- **Safety** In the envisioned interaction scenarios, robots act freely in the environment and not in caged environments. Physical actuation may cause harm to users, but there is also an inherent uncertainty on the human course of action. While the platform may perform a check on the type of interaction tasks assigned to the robot, the strongest safety guarantee can only be provided by autonomous control logic on the robot itself. For this reason, but also for latency reasons, our platform must not be able to directly control robots. Instead, the robot should have sufficient autonomy to determine the best course of action. Fast on-robot sensor-actuator loops will prevent potential harm to users, e.g. by immediately stopping the robot or by reducing the speed of actuation. These control loops are already available on mobile robots.
- **Timeliness** A delayed or instead too early interaction by the robot will negatively impact the user experience. The logic to determine time windows of appropriate context is the responsibility of the TGS and not of the platform. However, the platform should be designed in such a way that it introduces as least technical restrictions to this timeliness as possible. The platform, which requires an intensive and near real-time processing of sensor data streams, should be deployed close to the sources of the data, i.e. at the edge of the network on devices such as an IoT gateway. This deployment pattern is known in literature under various names, such as cloudlets [36], fog computing [37] or edge computing [38]. By deploying the platform close to the user, we also enable sending actions from the platform to be executed by the robot. Note that following this

model of cloudlet-based deployment might also help in protecting sensitive data [39], e.g. information on medical issues or on the presence in a home.

8. Related work

The integration of robots with smart environments has mostly been studied in the context of Ambient Assisted Living. The Ubiquitous Network Robot Platform [40] serves as a middleware between applications and robotic devices. It focuses on robotic applications that span multiple areas. The platform includes a user attribute registry that can be queried by the platform services.

Rashid et al. [41] introduced PEIS, a distributed middleware to realize an *ecological* view of the robot-environment relationship: robots, sensor motes and smart objects are seen as parts of the same system in which communication is realized via a shared tuple-space. The PEIS middleware was a cornerstone of the Rubicon and Robot-Era projects [42, 43], which pursued an ecology vision. In this vision, complex functionality is achieved by composing simple devices such as robots, sensor and effector nodes and purely computing nodes. In the Rubicon project, a three-layer system to control such an ecology is presented. The learning layer converts raw data into meaningful events, which are fed into a cognitive layer that sets goals for the ecology. The control layer is responsible for planning and execution the actions to reach these goals. In this way, the ecology was made adaptive to user preferences, e.g. it learned to have a robot clean the kitchen only after the user left it to start watching TV. These developments and insights were advanced in the Robot-Era project, including services in non-domestic environments. For instance, to coordinate the actions between multiple robots for shopping delivery and garbage collection. Our work shares some of the ideas, such as the combination of symbolic AI with data-driven machine learning algorithms and the need to have smart spaces cooperate with robots to execute tasks. One important differentiator is that we focus on *interaction*, where personalization and context are requisites for efficient and socially acceptable human-robot interaction. Also, our platform is conceived upfront for extensibility with services from 3rd party services. This extensibility is e.g. provided by using ontologically encoded knowledge that allows integration of domain-specific knowledge with more generic knowledge.

Other works have focused on how IoT information can be integrated into robotic planners and controllers. For instance, Cirillo et al. [44] use information from smart, sensor-rich environments to forecast the plan, or a set of plans, that a human is expected to perform. The robot adjusts its own plan to achieve its goals while respecting a set of interaction constraints, such as safety conditions, comfort conditions or task related conditions. For instance, these constraints may impose that the robot should never operate in the same room where the human is expected to be.

All the above works focus on the structural decomposition of functional tasks, e.g. to fetch an object or to clean up, or on limited interactions such as alerts, e.g. to take medicines. Our work instead provides support for semantic reasoning, which provides a flexible approach to include domain knowledge, contextual information and personal preferences when developing services. The use of semantics also ensures interoperability in many application domains.

Another strand of related work uses ontologies to allow robots to reason on their relationship with the environments and the objects in that environment. Seydoux et al. [45] proposes the IoT-O ontology containing a vocabulary to describe connected devices and their relation with the environment. The Semantic Sensor Network (SSN) ontology is extended with a novel Semantic Actuator Network to realize a Monitor-Actuator-Plan-Execution control loop on the robot. The use of ontologies as knowledge base underpinning a deliberative architecture for social interaction in human-robot collaboration scenarios was presented by [46]. Spatial relations and affordances for the whole scene are continuously computed and used to update the knowledge base.

