Noname manuscript No.
(will be inserted by the editor)

A Context-Aware Monitoring Architecture for Supporting System Adaptation and Reconfiguration

Oscar Cabrera · Marc Oriol · Xavier Franch · Jordi Marco

Received: date / Accepted: date

Abstract *Context:* Modern services and applications need to react to changes in their context (e.g. location, memory consumption, number of users) to improve the user's experience. To obtain this context, a monitoring infrastructure with adequate functionality and quality levels is required. But this monitoring infrastructure needs to react to the context as well, raising the need for context-aware monitoring tools. *Objective:* Provide a generic solution for context-aware monitoring able to effectively react to contextual changes. *Method:* We have designed CAMA, a service-oriented Context-Aware Monitoring Architecture that can be easily configured, adapted and evolved according to contextual changes. CAMA implements a decoupled architecture and manages a context domain ontology for modelling the inputs, outputs and capabilities of monitoring tools. *Results:* CAMA has been demonstrated in three real use cases. We have also conducted different evaluations, including an empirical study. The results of the evaluations show that (1) the overhead introduced by the architecture does not degrade the behavior of the system, except in extreme conditions; (2) the use of ontologies is not an impediment for practitioners, even when they have little knowledge about this concept; and (3) the reasoning capabilities of CAMA enable context-aware adaptations. *Conclusions:* CAMA is a solution useful for both researchers and practitioners. Researchers can use this architecture as a baseline for providing different extensions or implementing new approaches on top of CAMA that require context-aware monitoring. Practitioners may also use CAMA in their projects in order to manage contextual changes in an effective way.

Oscar Cabrera
CoDeS, Chilpancingo, Mexico
E-mail: cabrera.bejar82@gmail.com

Marc Oriol · Xavier Franch · Jordi Marco
Universitat Politècnica de Catalunya, Barcelona, Spain
E-mail: moriol@essi.upc.edu, franch@essi.upc.edu, jmarco@cs.upc.edu

Keywords Context-aware computing · Context acquisition · Context life cycle · Software adaptation · Software reconfiguration · Context ontology

1 Introduction

Context-awareness is a key feature in modern approaches and computer paradigms, such as Pervasive Computing, Smart Cities and the Internet of Things. This feature implies that services and applications must be aware of their changing contexts to automatically adapt their functionality in order to improve the quality of experience (QoE) of its users [16]. Context-awareness requires several components and processes working together with the aim to accomplish the context life cycle, which embraces context acquisition, modelling, reasoning and dissemination [19].

To illustrate this with an example, a video streaming service platform might be interested to acquire the QoE reported by its end-users in Social Media as contextual information for its services. For optimal performance, the configuration of the monitoring tools needs to be aware and adapt to the context as well. For instance, depending on the number of messages that users share in Social Media, the monitoring tools could adapt the keywords and frequency of monitoring to adjust the amount of messages collected. If an excessive amount of messages is obtained, it would be adequate to use more restrictive keywords in order to improve precision in expense of recall, whether if not many messages are obtained, it would be preferable to use less restrictive keywords to improve recall in expense of precision.

In the context life cycle, the first of the phases, *context acquisition*, is responsible for (1) gathering the context raw data from different entities (agents and resources) and (2) disseminating the gathered data to the interested parties. Context acquisition is implemented through *monitoring tools* that integrate software services and sensors as data gathering instruments [33, 40]. However, monitoring tools are software applications themselves; therefore, they also need to be aware of their own context to deliver the best possible functionality with adequate quality [45].

In this paper, we are interested in *context-aware monitoring*, which basically raises two main challenges to be solved related to the two aforementioned responsibilities of context acquisition, namely data gathering and dissemination. First, context-aware monitoring tools should support dynamic capabilities, i.e., reconfiguration and adaptation to continuously obtain and provide reliable context information. This characteristic is necessary to bind the behavior of the monitoring tool to the behavior of the context-aware system under supervision [4]. If the system incorporates new features or changes the behavior of existing ones, the monitoring tool needs to be able to gather different data. In addition, a context-aware monitor needs to react automatically when a source of data becomes temporarily unavailable. This challenge generates a first research question (RQ):

- RQ1: In what way can we provide a context-aware monitoring tool with the required reconfiguration and adaptation capabilities to face the constantly changing situation of the services and applications that compose a context-aware system?

This RQ can be decomposed into the following sub-RQs, considering the typology of events that may trigger a reconfiguration as explained above.

- RQ1.1: In what way can a context-aware monitoring tool manage the supervision of new features or entities?
- RQ1.2: In what way can a context-aware monitoring tool manage dynamic capabilities (i.e., reconfiguration, adaptation and evolution) when new requirements or changes in the context data, functions or quality features of an entity are detected?
- RQ1.3: In what way can a context-aware monitoring tool manage the failure of a data gathering instrument?

Note that the sub-RQs are related to the reconfiguration capabilities of a context-aware monitoring tool. However, RQ1.1 is more related to research the capability of adding new monitors for monitoring new features or entities; RQ1.2 is related to research the capability of reconfiguring monitors that are already deployed; and RQ1.3 investigates the capability of detecting failures in a monitoring tool and replacing it.

Second, the gathered data needs to be classified according to a well-defined data schema in order to avoid risks in data management, mainly in its dissemination and storage, due to ambiguity, inconsistencies and poor integrity and characterization. This challenge yields a second research question:

- RQ2: In what way can we structure, store and manage the data gathered by a context-aware monitoring tool in order to be easily disseminated to other components?

To the best of our knowledge, and as discussed later in the state of the art section, none of the existing context-aware monitoring frameworks fully satisfy all the presented RQs. To cover such a gap, we propose CAMA, a $\underline{\text{C}}$ontext-$\underline{\text{A}}$ware $\underline{\text{M}}$onitoring $\underline{\text{A}}$rchitecture. It is based on state of the art software patterns of service-oriented architecture [34], enabling to plug and play different monitors that can be easily reconfigured, evolved and adapted for supporting highly dynamic environments. CAMA includes a component to handle a *context ontology* for conceptualizing the inputs, outputs and capabilities of monitoring tools that can be useful to disseminate a unified and representative schema of monitored data. The ontological approach proposed in this paper includes the context ontology presented in [11] in the core of CAMA, and proposes its extension in every instance of the architecture in a particular scenario, yielding to scenario-specific context domain ontologies.

It is important to highlight the importance of using an ontology in the core of CAMA against other alternatives. In a previous work [10], we discussed the context modelling research, perspectives and formalism, and we found that one of the most suitable context modelling techniques are ontologies, in spite of some of its weaknesses and the existence of other popular techniques, such as key-value, mark-up scheme, graphical, object oriented and logic-based approaches [35]. Nowadays, we find that several authors working on context-aware computing choose ontologies for modelling context information [2][3][37]. The main reasons are: ontologies enable the sharing of knowledge by open dynamic agents (e.g., web services), they supply semantics for intelligent agents to reason about context information, they promote the interoperability among devices and computational agents, they enable semantic interoperability and provide common understanding of the structure of context information classes among users, and they provide inference mechanisms for predicting user needs and expectations, among others. CAMA could also be realized using one of the other aforementioned alternative techniques, but due to the characteristics discussed above, we considered the ontologies as the most suitable modelling technique in CAMA, as it can also provide the semantic data structure that can be used for different data sources (e.g., databases).

The feasibility of the proposed context-aware monitoring architecture has been demonstrated through three specific use cases belonging to different domains: smart city apps and services, a platform for saving home energy consumption, and a video streaming service. This diversity shows that our proposed solution can be applied to different domains and software platforms. Those use cases have also helped us to identify the requirements that such context-aware monitoring framework should have. Additionally, to demonstrate the feasibility of the proposed approach, we have performed different evaluations, including an empirical study that focuses on the usability of the proposed ontology that is managed by participants with different degrees of modeling ontologies and monitoring expertise. The results of the evaluations show that:

- Extending our proposed ontology with new monitoring tools to supervise new features or entities can be achieved in a timely and correct manner even with a low expertise in ontologies and monitoring (RQ1.1).
- Multiple monitoring tools can be reconfigured in parallel with an adequate performance (high frequency of reconfigurations and a low response time) (RQ1.2).
- Failing measure instruments can be replaced in parallel with an adequate performance (high frequency of replacements and a low response time) (RQ1.3).
- The data gathered by the architecture can be easily structured, stored and managed to be disseminated to other components (RQ2).

The rest of this paper is structured as follows. Section 2 surveys the related work. Section 3 presents the research method that has been used to define our proposed solution. Section 4 describes the CAMA monitoring architecture. Section 5 provides the details on the CAMA ontological basis. Section 6 provides implementation details of the proposed architecture and ontology. Section 7 evaluates different features of the monitoring architecture and the ontology proposed in this work. Finally, Section 8 presents the conclusions.

2 State of the art

In the last recent years, several research approaches have emerged in the field of context-aware monitoring. Most of them are aligned with trending computer paradigms (e.g., Pervasive Computing, Smart Cities and the Internet of Things).

We have conducted a literature review and evaluated the related work by analyzing how they address the RQs defined in Section 1. The selection of papers has been conducted following some systematic principles. We have searched for papers containing the keywords "context" and "monitoring" using the Google Scholar database. Due to the enormous amount of results, we picked the 25 most relevant papers according to the database ranking. From these 25 results, 14 were discarded for being out scope, leading to 11 papers to analyse. Then we conducted a backward snowballing process to obtain additional relevant papers that the string-based search could have missed. We included the works that were cited by more than one paper and were in the scope of context monitoring. In this step, 4 new papers were added, yielding to 15 papers. Then we grouped those papers that referred to the same proposal, leading to 13 proposal. Finally, we applied forward snowballing techniques to obtain the last version of those proposals.

The list of papers and its analysis is shown in Table 1 and described below.