The ontological pattern for contextual opportunities, presented in section 4.3, requires programmers to write explicit rules on contextual opportunities. Grosinger et al. [47] present an alternative approach that automatically deduces opportunities. The proposed system aims to maintain an equilibrium by pro-actively seeking for action schemes that can be executed to bring the system in a desirable state. While the advantage of the Equilibrium Maintenance Loop is that it can model and detect opportunities in a very sophisticated manner, it is however challenging to construct, maintain and extend to new use cases and contexts because it requires hand-coded specification of desired and undesired states and of state transitions. The advantage of our ontological approach is that it offers a simplified approach to model contextual opportunities and their dependencies on specific context parameters. This approach can easily be extended to new use cases and contexts. However, it does offer a less intricate manner to detect and predict these opportunities. Nevertheless, both approaches are complementary and provide an interesting avenue for future work.

9. Conclusion

To foster the adoption of social robots in our daily life, the timing, content and execution of their interactions should be constantly adjusted to our evolving preferences, needs and context. Using ontologic models, our platform allows developers to set-up data flows where robot interactions are triggered when a specific event is observed, or to define contextual opportunity windows in which an interaction should occur. The platform, using a message-driven microservice architecture, abstracts service developers from the complexity of scheduling, planning and execution.

A first area for future work is on the security and privacy aspects. Currently, there are no checks on which services subscribe to which topics. A related aspect is liability. As robots operate in the physical world, personal integrity needs to be taken into account when sending commands. Second, we will study more advanced techniques for multi-robot planning, resource reasoning and more advanced techniques for correctly estimating the timing duration of tasks.

Nevertheless, this platform provides a first step towards the integration of IoT and robotic technologies to create more enjoyable and more efficient human-robot interactions.

Acknowledgements

The research leading to these results was partially funded by the imec.ICON WONDER, imec.ICON ROBO-CURE and the ACTHINGS High Impact Initiative.

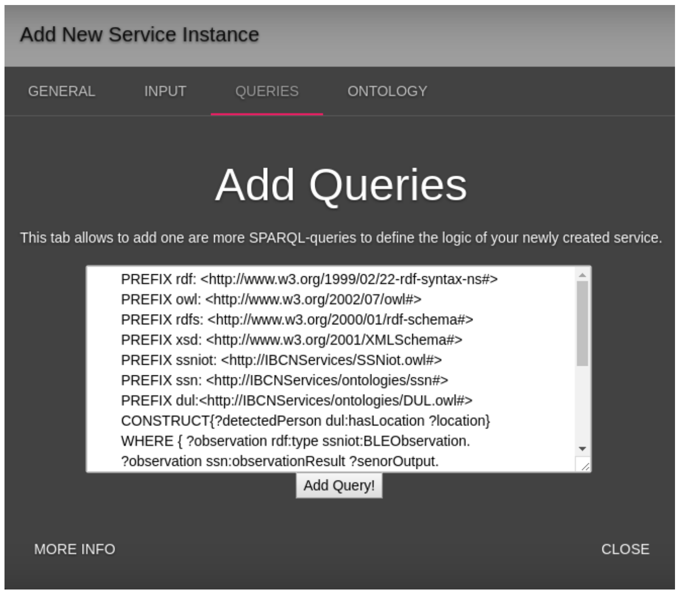
Appendix A. Semantic Service Definitions

Appendix A.1. GUI

Figure A.12 shows the GUI that was created to allow developers to easily input the required information to create a new TGS, i.e., its name in the GENERAL tab, the filter rules in the INPUT tab, the SPARQL queries in the QUERIES tab and the used ontologies in the ONTOLOGY tab.

Appendix A.2. SPARQL Query for the observation pattern in the smart nursing home NH2 scenario

Listing 3 shows the application logic of the semantic service that generates a solution to guide and sooth an elderly in response to the observation of the RFID tag of this elderly in the room of an other person.



This tab allows to add one are more SPARQL-queries to define the logic of your newly created service.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ssniot: <http://IBCNServices/SSNiot.owl#>
PREFIX ssn: <http://IBCNServices/ontologies/ssn#>
PREFIX dul:<http://IBCNServices/ontologies/DUL.owl#>
CONSTRUCT{?detectedPerson dul:hasLocation ?location}
WHERE { ?observation rdf:type ssniot:BLEObservation.
?observation ssn:observationResult ?senorOutput.
```

Figure A.12: User interface provided by the MASSIF Platform to create services

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ssniot: <http://IBCNServices/SSNiot.owl#>
PREFIX carecobots: <http://IBCNServices/ontologies/CareCoBots.owl#>
CONSTRUCT{
ssniot:Event rdf:type carecobots:GuideAndSoothWanderingPersonSolution.
carecobots:GuideAndSoothWanderingPersonSolution ssniot:isSolutionfor ?fault.
}
WHERE{
?observation ssniot:hasSymptom ?symptom.
?symptom ssniot:hasFault ?fault.
?fault rdf:type carecobots:WanderingIntoRestrictedAreaFault.
}
```

Listing 3: Example of a SPARQL query that makes up the application logic of a TGS. For conciseness the IRIs 'http://IBCNServices.github.io/Accio-Ontology/' have been shortened to 'http://IBCNServices/'.

Appendix A.3. SPARQL Query for the PreferredActionSelectionService

Listing 4 shows the application logic of the generic semantic service that outputs the most appropriate action based on a personalization request for a particular task. The choice is based on the instantiations of the personalization pattern as provided by the developers.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX profile: <http://IBCNServices/PersonProfileAccio.owl#>
PREFIX task: <http://IBCNServices/ontologies/TaskAccio.owl#>
PREFIX robotics: <http://IBCNServices/PersonalizedRobotics.owl#>
CONSTRUCT{
?actionPref task:isActionFor ?task.
}
WHERE{
?request rdf:type profile:PersonalizationRequest.
?request task:forTask ?task.
?task task:requiresAction ?actionGeneral.
?task task:executedOn ?person.
?person profile:hasPreference ?interactions.
?interactions rdf:type robotics:PreferredRobotInteractions.
?interactions task:requiresAction ?actionPref.
?actionPref rdf:type ?actionGeneral.
}
```

Listing 4: SPARQL query that implements the business logic of the PreferredActionSelectionService. For conciseness the IRIs 'http://IBCNServices.github.io/Accio-Ontology/' have been shortened to 'http://IBCNServices/'.

Appendix B. Task Management

Appendix B.1. JSON object for a time-constrained task

```
{
 "ID"        : "ActivitiesAnnouncer_S003_37",
 "Goal"      : "(activities_announced Bob_Smith_0001)",
 "StartTime" : "2017-08-12T06:26:23",
 "Deadline"  : "2017-08-12T07:20:00",
 "Priority"  : 4
}
```

Listing 5: JSON object for a time-constrained task.

Appendix B.2. Action description in PDDL

```
[
; Move
;
; Moves the robot from one waypoint to the other.
;
; @Param robot: The robot to move.
; @Param from-waypoint: The current location of the robot.
; @Param to-waypoint: The destination location of the robot.
(:action move
  :parameters
    (?robot         - Robot
     ?from-waypoint - Location
     ?to-waypoint   - Location)

  :precondition
    (and
      (robot-at-location ?robot ?from-waypoint)
      (can-move ?from-waypoint ?to-waypoint))
  :effect
    (and
      (robot-at-location ?robot ?to-waypoint)
      (not (robot-at-location ?robot ?from-waypoint))

      ; now in the same room as people in the room you just entered
      (forall (?person - person)
        (when (person-at-location ?person ?to-waypoint)
          (robot-in-same-room-as ?robot ?person)
        )
      )