Table 1: Analysis of related work

	RQ1									**RQ2**	
	RQ1.1			**RQ1.2**			**RQ1.3**				
Proposal	Supported (Yes/No)	Scope (Generic/Specific)	Mode (Auto./Manual)	Supported (Yes/No)	Scope (Generic/Specific)	Mode (Auto./Manual)	Supported (Yes/No)	Scope (Generic/Specific)	Mode (Auto./Manual)	Approach (Ont./Not Ont.)	Supported Queries (Rich / Poor)
MobiCon [26, 27]	Y	G	A	Y	S	A	Y	S	A	N	R
CARE [6]	Y	G	?	Y	S	A	N	N/A	N/A	?	R
Orchestrator [23]	Y	G	A	Y	G	A	Y	S	A	N	R
SeeMon [21, 22]	Y	G	A	Y	S	A	N	N/A	N/A	N	R
CoMon [25, 28]	Y	S	A	Y	G	A	Y	G	A	N	R
HiCon [12]	Y	G	A	Y	S	A	N	N/A	N/A	N	R
CLAD [31, 32]	N	N/A	N/A	Y	S	A	Y	G	A	N	P
Contory [38]	Y	S	A	Y	G	A	Y	G	A	N	R
DYNAMICO [42, 43, 44]	Y	S	A	Y	G	A	N	N/A	N/A	N	R
CALM [20]	N	N/A	N/A	N	N/A	N/A	N	N/A	N/A	N	P
H-SAUDE [13]	Y	G	M	Y	G	A	N	N/A	N/A	N	R
SINDI [29, 30]	Y	G	A	Y	S	A	N	N/A	N/A	N	R
Esposito et al. [14]	Y	G	?	N	N/A	N/A	N	N/A	N/A	O	R
CAMA (our proposal)	Y	G	A	Y	G	A	Y	G	A	O	R

RQ1.1 - In what way can a context-aware monitoring tool manage the supervision of new features or entities? Most of the approaches have an extensible

architecture capable of adding new monitoring tools to gather new metrics with a generic scope and in an automatic manner (i.e., without manually recompiling or linking the code) [12, 21, 22, 23, 26, 27, 29, 30]. Other approaches can add new monitoring tools but their scope is limited to just the healthcare environment [13], or to particular technologies, like cellphone devices [25, 28, 38] or SOA governance platforms [42, 43, 44]. CARE [6] and Esposito et al. [14] can handle multiple monitoring tools, but it is not clear if adding one may require some manual intervention. Finally, CLAD [31, 32] and CALM [20] do not provide the capability to add new monitoring tools.

RQ1.2 - In what way can a context-aware monitoring tool manage dynamic capabilities (i.e., reconfiguration, adaptation and evolution) when new requirements or changes in the context data, functions or quality features of an entity are detected? Some approaches include reconfiguration capabilities with a generic scope and in an automatic manner [13, 23, 25, 28, 38, 42, 43, 44]. Other approaches can reconfigure the monitoring tools dynamically but are limited to a specific scope (e.g., they can just activate/deactivate monitoring tools) [6, 12, 21, 22, 26, 27, 29, 30, 31, 32]. Finally, CALM [20] and Esposito et al. [14] do not support dynamic reconfigurations.

RQ1.3 - In what way can a context-aware monitoring tool manage the failure of a data gathering instrument? Most of the contributions are not able to detect and replace a failing monitoring tool [6, 12, 13, 14, 20, 21, 22, 29, 30, 42, 43, 44]. Only CoMon [25, 28], CLAD [31, 32] and Contory [38] have mechanisms to detect and replace failing monitoring tools with a generic scope and low effort. MobiCon [26, 27] and Orchestrator [23] can detect when a monitoring tool is unavailable but other types of failures are not investigated.

RQ2 - In what way can we structure, store and manage the data gathered by a context-aware monitoring tool in order to be easily disseminated to other components? Most approaches provide an API with a rich query mechanism to gather data, but without using an ontology [12, 13, 21, 22, 23, 25, 26, 27, 28, 29, 30, 38, 42, 43, 44]. The lack of an ontology hampers the ability to integrate the data with other sources or tools in a richly manner, as ontologies are well-known to be a powerful instrument that facilitates knowledge and data sharing between different frameworks. Esposito et al. [14] present the only approach that defines an ontology to provide semantics to the monitored data. CARE [6] provides an architecture that would eventually interact with ontological reasoners, but this aspect was not addressed in the paper. Finally, some approaches provide just a simple API to gather the monitored data without rich semantics [20, 31, 32].

Although all the approaches satisfy some of the RQs to a certain extent, none of them fully satisfies all the presented RQs. As depicted in Table 1, none of the existing approaches fully satisfies more than 2 out of 4 RQs.

Finally, it is worth to mention that context-aware monitoring is related to Application Performance Management (APM). APM tools monitor and manage the performance of complex software applications by supporting the detection, diagnosis and resolution of performance problems [1, 7, 18, 39]. However, APM tools are driven by performance issues, whereas context-aware monitoring is driven by contextual changes (which might include, but are not limited to, performance). Furthermore, to the best of our knowledge, current APM tools do not have context-aware self-adaptive capabilities.

To improve this state of the art, we have developed CAMA, a monitoring system capable of adding new monitoring tools (RQ1.1), with reconfiguration capabilities (RQ1.2), able to replace failing monitoring tools (RQ1.3) and able to provide access to the monitored data by means of a rich semantics mechanism with the elements structured and defined through an ontology (RQ2).

3 Research Method

To address our RQs, we have applied the Design Science Research (DSR) method [15]. DSR is a research methodology that offers guidelines to conduct research and is usually applied in the fields of Engineering and Computer Science.

DSR defines 4 types of knowledge contributions depending on the maturity of the problem (or application domain) and the maturity of the solutions in the existing literature (see Fig. 1).

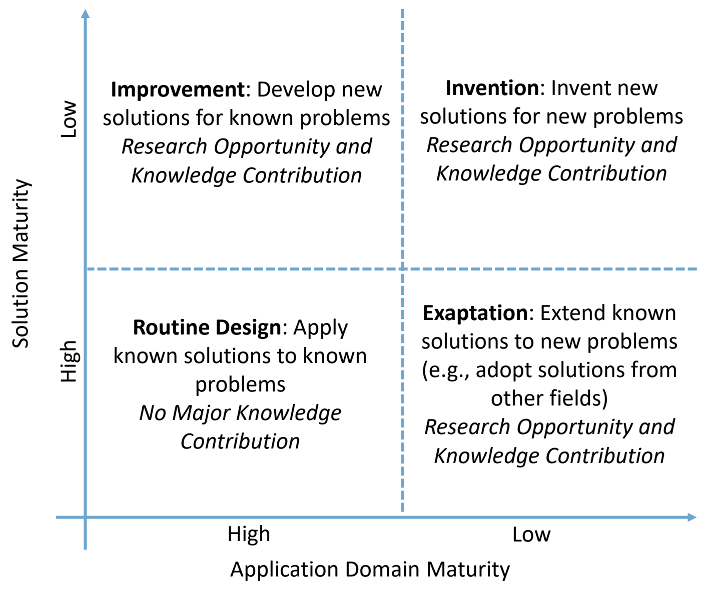
Solution Maturity
Low
High
Improvement: Develop new solutions for known problems Research Opportunity and Knowledge Contribution
Invention: Invent new solutions for new problems Research Opportunity and Knowledge Contribution
Routine Design: Apply known solutions to known problems No Major Knowledge Contribution
Exaptation: Extend known solutions to new problems (e.g., adopt solutions from other fields) Research Opportunity and Knowledge Contribution
High
Low
Application Domain Maturity

Fig. 1: DSR Knowledge Contribution Framework [15]

The contributions of this paper fit into the *Improvement* category. In this category, the domain application is a well-known and established field of research (in our case, the challenges in context-aware adaptive monitoring), whereas the existing solutions do not fully satisfy the problems being addressed (as described in the previous section).

DSR contributions in the *Improvement* category provide their results in the form of artifacts at one or more levels of abstraction (see Table 2).

Table 2: Contribution levels in DSR [15]

	Contribution level	**Example artifacts**
More abstract, complete and mature knowledge.	Level 3. Well–developed design theory about embedded phenomena.	Design theories (mid–range and grand theories).
$\Updownarrow$	Level 2. Nascent design theory–knowledge as operational principles/architecture.	Constructs, methods, models, design principles, technological rules.
More specific, limited, and less mature knowledge.	Level 1. Situated implementation of artifact.	Instantiations (software products or implemented processes).

In this work, we tackle the RQs by producing the needed artifacts at Level 1 and Level 2 (see Table 3).

Table 3: Contribution levels in our RQs

RQ	**Level 2**	**Level 1**
RQ1	CAMA: A software architecture for Context-aware monitoring tools.	An instantiation of CAMA by adding a set of monitoring tools for Social Network monitoring.
RQ2	A domain context ontology for CAMA.	An instantiation of CAMA by extending the ontology with the concepts of the Social Network monitoring domain.

These artifacts are part of the results of the EU H2020 project SUPERSEDE[1]. Their development has been conducted in an iterative manner. We first conducted several interviews and on-site workshops with three use cases of the project, namely Siemens, SEnerCon and ATOS to understand their needs related to context monitoring. Each use case belongs to a different domain, namely smart city apps and services, a platform for saving home energy consumption, and a video streaming service, respectively. The initial meetings with the use case providers were used to specify the purpose, requirements and scope of the artifacts to be developed. Subsequent regular meetings (both online and on-site) were conducted to assess that the development of CAMA fulfilled the defined objectives.

[1] `http://www.supersede.eu`

4 CAMA: A Software Architecture for Context-aware Monitoring

4.1 Architecture (DSR Level 2)

We propose in this section CAMA, a Context-Aware Monitoring Architecture able to integrate different individual monitoring tools. CAMA is intended to support the constant changes in the context that characterize the situation of different entities including not only the services and applications that conform the context-aware system under supervision, but also its own components.

To design CAMA, we have considered several high-level requirements that we identified with the help of the use cases involved. The four first ones derive from the RQs presented in Section 1, whilst the latter four are necessary quality requirements obtained from the use cases in order to make the resulting architecture usable in practice:

- **Req1:** CAMA must provide the capability to add new monitoring tools with minimal effort. This requirement addresses RQ1.1.
- **Req2:** CAMA must provide the capability to reconfigure the monitoring tools automatically. This requirement addresses RQ1.2.
- **Req3:** CAMA must provide the capability to replace a failing monitoring tool automatically. This requirement addresses RQ1.3.
- **Req4:** CAMA must provide the capability to access the monitored data using a rich semantics method mechanism with the elements structured and defined through an ontology. This requirement addresses RQ2.
- **Req5:** CAMA must facilitate its integration with other systems in order to support activities that are built on top of context monitoring (e.g., decision-making, self-healing, self-improvement,. . .).
- **Req6:** CAMA must allow the integration of several monitoring tools that monitor different software components or devices, and are not limited to a particular scope or technology.
- **Req7:** CAMA must be secure in terms of authentication and authorization in order to grant access only to those who have the required permissions.
- **Req8:** CAMA must perform in such a manner that any malfunction or low performance of any monitor does not affect the monitored system.

To satisfy these requirements, CAMA is structured as a loosely coupled architecture that is organized as follows (see Fig. 2):

- **Group of monitoring tools.** This set of components represent the monitoring tools that gather context-related data (e.g., number of tweets per second), supporting then *Req6*. Each monitor component in the architecture represents a monitoring tool, which is composed by one or more *measure instruments* that implement a specific logic to gather data from software services, applications and physical devices (channels in Fig. 2). The measure instruments' perspective of the architecture is useful when monitoring tools provide different APIs to retrieve different aspects of software systems. Measure instruments send the gathered data as events to

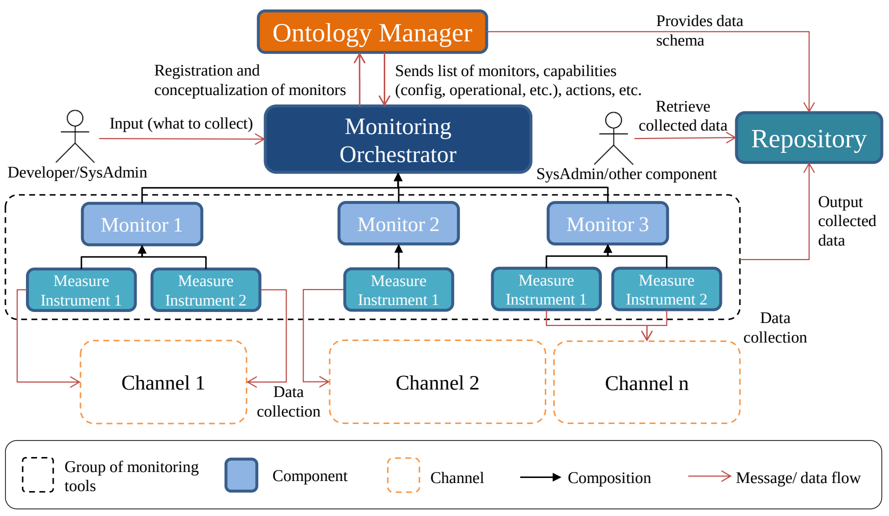

Fig. 2: Context-aware monitoring architecture

the repository, following the event-driven messaging design pattern. Monitoring tools and their measure instruments can be developed either from scratch or integrate available (commercial and open source) monitors to collect the data.

- **Monitoring orchestrator.** This component is responsible for first registering, and then orchestrating, the group of monitoring tools in the architecture. For these tasks, the monitoring orchestrator provides an interface from which a developer or system administrator (hereafter, sysAdmin) can register and integrate monitoring tools into the architecture, satisfying *Req1*. Then, the orchestration starts with the input of the developer/sysAdmin indicating "what to collect", and based on this information, the component executes actions at runtime over the monitoring tools that have been integrated into the architecture (e.g., applying assignments, changes, new configurations), allowing the monitoring tools to be reconfigured automatically. Once the monitoring tools are configured, they run independently of the orchestrator. In such a manner, a malfunction or bad performance of a monitoring tool does not affect the whole system, satisfying *Req8*. If the orchestrator fails, only the (re)configurations would be unavailable. Nevertheless, for critical systems, this can be mitigated by replicating and deploying more than one orchestrator.
- **Ontology manager.** This component provides and manages an ontology that characterizes the context of entities (agents or resources) through context information (time, location, environment, etc.). Thus, each monitoring tool that can be considered as a resource is registered and conceptualized in the ontology by considering the context information that the monitoring tool is able to supervise. The registration and conceptualization tasks start

when the orchestrator receives a subscription request of a new monitoring tool with the following parameters: name, type, description, endpoint, inputs and outputs (Table 4 provides an example of such parameters for the SocialMentionAPI monitoring tool). The ontology manager maps such parameters into classes and properties (datatype and object properties) or creates an instance if such type of monitoring tool has already been modelled. From this perspective, the ontology knows the list of monitoring tools that have been registered, the context information that a monitoring tool or mechanism is able to retrieve and provide (e.g., in the scope of the social environment, the messages of a certain social network), the operational and configuration capabilities of the monitoring tools, their status (by means of self-monitoring), as well as the actions that can be done over them. These actions that can be triggered by the reasoning capabilities of the ontology and by external components that are responsible for data analysis and decision-making have the aim of maintaining the health of both the entire architecture and the entities that are being supervised. Such mechanisms enable the detection and replacement of failing monitoring tools, and hence, satisfying *Req3*. The ontology also provides the data schema of monitored data for a unified and structured dissemination, satisfying *Req4*. To allow the dynamic addition of monitoring tools and provide the capability to deal with different monitored elements of heterogeneous monitoring tools, the ontology has been designed to be easily extensible and adaptable for each domain. More details about the ontology are given in Section 5.

– **Repository.** This component is responsible for storing the output of each monitoring tool that is structured by means of the data schema provided by the ontology through the ontology manager. The monitoring tools know the structure of such schema and can communicate with the repository through the interface provided by this component, which in turn can be used by a sysAdmin or other external component to collect the monitored data, also satisfying *Req4*. The interface of the repository component also provides the methods required for instantiating it using different types of components. For instance, it can be instantiated with Kafka for messaging systems to capture and publish data, or MySQL for data persistence, among others.

The resulting architecture is a service-oriented architecture (SOA). We consider that adopting and applying the SOA principles and patterns is the way to ensure *Req5*. Just to illustrate the consequences of such decision, the different APIs of the monitoring tools should be wrapped as RESTful microservices to provide highly decoupled services performing small tasks. Such perspective allows the reconfiguration of monitoring tools, satisfying *Req2*. Finally, to avoid unauthorized access, the communication among components can be done through an Enterprise Service Bus (ESB), which can easily include a security module handling authentication and authorization, satisfying *Req7*.

The sequence diagram to configure and run the monitoring tools is depicted in Fig. 3. The operational process of CAMA starts when a user (a developer

Table 4: example of parameters to register the SocialMentionAPI monitoring tool in the ontology

name	SocialMentionAPI
type	Social networks monitor
description	it is a social media search and analysis platform that aggregates user generated content from across the universe into a single stream of information
endpoint	http://localhost:8080/SocialMentionAPI
inputs	{keywords, timeslot, confStatus, outputFormat}
outputs	{idOutput, author, message, link, timestamp}

or sysAdmin) requests from the monitoring orchestrator (*Orchestrator*) the context information that should be collected. At this stage, the *Orchestrator* should be aware of the available monitoring tools (*Monitor*). Such task is supported by the ontology provided and managed by the *Ontology Manager*, which is responsible for registering and modelling inputs, outputs and configuration capabilities of the monitoring tools. With the information provided by the ontology, the *Orchestrator* is able to perform different (re)configuration operations over the monitoring tools. Therefore, if a *Monitor* fails and there is another one able to supervise the same context information (which is discovered thanks to the reasoning capabilities of the ontology), the failing *Monitor* can be replaced. Once the available monitoring tools are identified, the *Ontology Manager* will return them to the *Orchestrator*. The *Orchestrator* will then configure each *Monitor* accordingly, which in turn will run the *Measure Instruments* that periodically will gather and send the data to the *Repository*.

Concerning technological details, the *Monitors* to be plugged into the *Orchestrator* should be wrapped as RESTful micro-services. Finally, all the context information retrieved by the *Monitors* is stored in the repositories defined in the *Repository* component (e.g., MySQL). To store and disseminate a unified and structured set of monitored data, this component is also supported by

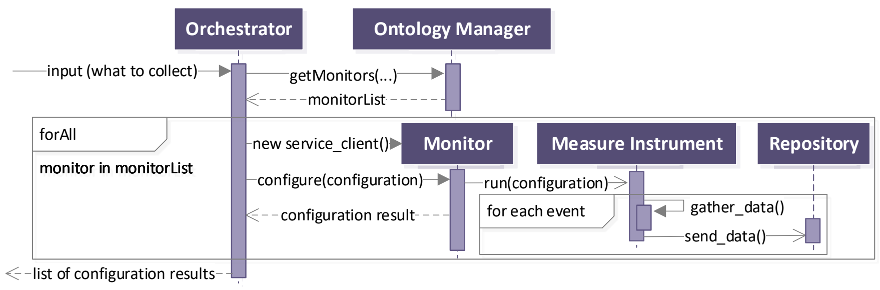

Fig. 3: Sequence diagram to configure and run the monitors

the schema of the ontology that conceptualize the outputs of the monitoring tools.

4.2 CAMA instantiated in a real use case (DSR Level 1)

In this section, we present a real use case where CAMA was applied. We use the monitoring needs of three companies (Siemens, Atos and SEnerCon) that participated in the SUPERSEDE European project. These needs were basically aligned with the state of the art as reported in Section 2. For instance, in an on-site workshop with the companies' representatives, we identified that their current technologies do not provide mechanisms to add new monitoring tools at runtime (e.g., to gather new context information). In terms of concrete operational needs, the companies were interested in monitoring: 1) user activity data, including page views, clicks, apps, ratings, etc.; 2) social networks activity, mainly when an event is being covered; 3) system stability including RAM activity, hard disk, processes, etc.

To cope with these needs, we have implemented and deployed in CAMA four types of monitoring tools for: App Marketplaces, Twitter, User events and IT infrastructure (see Fig. 4).

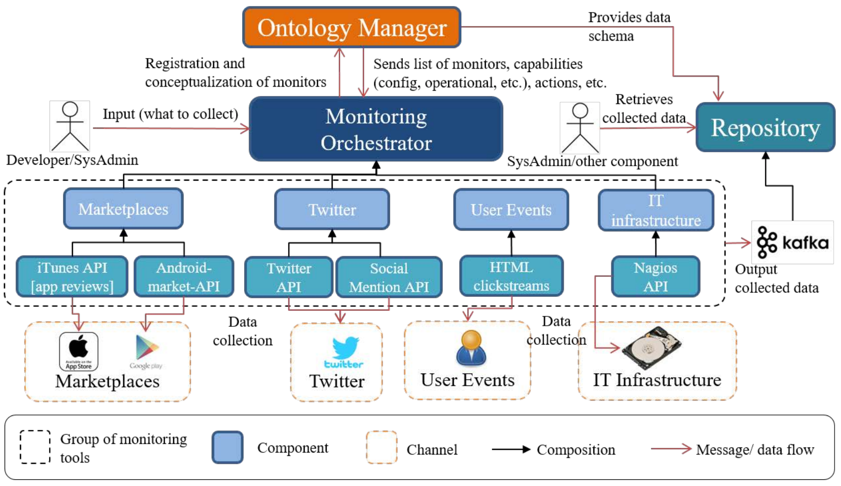

Fig. 4: Monitoring architecture instantiated in a real use case

As it can be seen in Fig. 4, the *Marketplaces* monitoring tool has been implemented through three measure instruments: *iTunesAPI*, *googlePlayAPI* and *appTweak*. The two first measure instruments use the official APIs to connect with the *Marketplaces* of iTunes and Android, respectively; whereas the last measure instrument integrates a popular tool used by more than 80000 apps for monitoring app reviews and comments. Similarly, we have also deployed in the architecture of CAMA a monitoring tool for Twitter, integrating two measure instruments to retrieve tweet's information: *Twitter API*[2] and *Social Mention API*[3]. The former is the official Twitter API, whereas the latter is a popular monitoring tool developed by socialmention.com. Marketplaces and Twitter monitoring tools were implemented and deployed in the architecture of CAMA with the aim of improving users' QoE of applications deployed in marketplaces and streaming services such as live webcasting of sport events. For instance, such streaming service was used to stream the Olympic Games 2016, and the Twitter monitoring tool was able to retrieve 18,355 tweets, which were later used for analysis to improve the QoE of such service.