      ; now no longer in same room as people in room you just left
      (forall (?person - person)
        (when (person-at-location ?person ?from-waypoint)
          (and
            (not (robot-in-same-room-as ?robot ?person))
            (not (robot-near-person ?robot ?person))
          )
         )
      )
    )
)]
```

Listing 6: Example of an Action description in PDDL syntax

Appendix C. Technologies

Element	Technology	Version
Message broker	Apache Kafka	2.11-1.1.0
Reasoner	MASSIF	n.a.
Planner	Fast Downward	n.a.
Sensor interface	DYAMAND	0.12.0
Actuator interface	DYAMAND	0.12.0
Robot	Aldebaran Pepper	1.8a
Robot software	NaoQi	2.5.5
Light sensor	Grove Light sensor	1.0
Sound sensor	Grove Sound sensor	n.a

Table C.2: Technology overview for used for the prototype implementation described in section 6.

Element	URL
Apache Kafka	`https://www.apache.org/dist/kafka/1.1.0/RELEASE_NOTES.html`
MASSIF	`https://github.com/IBCNServices/MASSIF`
IORT ontology	`https://github.com/IBCNServices/Accio-Ontology/tree/gh-pages`
Fast Downward	`http://www.fast-downward.org/`
Aldebaran Pepper	`http://doc.aldebaran.com/2-5/home_pepper.html`
Grove sensors	`http://wiki.seeedstudio.com/Grove/`

Table C.3: URLs for the various components used to build the prototype.

References

[1] ISO Standard, 8373:2012, Robots and robotic devices – Vocabulary (2012).

[2] Q. Xu, J. S. L. Ng, O. Y. Tan, Z. Huang, Needs and attitudes of singaporeans towards home service robots: a multi-generational perspective, Universal Access in the Information Society 14 (2015) 477–486.

[3] A. Pereira, I. Leite, S. Mascarenhas, C. Martinho, A. Paiva, Using empathy to improve human-robot relationships, in: M. H. Lamers, F. J. Verbeek (Eds.), Human-Robot Personal Relationships, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 130–138.

[4] I. Leite, C. Martinho, A. Paiva, Social robots for long-term interaction: A survey, International Journal of Social Robotics 5 (2013) 291–308.

[5] M. Heerink, B. Kröse, V. Evers, B. Wielinga, Assessing acceptance of assistive social agent technology by older adults: the almere model, International Journal of Social Robotics 2 (2010) 361–375.

[6] O. Vermesan, A. Bröring, E. Tragos, M. Serrano, D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, M. Dragone, A. Saffiotti, et al., Internet of robotic things: converging sensing/actuating, hypoconnectivity, artificial intelligence and iot platforms, in: O. Vermesan, J. Bacquet (Eds.), Cognitive hyperconnected digital transformation: internet of things intelligence evolution, River Publishers, Gistrup, Denmark, 2017, pp. 97–155.

[7] G. Milliez, R. Lallement, M. Fiore, R. Alami, Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring, in: The Eleventh ACM/IEEE International Conference on Human Robot Interaction, HRI '16, IEEE Press, Piscataway, NJ, USA, 2016, pp. 43–50.

[8] I. Leite, R. Henriques, C. Martinho, A. Paiva, Sensors in the wild: Exploring electrodermal activity in child-robot interaction, in: Proceedings of the 8th ACM/IEEE International Conference on Human-robot Interaction, HRI '13, IEEE Press, Piscataway, NJ, USA, 2013, pp. 41–48.

[9] Robo-Cure. Social robots, connected devices and artificial intelligence to improve healthcare, 2017. Online. https://www.imec-int.com/

`cache/pdfs/en/what-we-offer/research-portfolio/robo-cure.pdf` Last-accessed on 2018-03-29.

[10] M. A. Al-Taee, W. Al-Nuaimy, Z. J. Muhsin, A. Al-Ataby, Robot assistant in management of diabetes in children based on the internet of things, IEEE Internet of Things Journal 4 (2017) 437–445.

[11] T. T. Tran, T. Vaquero, G. Nejat, J. C. Beck, Robots in retirement homes: Applying off-the-shelf planning and scheduling to a team of assistive robots, Journal of Artificial Intelligence Research 58 (2017) 523–590.