Other monitoring tools that we have implemented and deployed in CAMA are the *User events* and *IT infrastructure* monitoring tools with the aim of improving users' QoE through monitoring user events listening clickstreams and the IT infrastructure (specifically, regarding hard disk monitoring). Such monitoring tools have been implemented through a specific measure instrument called *HTML clickstreams* to retrieve the user activity and *Nagios API*[4] to retrieve the disk information, respectively. For instance, regarding the *HTML clickstreams*, we were able to collect 260,637 user events in a period of 80 days.

During the execution of CAMA, the ontology handled by the ontology manager is aware of the inputs, outputs and configuration capabilities of the mentioned monitoring tools. With this information, the monitoring orchestrator can perform actions over the monitoring tools. For instance, when the *Twitter API* fails, identified by means of monitoring logs and the reasoning capabilities of the ontology, the orchestrator can change the *Twitter API* by the *Social Mention API*, which can feed the same required data in the monitoring infrastructure. Finally, the data collected by the instantiated monitoring tools is sent to Kafka, which is a publish-subscribe messaging system, and then such data is unified, structured and stored in a repository. At the end, a sysAdmin can retrieve the collected monitored data from the repository by means of the interface provided by the repository component.

5 Context ontology for CAMA

This section provides details on the context domain ontology managed by the ontology manager.

One of the well-known problems of data management, storing and data characterization is the lack of semantics (schema-less) [17]. This should be overcome by the definition of an ontology [10] that should link the inputs and outputs of monitoring tools and data sources.

[2] `https://dev.twitter.com/`
[3] `http://socialmention.com/`
[4] `https://www.nagios.org/`

5.1 Domain Context Ontology (DSR Level 2)

The proposed domain context ontology for CAMA has been built by reusing an existing generic ontology, which was presented in a previous work [11]. The main aim of such generic ontology is to be easily reused, extended and adapted for specific or generic purposes. These capabilities of the generic ontology follow three levels of abstraction: upper, middle and lower levels. In general, these levels provide classes in different levels of abstraction, being the classes of the middle level used for building domain-specific ontologies.

The methodology for building the generic ontology with the three levels of abstraction is described below:

- Upper-level ontology. It was built through a systematic mapping of different ontologies allowing to retrieve and consolidate the most abstract classes of such ontologies. In general, we applied a class hierarchy study for identifying the most used abstract classes in the first and second levels of the hierarchies specified in the studied ontologies. It allows to consolidate and define a generic structure of abstract classes and that we called upper-level ontology. The building process details are described in [10].
- Middle-level ontology. It was defined based on the upper-level ontology and structured in a modular way to enhance its reusability, following the integration process method prescribed by Pinto and Martins [36]. In general, we applied a study of different ontologies related to the classes specified in the upper-level ontology (e.g., for the Time class we studied ontologies of time). It allows to consolidate and reuse vocabulary of different ontologies in a modular way. The building process details are described in [11].
- Lower-level ontology. It is the only level that depends on the domain and it is created following a set of initial criteria and semantic principles given by the middle and upper level ontologies and through functional requirements identified by means of competence questions.

In this work, we focus on the development process of the lower-level ontology, as this is the only level that requires to be tailored for each specific domain. The other levels (middle and upper levels) have been integrated in CAMA without requiring any modification.

We have selected the classes of the first two levels (upper and middle levels, with a total of 20 classes) that lie in the heart of CAMA and are shared by all its possible monitoring instances, providing a unifying view of the monitoring infrastructures. In Fig. 5. those classes that can be reused to model any kind of monitoring tool are depicted as grey and yellow rectangles with a continuous line.

It is also important to highlight the reuse capability of the ontologies presented in the paper. Such reusability can be carried out at different levels of abstraction. For example, in the case of CAMA, we have not started from scratch, but we have reused the upper and middle levels of abstraction of the ontology and extended it with the specific Domain-level (see Fig. 5). Starting a new knowledge without any basis may cause re-building existing ontologies yielding significant cost and efforts.

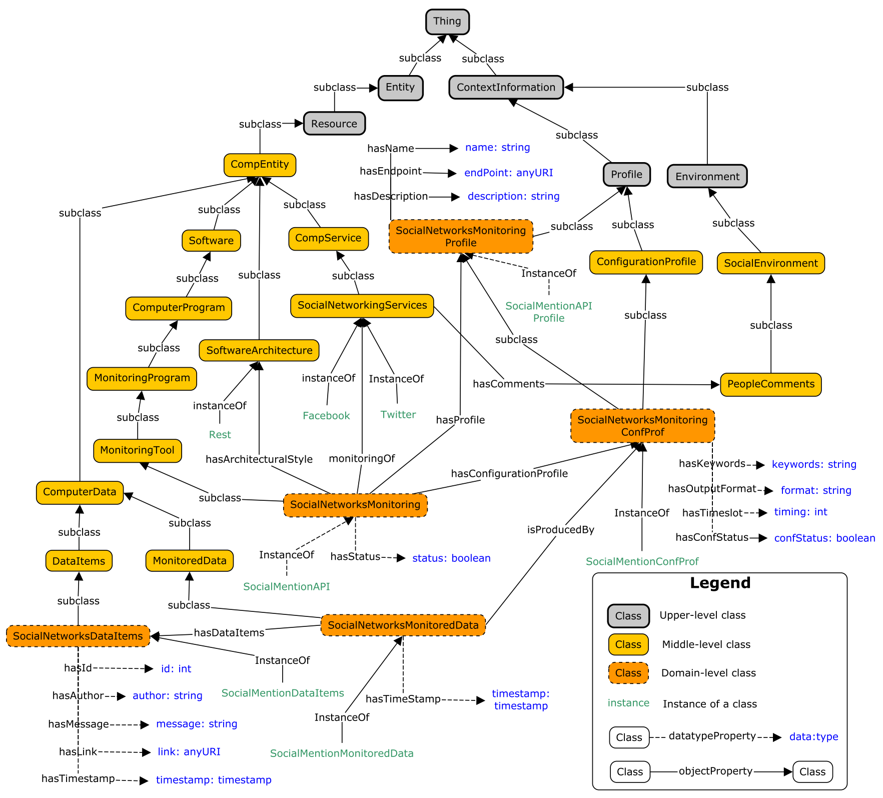

Fig. 5: Conceptualization of a monitoring tool through the CAMA ontology

5.2 Ontology instantiated in a real use case (DSR Level 1)

Considering the structure of the upper and middle levels of the proposed ontology derived from the previous task, we build its lower level (domain-specific ontology). Such third level (domain level; 5 classes) is scenario-dependent, meaning that every particular instance of CAMA will present its own ontology; this is why it corresponds to the concept of domain ontology [24]. The third level depicted in Fig. 5, represented as dotted orange rectangles, corresponds to the social networks monitoring tools.

The proposed context ontology for CAMA, including the upper, middle and lower levels, has the aim to fulfill certain functional requirements, including the conceptualization of inputs, outputs and general capabilities of a monitoring tool. According to Uschold and Gruninger, the functional requirements of an ontology are defined through competency questions that it should respond to, i.e., a type of questions that the customer is expecting that the ontology answers [41]. In this regard, the competence questions that can be queried in the proposed ontology are the following:

– What are the parameters that can be configured in a monitoring tool?
– What kind of context information can be monitored by a monitoring tool?
– What is the monitored data gathered by a specific monitoring tool?
– What kind of response format is given by a certain configuration instance?
– What instances of a monitoring tool or related configurations are activated?
– What are the monitoring tools that can monitor a specific service or application (e.g., retrieve all the monitoring tools that can monitor the Twitter or the CPU of a system)?

As depicted in the ontology proposed for CAMA (see Fig. 5), a monitoring tool has been conceptualized to represent that "any *monitoring tool* is a *monitoring program*, which in turn is, following the hierarchy of concepts, a *computer program*, a *software* and a *computational entity*". From this perspective, we can also represent that "any instance of the *SocialNetworksMonitoring* class is related to one or more instances of the *SocialNetworksMonitoringProfile* class for describing its generic profile information such as extended name, the endpoint where the monitoring service can be accessed by the orchestrator of the architecture or by another client application, the detailed description of the monitoring tool, etc.". Such modelling can be applied to any monitoring tool for representing its inputs, outputs and capabilities.

The input of the monitoring tool has been conceptualized to represent that "any instance of the *SocialNetworksMonitoring* class has a status *On* or *Off* indicating if the monitoring tool is activated or not". Each instance of the *SocialNetworksMonitoring* class can be related to one or more instances of the *SocialNetworksMonitoringConfProf* class consisting of different parameters that can be configured, such as the configuration status that represents if a certain configuration is activated or not, the keywords that are going to be searched, the response format (json, xml or csv) and so on. Note that these parameters represented as datatype properties in the ontology are intended to be generic from any social network monitoring tool.

The output of a monitoring tool (monitored data) is modelled to indicate that:

"One or more instances of the *SocialNetworksMonitoredData* class is produced by an instance of the *SocialNetworksMonitoringConfProf* class. Each instance of the *SocialNetworksMonitoredData* class has a timestamp and one or more number of data items (instances of the *SocialNetworksDataItems* class), each of them consisting of different response properties such as id (unique hash id), message, link, timestamp, image (story or item image), user (author's username), user_image (author's profile image), user_link (author's profile url), domain (the origin source's domain), source (the original source's name), and type (blogs, microblogs, etc.)".

Note that the specified response properties are an approximation of a response of social networks monitoring tools; however, not all of them are mandatory. As depicted in Fig. 5, we are expecting generic information such as the id, message, author message, timestamp and link of the monitored data.

5.3 Supporting material for building domain ontologies

In Fig. 5 we have conceptualized three levels of abstraction specifying the upper, middle and domain level classes of the CAMA ontology. For the domain level, we provide as an example a new type of monitoring tool named *SocialNetworksMonitoring* that may help the reader to analogously create a new type of monitoring tool. We describe in Table 5 the input and output data needed by each specific-level class depicted in Fig. 5. This information (hierarchy and glossary of terms) simulates the state that a modeler interested in using this ontology would find as a starting point.

Table 5: Glossary of data

Class	**Parameter/data**	**Description**
SocialNetworks Monitoring	*idSNMTool*	Identifier of the monitoring tool
	statusSNMTool	Specifies if the monitoring tool is running or not
SocialNetworks Monitoring Profile	*name*	Name of the monitoring tool
	endpointSNMTool	URI of the monitoring tool
	description	A short description of the monitoring tool
SocialNetworks Monitoring ConfProf	*idConf*	Identifier of a configuration
	confStatus	Status specifying whether the configuration is up and running or not
	accounts	Accounts that are going to be monitored
	keywords	The keywords that we want to search for
	timeslot	The timeslot between two monitoring invocations
	kafkatopic	The topic that is used for storing the monitored data in Kafka.
SocialNetworks Monitored Data	*idOutput*	Identifier of each output of the monitoring tool
	searchTimestamp	Time in which the data has been collected
	numDataItems	How many data items have been collected
SocialNetworks Data Items	*idItem*	The identifier of the item
	author	The author of the social network message
	message	The text of a social network message
	link	The link to the original message
	timestamp	The timestamp in which that message was created

6 Implementation

A prototype of the framework has been implemented in Java 8 with RESTful services built on JAX-RS and running under Apache Tomcat. The code, deployment instructions and API documentation are available in Zenodo [8].

The monitoring tools currently implemented are:

- Twitter → Twitter API: monitors in real-time the tweets that satisfy a specific search criteria (e.g., keywords, accounts, etc.) and retrieves the messages and some metadata (e.g., user, timestamp, tweetID, etc.). To obtain the tweets, this component uses the Streaming API provided by Twitter[5].
- Marketplaces → googlePlayAPI: monitors in real-time the feedback provided by users to a specific app in GooglePlay and retrieves the messages, ratings and some metadata (e.g., user, timestamp, etc.). To obtain the feedback, this component uses the Google Play Developer API[6].
- Marketplaces → iTunes API: monitors in real-time the feedback provided by users to a specific app in AppStore and retrieves the messages, ratings and some metadata (e.g., user, timestamp, etc.). To obtain the feedback, this component uses the RSS Feed Generator of iTunes[7].
- User events → HTML clickstreams: monitors the clickstream of a user in an HTML web page. To obtain the clickstream, this component uses JavaScript functions that listen to the user events.

Regarding the implementation of the ontology, all the levels and modules of the ontology and the social networks monitoring tools have been implemented separately into the Protégé editor in OWL to facilitate the reuse of the proposed resources. The implemented resources are available in Zenodo [8].

7 Evaluation

We have conducted a set of activities that evaluates how CAMA satisfies the different RQs defined (see Table 6).

Table 6: List of evaluation objectives

RQ#	Evaluation objective
RQ1.1	Assess that CAMA provides the capability to add new monitoring tools with minimal effort
RQ1.2	Assess that CAMA provides the capability to reconfigure the monitoring tools automatically
RQ1.3	Assess that CAMA provides the capability to replace a failing monitoring tool automatically
RQ2	Assess that CAMA provides the capability to access the monitored data using a rich semantics method with structured elements defined by the ontology

[5] `https://dev.twitter.com/streaming/overview`
[6] `https://developers.google.com/android-publisher/`
[7] `https://rss.itunes.apple.com/`

7.1 Evaluation of RQ1.1

This subsection describes the evaluation that assesses CAMA's capability to add new monitoring tools with minimal effort. Adding new monitoring tools does not require any change in the code of the monitoring system, and only requires registering it with its inputs and outputs in the ontology of CAMA.

To this aim, we have prepared an exercise to extend the ontology with a new monitoring tool that was filled by different practitioners (see protocol below). We assume that the new monitoring tool to be added already provides a web service interface and, therefore, there is no need to programmatically implement a web service wrapper (although it could be done if needed).

7.1.1 Protocol of the evaluation

To carry out this evaluation, we conducted the following tasks:

1. Definition of a scenario/exercise consisting of the development of a domain ontology for conceptualizing monitoring tools where inputs, outputs and general capabilities should be specified. The scenario was beta-tested by 2 researchers whose feedback helped to make the instrument fit for purpose. Details of the scenario/exercise are provided in [9].
2. Selection of the participants to run the exercise. To conduct this task, we recruited 18 researchers and practitioners from different research centers and companies with different expertise and background on monitoring of services and applications, and/or ontologies (this information is included in Table 7).
3. Explanation to the participants of the aim of this practice, providing them the supporting material described in Section 5.3, and also giving them general considerations of the scenario.
4. Execution of the exercise by the participants.
5. Analysis of results, in terms of the time taken for them to build the ontology and the quality of the result.

7.1.2 Results of the evaluation

The results of the activity are presented in Table 7 with the following variables:

- **Lev.Ont.** Level of expertise in modelling ontologies (from 1-low to 5-high).
- **Lev.Mon.** Level of expertise in monitoring systems (from 1-low to 5-high).
- **T1.** Time to read and understand the exercise and supporting material.
- **T2.** Time to elaborate and write down the solution.
- **Usability**. Subjective opinion given by the participant stating whether the proposed ontology is usable or not to conceptualize any monitoring tool.
- **Correctness.** Quality of the model designed by the participant and evaluated by the researchers with possible values:
  - **Excellent**. All needed classes and properties that maintain the model consistent were specified.

- **Very good**. All needed classes were specified; however, some property (data or object) was not specified.
- **Fair**. The model is not entirely wrong, but inconsistencies were found.
- **Poor**. The model does not represent the classes and properties that we expected, and several inconsistencies were found.

- **Improvements.** Subjective opinion through which the participant expressed whether the abstract layer of the proposed ontology (upper and middle level classes) should be improved.
- **Difficulties.** Subjective opinion in which the participant expressed whether s/he had difficulties to understand the exercise and supporting material.

Table 7: Results for the evaluation of adding new monitoring tools in the ontology

ID	**Lev.Ont**	**Lev.Mon**	**Participants**	**T1 (seconds)**	**T2 (seconds)**	**Usability**	**Correctness**	**Improvements**	**Difficulties**
P1	4	1	Researcher	390	853	Yes	Very good	No	Yes
P2	4	1	Practitioner	300	1080	Yes	Very good	No	No
P3	4	3	Practitioner	480	900	Yes	Excellent	No	No
P4	3	3	Researcher	335	932	Yes	Very good	No	No
P5	3	1	Researcher	607	716	Yes	Very good	Yes	No
P6	3	1	Researcher	660	1320	Yes	Very good	Yes	No
P7	3	2	Researcher	252	559	Yes	Excellent	Yes	No
P8	3	2	Researcher	465	895	Yes	Excellent	Yes	Yes
P9	3	4	Researcher	490	900	Yes	Excellent	No	No
P10	3	3	Researcher	240	660	Yes	Excellent	No	No
P11	2	3	Practitioner	240	600	Yes	Very good	No	No
P12	2	3	Researcher	885	1116	Yes	Very good	No	No
P13	2	1	Researcher	457	1822	Yes	Fair	No	No
P14	2	5	Researcher	476	749	Yes	Excellent	No	Yes
P15	1	1	Practitioner	921	840	Yes	Very good	No	Yes
P16	1	2	Researcher	780	1020	Yes	Very good	No	No
P17	1	3	Researcher	931	653	Yes	Excellent	No	No
P18	1	1	Practitioner	220	861	Yes	Very good	No	Yes

Visual representations of the results are depicted in Fig. 6 and Fig. 7, grouping the results by Lev. Ont. and Lev. Mon., respectively.

The analysis of the results obtained from the empirical study are described and classified as follows, considering the most relevant findings of Table 7:

- **T1 vs Lev.Ont and Lev.Mon**. We have analyzed the time to read and understand the exercise and supporting material (T1) in a two-way between-subjects ANOVA, with level of expertise in modelling ontologies (Lev.Ont) and level of expertise in monitoring (Lev.Mon) as a between-subjects variables. This ANOVA (see Table 8) tests whether there are (1)

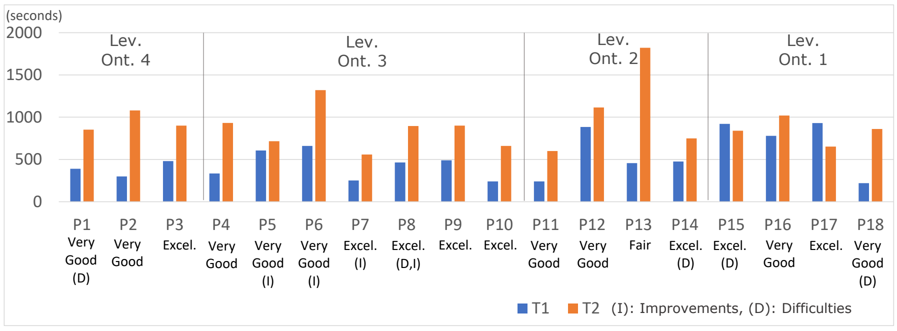

Fig. 6: Results for the evaluation of adding new monitoring tools grouped by Lev. Ont.

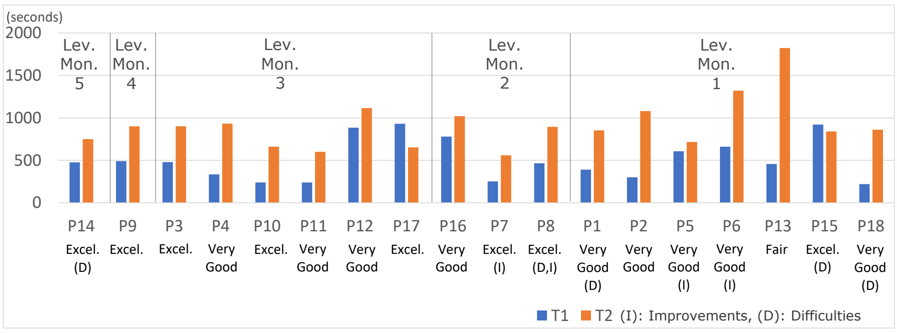

Fig. 7: Results for the evaluation of adding new monitoring tools grouped by Lev. Mon.

any differences between the T1's means for each level of Lev.Ont, (2) any differences between the T1's means for each level of Lev.Mon, and (3) any interaction between Lev.Ont and Lev.Mon. Results show that:

- The main effect of Lev.Ont on T1 was not significant, $F(3,6) = 1.015,\ p = 0.449$. Thus, the time to read and understand the exercise and supporting material is independent from the level of expertise in modelling ontologies.
- The main effect of Lev.Mon on T1 was not significant, $F(3,6) = 0.023,\ p = 0.995$. Thus, the time to read and understand the exercise and supporting material is independent from the level of expertise in monitoring.
- The Lev.Ont:Lev.Mon interaction was not significant, $F(5,6) = 0.606,\ p = 0.700$. Thus, the effects of levels of expertise in modelling ontologies and levels of expertise in monitoring combined differentially do not affect the time to read and understand the exercise and supporting material.