[12] B. Bruno, N. Y. Chong, H. Kamide, S. Kanoria, J. Lee, Y. Lim, A. K. Pandey, C. Papadopoulos, I. Papadopoulos, F. Pecora, A. Saffiotti, A. Sgorbissa, Paving the way for culturally competent robots: A position paper, in: 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pp. 553–560.

[13] F. Ongenae, F. De Backere, J. Nelis, S. De Pestel, C. Mahieu, S. Elprama, C. Jewell, A. Jacobs, P. Simoens, F. De Turck, Personalized robot interactions to intercept behavioral disturbances of people with dementia, in: 15th International Semantic Web Conference, ISWC, 2016, pp. 1–4.

[14] B. Chikhaoui, B. Ye, A. Mihailidis, Ensemble learning-based algorithms for aggressive and agitated behavior recognition, in: C. R. García, P. Caballero-Gil, M. Burmester, A. Quesada-Arencibia (Eds.), Ubiquitous Computing and Ambient Intelligence, Springer International Publishing, Cham, 2016, pp. 9–20.

[15] Y. Akimoto, E. Sato-Shimokawara, Y. Fujimoto, T. Yamaguchi, Approach function study for concierge-type robot by model-based development with user model for human-centred design, ROBOMECH Journal 3 (2016) 26.

[16] C. R. Huang, P. C. Chung, K. W. Lin, S. C. Tseng, Wheelchair detection using cascaded decision tree, IEEE Transactions on Information Technology in Biomedicine 14 (2010) 292–300.

[17] M. Shields, Control- versus data-driven workflows, in: I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields (Eds.), Workflows for e-Science: Scientific Workflows for Grids, Springer, London, 2007, pp. 167–173.

[18] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, S. Clarke, Middleware for internet of things: A survey, IEEE Internet of Things Journal 3 (2016) 70–95.

[19] R. Janssen, R. van de Molengraft, H. Bruyninckx, M. Steinbuch, Cloud based centralized task control for human domain multi-robot operations, Intelligent Service Robotics 9 (2016) 63–77.

[20] F. Pecora, M. Cirillo, F. Dell'Osa, J. Ullberg, A. Saffiotti, A constraint-based approach for proactive, context-aware human support, Journal of Ambient Intelligence and Smart Environments 4 (2012) 347–367.

[21] P. Simoens, M. Dragone, A. Saffiotti, The internet of robotic things: A review of concept, added value and applications, International Journal of Advanced Robotic Systems 15 (2018) 1–11.

[22] P. Barnaghi, W. Wang, C. Henson, K. Taylor, Semantics for the internet of things: Early progress and back to the future, Int. J. Semant. Web Inf. Syst. 8 (2012) 1–21.

[23] J. Nelis, T. Verschueren, D. Verslype, C. Develder, Dyamand: Dynamic, adaptive management of networks and devices, in: Proceedings of the 2012 IEEE 37th Conference on Local Computer Networks (LCN 2012), LCN '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 192–195.

[24] L. Bass, P. Clements, R. Kazman, Software architecture in practice, 3rd edition, Pearson Education, 2012.

[25] B. Bruno, N. Y. Chong, H. Kamide, S. Kanoria, J. Lee, Y. Lim, A. K. Pandey, C. Papadopoulos, I. Papadopoulos, F. Pecora, et al., The caresses eu-japan project: making assistive robots culturally competent, in: Ambient Assisted Living, Italian Forum, ForItAAL, Genova, Italy, 2017.

[26] Z. Zainol, K. Nakata, Generic context ontology modelling: A review and framework, in: 2010 2nd International Conference on Computer Technology and Development, pp. 126–130.

[27] F. Ongenae, P. Duysburgh, N. Sulmon, M. Verstraete, L. Bleumers, S. De Zutter, S. Verstichel, A. Ackaert, A. Jacobs, F. De Turck, An ontology co-design method for the co-creation of a continuous care ontology, Applied Ontology 9 (2014) 27–64.

[28] P. Bonte, F. Ongenae, F. De Backere, J. Schaballie, D. Arndt, S. Verstichel, E. Mannens, R. Van de Walle, F. De Turck, The massif platform: a modular and semantic platform for the development of flexible iot services, Knowledge and Information Systems 51 (2017) 89–126.

[29] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL-the planning domain definition language, 1998.

[30] M. Helmert, The fast downward planning system, J. Artif. Int. Res. 26 (2006) 191–246.