However, given the small number of observations, we have to be careful with drawing conclusions from these results.

Table 8: Results of ANOVA Test - T1 vs Lev.Ont and Lev.Mon

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Lev.Ont	3	246750	82250	1.015	0.449
Lev.Mon	3	5486	1829	0.023	0.995
Lev.Ont:Lev.Mon	5	245730	49146	0.606	0.700
Residuals	6	486365	81061		

- **T2 vs Lev.Ont and Lev.Mon**. We have analyzed the time to elaborate and write the solution (T2) in a two-way between-subjects ANOVA, with level of expertise in modelling ontologies (Lev.Ont) and level of expertise in monitoring (Lev.Mon) as a between-subjects variables. This ANOVA (see Table 9) tests whether there are (1) any differences between the T2's means for each level of Lev.Ont, (2) any differences between the T2's means for each level of Lev.Mon, and (3) any interaction between Lev.Ont and Lev.Mon. Results show that:
  - The main effect of Lev.Ont on T2 was not significant, $F(3,6) = 0.675,\ p = 0.598$. Thus, the time to elaborate and write down the solution did not differ between the different levels of expertise in modelling ontologies.
  - The main effect of Lev.Mon on T2 was not significant, $F(3,6) = 1.693,\ p = 0.267$. Thus, the time to elaborate and write down the solution did not differ between the different levels of expertise in monitoring.
  - The Lev.Ont:Lev.Mon interaction was not significant, $F(5,6) = 1.563,\ p = 0.307$. Thus, the effects of levels of expertise in modelling ontologies and levels of expertise in monitoring combined differentially do not affect the time to elaborate and write down the solution.

  However, given the small number of observations, we have to be careful with drawing conclusions from these results.

Table 9: Results of ANOVA Test - T2 vs Lev.Ont and Lev.Mon

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Lev.Ont	3	146872	48957	0.675	0.598
Lev.Mon	3	368212	122737	1.693	0.267
Lev.Ont:Lev.Mon	5	555589	111118	1.533	0.307
Residuals	6	434961	72494		

- **Correctness vs. difficulties.** We have evaluated the correlation between the quality level of the model designed by the participants (*correctness*) and the *difficulties* that they perceived while reading the exercise and supporting material. In this case, as *correctness* and *difficulties* are categorical variables, we conduct a Pearson's Chi-Square test of independence to examine the relation between *correctness* and *difficulties* (see Table 10). Results show that the relation between these variables was not significant, $X^2(2, N = 18) = 0.41143,\ p = 0.8141$. There is no correlation between the quality level of the model designed by the participants (*correctness*) and

Table 10: Results of Pearson's Chi-square test - correctness vs difficulties

```
        Pearson's Chi-squared test
data: COvsDItable
X-squared = 0.41143, df = 2, p-value = 0.8141
```

the *difficulties* that they perceived while reading the exercise and supporting material. Additionally, as it can be seen in the *correctness* criterion, the level of expertise in modelling ontologies (Lev.Ont) and monitoring systems (Lev.Mon) was not an important factor to impact negatively on the correctness of the models. In fact, only one participant (with Lev.Ont=2 and Lev.Mon=1) provided a solution that was not considered very good or of excellent quality. Although the model was not entirely wrong, some inconsistencies were found in his solution. These results allow us to conclude that the ontology is easy to extend and the resulting extended ontology is, in most cases, of very high quality.

– **Usability.** The findings based on the results of *usability* show that 100% of the participants, regardless of their expertise, found the proposed ontology useful, easily extensible and reusable, providing the concepts and relationships needed to conceptualize any monitoring tool. Beyond the aid of the instructions and supporting material that we have made available to the participants, we consider that this satisfactory result is due to the different features offered by the proposed ontology highlighting the intuitive and easy representation of entities and context information to characterize a monitoring tool, the precise and concise primitives needed to represent a monitoring tool, the short size that makes it easy to handle, and the three-level approach taken in a previous work [11] for abstracting the primitives and facilitate the extension of the model.

In addition to the quantitative analysis, we conducted a qualitative analysis asking participants to provide feedback on how the presented ontology could be improved. Four participants (22%) provided a suggestion to improve the proposed model. Some suggestions were about adding more attributes (i.e., data type properties) to describe this kind of monitoring tools. However, this suggestion does not affect the proposed ontology since it depends on the scenario and the modeler to include more or less attributes. Another suggestion was the creation of a middle level class named "MarketplaceService" as a subclass of "CompService" since a participant did not consider a marketplace service a type of "SocialNetworkingServices". We consider that such suggestion manifest the reusability capabilities of the proposed ontology to model any type of monitoring tool. It is worth noting that we considered correct both solutions (the proposed model and the suggested improvement) as long as this differentiation remains consistent with the rest of the model.

7.2 Evaluation of RQ1.2

This subsection describes the evaluation that assesses CAMA's capability to reconfigure the monitoring tools automatically. It is worth to remark that the analysis and decision-making process that would trigger the adaptation is out of the scope of this work. Here we evaluate how CAMA enacts a reconfiguration assuming that an external component has already conducted an analysis and decision-making process that has resulted in a reconfiguration need.

The use case selected for this evaluation is related to COVID-19. In particular, the reconfiguration of monitors to gather COVID-19 related messages shared in social media, and the reconfiguration of monitors that gather reviews of mobile apps related to COVID-19.

7.2.1 Protocol of the evaluation

The protocol of this evaluation is as follows:

1. **Selection of monitoring tools:** The monitoring tools selected for this evaluation are the ones already implemented which had reconfiguration capabilities. Namely, Twitter, GooglePlay and AppStore (as introduced in Section 4). In contrast, the monitoring tool of HTML clickstreams was not tested, as in its current implementation, this monitoring tool does not provide any parameters to reconfigure.
2. **Number of reconfiguration executions:** For each monitoring tool, we need to execute n different random reconfigurations at runtime, being n the number of executions required for statistical significance. To obtain such *n,* we used the following formula [5]:

   $$n = \frac{Z_\alpha^2 Npq}{e^2(N-1)+Z_\alpha^2 pq}$$

   where:
   - n: is the sample size needed. In our case, it is the number of reconfiguration executions required.
   - Z_α: is a constant that depends on the confidence level desired in the experiment. For a typically used 95% of confidence, Z_α is 1.96.
   - N: is the population size. In our case, the total number of reconfigurations that could be executed is unconstrained. Hence, $N = +\infty$.
   - e: is the sample error accepted for the evaluation. In our experiment, we considered 0.1.
   - p and q: are the probability of success and failure respectively. We used the typical value of 0.5 for each of them.

   Since the value of N is $+\infty$, the formula results in a $+\infty/-\infty$, which is undetermined. To resolve such indetermination we applied limit theory.

   $$\lim_{N\to+\infty} \frac{Z_\alpha^2 Npq}{e^2(N-1)+Z_\alpha^2 pq} = \frac{Z_\alpha^2 pq}{e^2} = 96.04$$

   The resulting number of reconfigurations to execute is 96.04. We decided to round that number up to 100.

3. **Frequency of reconfigurations:** We have executed the experiment in several iterations, where every iteration consists of the 100 reconfigurations per monitoring tool previously mentioned. In each iteration we used a different frequency of reconfiguration. In the first iteration, we started with a frequency of one reconfiguration per second for the Twitter monitoring tool, and one reconfiguration every 250 ms for the GooglePlay and AppStore monitoring tools. On each iteration, we reduced the time slot, until reaching 1 reconfiguration every 100 ms for Twitter, and 1 reconfiguration every 50 ms for GooglePlay and AppStore.
4. **Number of monitoring tools:** We have executed the experiment in several iterations (on top of the iterations previously described), using in each of these iterations a different number of monitoring tools running in parallel. In the first iteration, we started the experiment with 5 running monitoring tools (i.e., 5 monitoring tools being reconfigured in parallel 100 times, leading to 500 reconfigurations in the first iteration). On each iteration, we increased the number of running monitoring tools by 5, hence, subsequent iterations had 10, 15, 20, etc. monitoring tools running in parallel.
5. **Parameters to reconfigure:** Each reconfiguration changes all the parameters that can be reconfigured for that monitoring tool. For the Twitter monitoring tool, they are the timeslot and keywords. For GooglePlay and AppStore monitoring tools, they are the timeslot and App IDs. To set up 100 randomly generated configurations, we generated the time slots randomly from 1 to 10s (i.e. the time slot of the monitors to gather the data, not to be confused with the time slot of the reconfigurations in the experiment). For the Twitter monitoring tool, the keywords were obtained from the list of keywords used by Twitter in the official COVID-19 stream endpoint, which consists of 564 keywords related to COVID-19[8]. Finally, for the GooglePlay and AppStore monitoring tools, the apps were obtained from a list of official contact tracing apps of different countries. The detailed list of apps is available at [9].
6. **Measurements:** For each reconfiguration, we measure the time it takes to complete.
7. **Participants conducting the experiment:** The evaluation was executed by the first two authors of this paper guided by their supervisors (the last two authors of the paper).
8. **Infrastructure**: We deployed the monitoring tools in a virtual machine with 2.46 GHz CPU (x2), 4 GB RAM and 200 GB HDD.

The details and fine-grained data of this experiment is located at [9].

7.2.2 Results of the evaluation

Here we present the results of the evaluation following the protocol described previously. We present the average response times to reconfigure the moni-

[8] `https://developer.twitter.com/en/docs/labs/covid19-stream/filtering-rules`

tors for each time slot and number of monitoring tools being reconfigured in parallel.

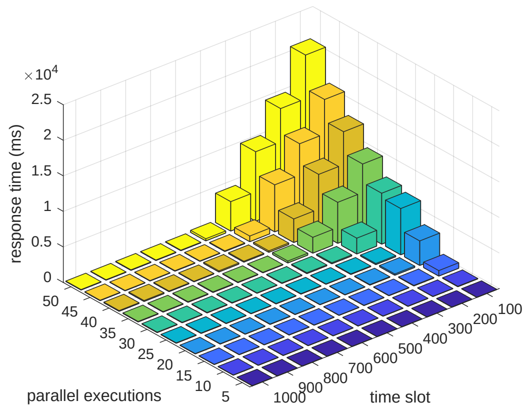

Fig. 8: Average response times to reconfigure Twitter monitoring tools

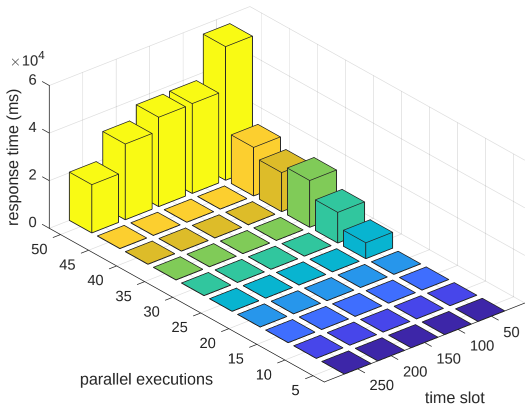

Fig. 9: Average response times to reconfigure GooglePlay monitoring tools

The results of reconfiguring the Twitter monitoring tools is presented in Fig. 8. The results show that CAMA is able to reconfigure simultaneously up to 15 monitors with a time slot of 100 ms smoothly. Between 20 and 25 monitors being reconfigured at the same time, the shortest time slot for smooth reconfigurations is at 200 ms. With 30 monitors, the limit is at 300 ms, whereas between 35 and 45 monitors, such limit is at 400 ms. With 50 parallel monitors, CAMA can handle reconfigurations with a time slot of up to 600ms (i.e., it is able to reconfigure 50 monitors simultaneously every 600 ms).

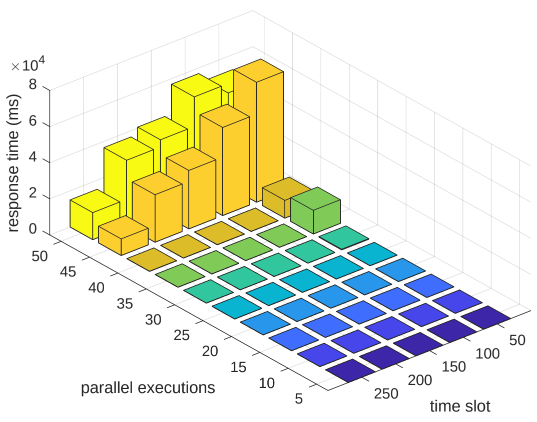

Fig. 10: Average response times to reconfigure AppStore monitoring tools

The results of reconfiguring the GooglePlay and AppStore monitoring tools are presented in Fig. 9 and Fig. 10 respectively. For the GooglePlay monitoring tools, CAMA is able to reconfigure up to 20 monitors simultaneously with a time slot of 50 ms. Between 25 and 45 monitors, CAMA is able to reconfigure them smoothly with a time slot of 100 ms, whereas with 50 monitors, the response time starts to increase significantly. For the AppStore monitoring tools, CAMA can reconfigure up to 30 monitors simultaneously with a time slot of 50 ms. Between 35 and 40 monitors, CAMA is able to reconfigure them with a time slot of 100 ms. Finally, with 45 or more monitors, the response time starts to increase significantly.

Finally, it is worth to remark that, if needed, the performance limitations can be mitigated by running the system in a more powerful machine or in multiple servers applying load balancing techniques.

7.3 Evaluation of RQ1.3

In this subsection, we describe the evaluation that assesses CAMA's capability to replace a failing monitoring tool automatically. It is worth to remark that the analysis on whether a monitoring tool is failing or not is out of scope of this evaluation. Here we evaluate how CAMA switches from one monitoring tool to another, assuming that an external component has detected a failing monitoring tool.

7.3.1 Protocol of the evaluation

The protocol of this evaluation is as follows:

1. **Selection of monitoring tools:** The monitoring tools used to replace their measure instruments are the ones for GooglePlay and AppStore, as

these two monitoring tools are the only ones that have more than one measure instrument at the current state of implementation. The measure instruments for GooglePlay are appTweak and googlePlayAPI, whereas the measure instruments for AppStore are appTweak and iTunesAPI.

2. **Number of reconfiguration executions:** As in the previous experiment, the number of required executions for statistical significance are 100. For each monitoring tool, we execute 100 replacements from a measure instrument to its alternative.
3. **Frequency of reconfigurations:** As in the previous experiment, we have executed several iterations using a different frequency for the measure instrument replacements in each iteration. We started from one measure instrument replacement every 250 ms and reduced the time slot on every iteration until reaching one measure instrument replacement every 50 ms.
4. **Number of monitoring tools:** We have executed the experiment in several iterations (on top of the iterations previously described), using in each of these iterations a different number of monitoring tools with their measure instruments being replaced in parallel, starting from 5 until reaching 50.
5. **Parameters to reconfigure:** We replace the measure instruments from the original measure instrument to the alternative, and vice versa. Since such replacements need to take place in a running configuration we use the same randomly generated configurations used in the previous experiment.
6. **Measurements:** For each replacement, we measure the time to complete the replacement.
7. **Participants conducting the experiment:** As in the previous experiment, the evaluation is executed by the first two authors of this paper guided by their supervisors (the last two authors of the paper).
8. **Infrastructure**: As in the previous experiment, we deployed the monitoring tools in a virtual machine with 2.46 GHz CPU (x2), 4 GB RAM and 200 GB HDD.

The details and fine-grained data of this experiment is located at [9].

7.3.2 Results of the evaluation

Here we present the results of the evaluation following the protocol described previously.

The results of replacing a measure instrument is presented in Fig. 11 for the GooglePlay monitoring tool and Fig. 12 for the AppStore.

For the GooglePlay monitoring tool, the results show that CAMA is capable of replacing the measure instrument of up to 15 monitoring tools in parallel every 50 ms. For 20 to 50 monitoring tools, CAMA is capable of replacing all their measure instruments in parallel every 100ms with an adequate response time.

For the AppStore monitoring tool, CAMA has been able to replace the measure instruments of up to 30 monitoring tools in parallel every 50 ms. For

35 to 45 monitoring tools, CAMA is capable of replacing all their measure instruments every 100ms, whereas with 50 monitors, the response time starts to degrade quickly. Nevertheless, we argue that in an environment where this amount of replacements are required, the system can be deployed in multiple servers applying load balancing techniques.

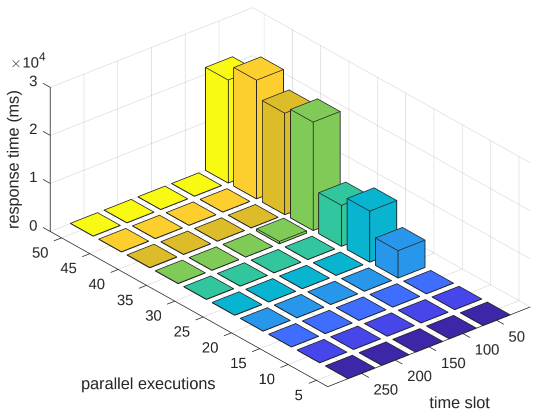

Fig. 11: Average response times to replace a Measure Instrument of the Google-Play monitoring tool

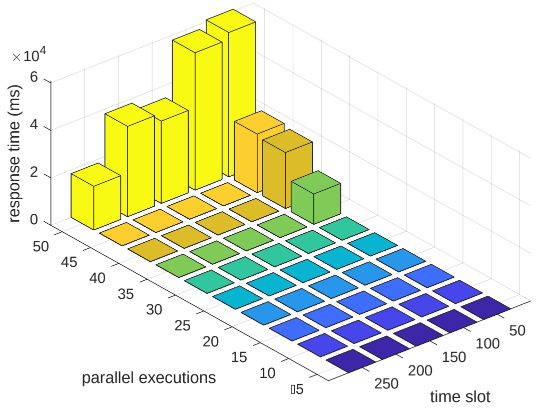

Fig. 12: Average response times to replace a Measure Instrument of the App-Store monitoring tool

Table 11: SPARQL query for retrieving the list of monitors and their status by configuration

SPARQL query:

```
SELECT ?MonitorsList ?MServices ?MonitoringInstances ?Status
?AppIDs
WHERE {
?MonitorsList e:isMonitoringToolOf ?SocialNetworkingServices .
?MonitorsList e:isMonitoringToolOf ?MServices .
?MonitorsList e:hasConfigurationProfile ?MonitoringInstances .
?MonitoringInstances e:confStatus ?Status .
?MonitoringInstances e:appID ?AppIDs
}
```

MonitorsList	MServices	MonitoringInstances	Status	AppIDs
GooglePlayAPI	MarketPlaces	GooglePlayConfProf1	On	be.sciensano.coronalert
GooglePlayAPI	MarketPlaces	GooglePlayConfProf2	Off	com.NIC.covid19
GooglePlayAPI	MarketPlaces	GooglePlayConfProf3	On	cz.covid19cz.erouska
GooglePlayAPI	MarketPlaces	GooglePlayConfProf4	On	com.gha.covid.tracker
SocialMentionAPI	Twitter	SocialMentionConfProf1	On	
TwitterAPI	Twitter	TwitterConfProf1	Off	
TwitterAPI	Twitter	TwitterConfProf2	On	
AppTweakAPI	MarketPlaces	AppTweakConfProf1	On	com.hamagen
AppTweakAPI	MarketPlaces	AppTweakConfProf2	On	com.moc.gh
AppTweakAPI	MarketPlaces	AppTweakConfProf3	Off	1503717224
iTunesAPI	MarketPlaces	iTunesConfProf1	On	1499780720
iTunesAPI	MarketPlaces	iTunesConfProf2	On	1511740371

7.4 Evaluation of RQ2

In this subsection, we summarize the evaluation that assesses CAMA's capability to access the monitored data using a rich semantics method with the elements structured and defined through an ontology.

We evaluate that the knowledge represented in the ontology is consistent with its scope, i.e., that the ontology has enough information to answer the competence questions specified in Section 5. The expected results should demonstrate that the levels and modules, the extension of the ontology, and the conceptualization of the domain ontology are correct and consistent. After applying such evaluation, the results showed that the definition of the domain ontology was correct and consistent with the required functional requirements. The reader may refer to the annex [9] for more details about the protocol and results of the evaluation, which have been omitted due to space limitations.

The ontology of CAMA has mainly two roles: the role of responding the competence questions defined in Section 5.2 through queries in SPARQL, and the role of reasoning for deducing new context knowledge. In the first role, CAMA triggers a SPARQL query automatically through its monitoring orchestrator component when a Developer/SysAdmin makes a request of what to collect. Humans also can carry out such role manually outside the architecture of CAMA, i.e., SPARQL queries can be done manually directly to the ontology by using ontology editors (e.g., Protege). Whereas the second role can be executed only automatically by the internal rules, semantics and reasoner engines implemented in the ontology. For instance, the SPARQL query showed in Table 11 can be executed manually by humans or automatically by CAMA to retrieve the list of monitors registered in the ontology, the monitored services related to the list of monitors, the monitoring instances with their status (i.e., if an instance of a monitor is working or not) and the list of apps monitored by the instances. In the second role, the ontology can automatically detect if a monitoring tool of the SocialNetworksMonitoring class is not working properly (e.g., by detecting if the data items retrieved by a specific instance of the SocialNetworksMonitoringConfProf class are null), or if a reconfiguration of the keywords are needed because there are too many data items being collected.