[31] F. Ongenae, J. Famaey, S. Verstichel, S. D. Zutter, S. Latré, A. Ackaert, P. Verhoeve, F. D. Turck, Ambient-aware continuous care through semantic context dissemination, BMC Medical Informatics and Decision Making 14 (2014) 97.

[32] S. Sicari, A. Rizzardi, L. Grieco, A. Coen-Porisini, Security, privacy and trust in internet of things: The road ahead, Computer Networks 76 (2015) 146 – 164.

[33] Z. Yan, P. Zhang, A. V. Vasilakos, A survey on trust management for internet of things, Journal of Network and Computer Applications 42 (2014) 120 – 134.

[34] R. W. Reeder, C.-M. Karat, J. Karat, C. Brodie, Usability challenges in security and privacy policy-authoring interfaces, in: C. Baranauskas, P. Palanque, J. Abascal, S. D. J. Barbosa (Eds.), Human-Computer Interaction – INTERACT 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 141–155.

[35] D. Garcia, M. B. F. Toledo, M. A. M. Capretz, D. S. Allison, G. S. Blair, P. Grace, C. Flores, Towards a base ontology for privacy protection in service-oriented architecture, in: 2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1–8.

[36] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, IEEE Pervasive Computing 8 (2009) 14–23.

[37] F. Bonomi, R. Milito, P. Natarajan, J. Zhu, Fog computing: A platform for internet of things and analytics, in: N. Bessis, C. Dobre (Eds.), Big Data and Internet of Things: A Roadmap for Smart Environments, Springer International Publishing, Cham, 2014, pp. 169–186.

[38] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (2016) 78–81.

[39] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, B. Amos, Privacy mediators: Helping iot cross the chasm, in: Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, HotMobile '16, ACM, New York, NY, USA, 2016, pp. 39–44.

[40] S. Nishio, K. Kamei, N. Hagita, Ubiquitous network robot platform for realizing integrated robotic applications, in: S. Lee, H. Cho, K.-J. Yoon, J. Lee (Eds.), Intelligent Autonomous Systems 12: Volume 1 Proceedings of the 12th International Conference IAS-12, held June 26-29, 2012, Jeju Island, Korea, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 477–484.

[41] J. Rashid, M. Broxvall, A. Saffiotti, A middleware to integrate robots, simple devices and everyday objects into an ambient ecology, Pervasive and Mobile Computing 8 (2012) 522 – 541. Special Issue on Ambient Ecologies.

[42] F. Cavallo, R. Limosani, A. Manzi, M. Bonaccorsi, R. Esposito, M. Di Rocco, F. Pecora, G. Teti, A. Saffiotti, P. Dario, Development of a socially believable multi-robot solution from town to home, Cognitive Computation 6 (2014) 954–967.

[43] G. Amato, D. Bacciu, M. Broxvall, S. Chessa, S. Coleman, M. Di Rocco, M. Dragone, C. Gallicchio, C. Gennaro, H. Lozano, T. M. McGinnity, A. Micheli, A. K. Ray, A. Renteria, A. Saffiotti, D. Swords, C. Vairo, P. Vance, Robotic ubiquitous cognitive ecology for smart homes, Journal of Intelligent & Robotic Systems 80 (2015) 57–81.

[44] M. Cirillo, L. Karlsson, A. Saffiotti, Human-aware planning for robots embedded in ambient ecologies, Pervasive and Mobile Computing 8 (2012) 542 – 561. Special Issue on Ambient Ecologies.

[45] N. Seydoux, K. Drira, N. Hernandez, T. Monteil, Iot-o, a core-domain iot ontology to represent connected devices networks, in: E. Blomqvist, P. Ciancarini, F. Poggi, F. Vitali (Eds.), Knowledge Engineering and Knowledge Management, Springer International Publishing, Cham, 2016, pp. 561–576.

[46] S. Lemaignan, M. Warnier, E. A. Sisbot, A. Clodic, R. Alami, Artificial cognition for social humanrobot interaction: An implementation, Artificial Intelligence 247 (2017) 45 – 69. Special Issue on AI and Robotics.

[47] J. Grosinger, F. Pecora, A. Saffiotti, Making robots proactive through equilibrium maintenance, in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16, AAAI Press, 2016, pp. 3375–3381.