7.5 Threats to validity

In this section we discuss the threats to validity of our evaluation and the actions we have taken to mitigate them.

1. **Internal validity**. The internal validity of the evaluation concerns our ability to draw conclusions from the conducted experiments and the outcomes observed. To mitigate such threat we applied several statistical methods. In the evaluation of RQ1.1, we applied ANOVA and Chi-square tests to assess if the differences observed were statistically significant. In the evaluation of RQ1.2 and RQ1.3, we used statistical methods to compute the required number of executions for statistical significance, and executed our experiments accordingly.
2. **Construct validity**. In the evaluation of RQ1.1, there could be a potential bias of the study participants when answering the questions. To mitigate such risk, study participants were not limited to members of the EU project SUPERSEDE, but other researchers and practitioners from outside the consortium participated also. On the other hand, results of the evaluation were not part of any deliverable or document of the project, assessing that there were no conflicts of interest by the participants belonging to the consortium. In the evaluation of RQ1.2 and RQ1.3, we generated and executed the reconfigurations parameters randomly, to avoid any possible bias on the characteristics of such reconfigurations.
3. **Conclusion validity**. As remarked in the evaluation of RQ1.1, due to the small number of observations, we have to be careful with drawing conclusions from the results of the evaluation of such RQ. RQ1.2 and RQ1.3 do not present this threat.
4. **External validity**. External validity refers to the generalizability of our conclusions. In RQ1.1, we have evaluated how participants could extend

the ontology to model other monitoring tools, such as a marketplace monitoring tool. Extending the ontology for other monitoring tools would be analogous, but further work is required to assess that extending the ontology to other monitoring tools would provide similar results. For RQ1.2 and RQ1.3 we have executed our experiments for the GooglePlay and AppStore monitoring tools, as they are the only ones that have more than one measure instrument at the current state of implementation. Further experiments with more monitoring tools and measure instruments are needed to ensure the external validity of our proposal. In any case, for large-scale systems demanding high amount of resources, we could apply load-balancing techniques to prevent overloads.

8 Conclusions

In this paper, we have presented CAMA, a context-aware monitoring architecture for context acquisition, which builds upon a context ontology for modelling inputs, outputs and capabilities of monitoring tools, and data management. Such contribution has been developed for supporting highly dynamic environments that affect different entities including the monitoring infrastructure, services and applications. For this purpose, CAMA answers the research questions of the paper as follows:

RQ1. CAMA provides the capabilities of replacing, adding, reconfiguring and adapting monitoring tools by means of a decoupled architecture based on services, which can be orchestrated with the support of an ontology that includes the reasoning capabilities and the provisioning of a clear schema of parameters to configure each monitoring tool integrated in the architecture. CAMA follows SOA principles, enabling extensions to the architecture to obtain, process and store the data through different methods and techniques.

RQ2. CAMA provides the capability to access the monitored data using a rich semantics method by means of a context domain ontology. Such ontology provides a unified and representative schema of the monitored data produced by the monitoring tools and their related configuration instances.

To demonstrate the feasibility and potential of the proposed approach, we investigated on real requirements from three companies. On top of that, we evaluated the feasibility of the proposed ontology of CAMA by means of an empirical study where different criteria were evaluated. We concluded that the time consumed by different participants for modelling monitoring tools is really low. Furthermore, the level of expertise in both modelling ontologies and monitoring systems was not an important variable to manage the proposed ontology and supporting material.

One of the main limitations of context-aware monitoring is that it is not possible to know a priori all the possible contexts in which it might be used.

To tackle this limitation, we have designed the architecture of CAMA and its ontology in the most generic way possible. On the one hand, the architecture can add new monitoring tools by means of a decoupled architecture. On the other hand, the ontology can be extended by using different levels of abstraction. Nevertheless, the current implementation has only been tested for Social Network monitoring and further validation is needed.

As future work, we will integrate more types of physical and logical monitoring tools, considering complex scenarios of Smart Cities and Internet of Things. Additionally, we are planning to support different context-aware scenarios including ranking and adaptation of services and applications; evolution and adaptation of personalized software; and improving the QoE of the user.

Acknowledgements This work was partially supported by the Spanish project GENESIS TIN2016-79269-R, and SUPERSEDE project, funded by the European Union's Information and Communication Technologies Programme (H2020) under grant agreement no 644018.

References

1. Ahmed TM, *et al.* (2016) Studying the effectiveness of application performance management (apm) tools for detecting performance regressions for web applications: An experience report. In: MSR'16
2. Alenezi M (2020) Ontology-Based Context-Sensitive Software Security Knowledge Management Modeling. International Journal of Electrical and Computer Engineering 10(6)
3. Alirezaie M, *et al.* (2017) An ontology-based context-aware system for smart homes: E-care@home. Sensors 17(7)
4. Baldauf M, Dustdar S, Rosenberg F (2007) A Survey on Context-Aware Systems. International Journal of Ad Hoc Ubiquitous Computing 2(4)
5. Berenson ML (1996) Basic Business Statistics: Concepts and Applications, 6th edn. Pearson
6. Bettini C, Maggiorini D, Riboni D (2007) Distributed Context Monitoring for the Adaptation of Continuous Services. World Wide Web 10(4)
7. Brebner PC (2016) Automatic performance modelling from application performance management (apm) data: An experience report. In: ICPE'16
8. Cabrera O, *et al.* (2021) A Context-Aware Monitoring Architecture for Supporting System Adaptation and Reconfiguration - Software code and ontology. `https://doi.org/10.5281/zenodo.4482144`
9. Cabrera O, *et al.* (2021) A Context-Aware Monitoring Architecture for Supporting System Adaptation and Reconfiguration - Supporting material. `https://doi.org/10.5281/zenodo.4478866`
10. Cabrera O, Franch X, Marco J (2017) Ontology-based context modeling in service-oriented computing: A systematic mapping. Data & Knowledge Engineering 110

11. Cabrera O, Franch X, Marco J (2019) 3LConOnt: a three-level ontology for context modelling in context-aware computing. Software & Systems Modeling 18(2)
12. Cho K, *et al.* (2008) HiCon: a hierarchical context monitoring and composition framework for next-generation context-aware services. IEEE Network 22(4)
13. Copetti A, *et al.* (2009) Intelligent context-aware monitoring of hypertensive patients. In: PervasiveHealth'09
14. Esposito A, *et al.* (2008) A Framework for Context-Aware Home-Health Monitoring. In: UIC'08
15. Gregor S, Hevner AR (2013) Positioning and Presenting Design Science Research for Maximum Impact. MIS Quarterly 37(2)
16. Gu T, Pung HK, Zhang DQ (2005) A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications 28(1)
17. Gudivada VN, Baeza-Yates R, Raghavan VV (2015) Big Data: Promises and Problems. Computer 48(03)
18. Heger C, *et al.* (2017) Application performance management: State of the art and challenges for the future. In: ICPE'17
19. Hong JY, Suh EH, Kim SJ (2009) Context-aware systems: A literature review and classification. Expert Systems with Applications 36(4)
20. Jatoba LC, *et al.* (2008) Context-aware mobile health monitoring: Evaluation of different pattern recognition methods for classification of physical activity. In: IEMBS'08
21. Kang S, *et al.* (2008) SeeMon: Scalable and Energy-Efficient Context Monitoring Framework for Sensor-Rich Mobile Environments. In: MobiSys'08
22. Kang S, *et al.* (2010) A Scalable and Energy-Efficient Context Monitoring Framework for Mobile Personal Sensor Networks. IEEE Transactions on Mobile Computing 9(5)
23. Kang S, *et al.* (2010) Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments. In: PerCom'10
24. Kishore R, Sharman R (2004) Computational Ontologies and Information Systems I: Foundations. Comm of the Association for Inf Systems 14
25. Lee Y, *et al.* (2012) CoMon: Cooperative Ambience Monitoring Platform with Continuity and Benefit Awareness. In: MobiSys'12
26. Lee Y, *et al.* (2012) MobiCon: A Mobile Context-Monitoring Platform. Communications of the ACM 55(3)
27. Lee Y, *et al.* (2012) MobiCon: Mobile context monitoring platform: Incorporating context-awareness to smartphone-centric personal sensor networks. In: SECON'12
28. Lee Y, *et al.* (2016) CoMon+: A Cooperative Context Monitoring System for Multi-Device Personal Sensing Environments. IEEE Transactions on Mobile Computing 15(8)
29. Mileo A, *et al.* (2010) A Logical Approach to Home Healthcare with Intelligent Sensor-Network Support. The Computer Journal 53(8)

30. Mileo A, Merico D, Bisiani R (2010) Support for context-aware monitoring in home healthcare. Journal of Ambient Int and Smart Environments 2(1)
31. Murao K, *et al.* (2009) A Context-Aware System That Changes Sensor Combinations Considering Energy Consumption. In: Pervasive'09
32. Murao K, Terada T, Nishio S (2010) Toward Construction of Wearable Sensing Environments. In: Wireless Sensor Network Technologies for the Information Explosion Era, Springer
33. Oriol M, *et al.* (2018) Fame: Supporting continuous requirements elicitation by combining user feedback and monitoring. In: RE'18
34. Papazoglou MP, van den Heuvel WJ (2007) Service oriented architectures: approaches, technologies and research issues. The VLDB Journal 16(3)
35. Perera C, *et al.* (2014) Context Aware Computing for The Internet of Things: A Survey. IEEE Communications Surveys Tutorials 16(1)
36. Pinto HS, Martins JP (2001) A methodology for ontology integration. In: K-CAP'2001
37. Rhayem A, Mhiri MBA, Gargouri F (2020) Semantic web technologies for the internet of things: Systematic literature review. Internet of Things 11
38. Riva O (2006) Contory: A Middleware for the Provisioning of Context Information on Smart Phones. In: Middleware'06
39. Shen Y, *et al.* (2020) Implementation and application of APM monitoring system under big data of power grid. IOP Conference Series: Materials Science and Engineering 719
40. Sim J, Lee Y, Kwon O (2014) Context-aware enhancement of personalization services: A method of power optimization. Expert Systems with Applications 41(13)
41. Uschold M, Gruninger M (1996) Ontologies: principles, methods and applications. Knowledge Engineering Review 11(2)
42. Villegas NM, Müller HA (2010) Context-Driven Adaptive Monitoring for Supporting SOA Governance. In: MESOA'10
43. Villegas NM, *et al.* (2010) DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II
44. Villegas NM, Müller HA, Tamura G (2011) Optimizing run-time SOA governance through context-driven SLAs and dynamic monitoring. In: MESOCA'11
45. Zavala E, Franch X, Marco J (2019) Adaptive monitoring: A systematic mapping. Information and Software Technology 105