Ubiquitous Integration of Cooperating Objects

STAMATIS KARNOUSKOS
SAP Research
and
VLADIMIR VILLASEÑOR
Tampere University of Technology
and
MARCUS HANDTE
Universität Duisburg-Essen
and
PEDRO JOSÉ MARRÓN
Universität Duisburg-Essen

Billions of devices are expected to be online by 2020. These will not only provide information by monitoring the real-world, but create complex collaborations in order to provide sophisticated value-added services. Slowly, we are witnessing the emergence of Cooperating Objects in the Internet of Things, which will rapidly change the way we design, develop and realize cyber-physical dependent applications. We investigate which requirements this poses, and evaluate several middleware systems which we have used in the past. We propose an architecture that is used to integrate cooperating objects and enable their collaboration, and depict this in a demonstration example. Finally we prioritize the requirements, and discuss on future directions that could be followed.

Categories and Subject Descriptors: H.4.0 [**Information Systems Applications**]: General; K.6.0 [**Management of Computing and Information Systems**]: General

General Terms: Design, Management

Additional Key Words and Phrases: Cooperating Objects, Cyber-Physical Systems, Enterprise Integration, Internet of Things, Middleware, Web technologies

1. COOPERATING OBJECTS

The core idea behind amalgamating the physical and virtual (business) world is to seamlessly gather useful information about objects of the physical world and use the information in various applications in order to provide some added value. As we are moving towards the "Internet of Things" (IoT), [**?**] millions of devices will be interconnected and will cooperate, providing and consuming information available on the network. Since these devices need to interoperate, the service-oriented approach seems to be a promising solution for building systems; i.e., each device offers its functionality as one or more services, while in parallel they may discover and invoke new functionality from other services on-demand [Guinard et al. 2010]. Cooperating Objects are an integral part of the future IoT. The latter is expected to enable unprecedented interconnection of networked embedded devices and further blur the line between the real and virtual world.

A number of different system concepts have become apparent in the broader context of embedded systems over the past couple of years such as pervasive and ubiquitous computing and most recently wireless sensor networks (WSN) that sense their environment (monitoring) and manage it (control). In the last years, these systems have moved from standalone operation towards collaboration, i.e. cooperation among them and with third-party provided services in order to take advantage of additional knowledge and realize sophisticated functionality and system-wide goals.

All these three types of systems (i.e. embedded systems, pervasive and ubiquitous computing and wireless sensor networks) that act and react on their environment are actually quite diverse, novel systems that, on the one hand, share some principal commonalities and, on the other hand, have some different aspects that complement each other to form a coherent group of objects that cooperate with each other to interact with their environment. In particular, important notions such as control, heterogeneity, wireless communication, dynamics/ad-hoc nature, and cost are present to various degrees in each of these types of systems.

According to the Cooperating Objects Network of Excellence (CONET) [Marrón et al. 2011], Cooperating Objects consist of embedded computing devices equipped with communication, as well as sensing or actuation capabilities that are able to cooperate and organize themselves autonomously into networks to achieve a common task. However, it is important to notice that an object may refer not only to a standalone physical device, but also to a business application masked as a physical device. The vision of Cooperating Objects is to tackle the emerging complexity by cooperation and modularity. Towards this vision, the ability to communicate and interact with other objects and/or the environment is a major prerequisite. While in many cases cooperation is application specific, cooperation among heterogeneous devices can be supported by shared abstractions.

Achieving enhanced system intelligence by cooperation of smart embedded devices pursuing common goals is relevant in many types of perception and system environments. In general, such devices with embedded intelligence and sensing/actuating capabilities are heterogeneous, yet they need to interact seamlessly and intensively over wired and/or wireless networks. More constrained devices may also cooperate with more powerful (or less congested) neighbors to meet service requests, opportunistically taking advantage of global resources and processing power. Independently of the structuring level (weakly structured or highly structured), process-driven applications make use of different kinds of data resources and combine them to achieve the application task.

Cooperation between objects can be understood in the following context:

— Two (or more) objects (i.e., object-to-object or object-to-business) are able to engage into a conversation in a loosely-coupled manner.

— The objects have a common understanding of well-defined communication patterns and protocols.

— The objects are able to exchange data relevant to their capabilities and needs.

— The objects share computational resources when needed by means of information migration or data mash-ups.

— The objects are able to cluster in order to create distributed data gathering/processing platforms.

Another interesting trend is the evolution towards global service-based infrastructures which rely on the Service-Oriented Architecture (SOA) paradigm. As such, new functionality is introduced by combining services in a cross-layer form, i.e. services relying on the enterprise system, on the network itself and at device level can be combined in order to create more sophisticated ones. New integration scenarios can be applied by orchestrating the services in context-specific ways. In addition, sophisticated services can be created at any layer (even at device layer) taking into account and based only on the provided functionality of other entities that can be exposed as a service. In parallel, dynamic discovery and peer-to-peer communication will allow to optimally exploit the functionality of a given device. It is clear that we move away from isolated stand-alone hardware and software solutions towards more cooperative models. However, in order to achieve that, several challenges need to be tackled.

The increasing computing capabilities of embedded devices will allow the implementation of new software for completely novel processes. Enabled by software, the IoT will provide for virtually infinite integration of sensors, actuators, microsystems, mechatronic systems, and robots. The world market for technologies, products, and applications alone that are related to what the IoT enables — i.e., monitoring and control (M&C) — will increase significantly. In fact, it is expected [European Commission DG Information Society & Media 2008] that the world M&C market will grow from 188 Bn€ in 2007 to 500 Bn€ in 2020.

The convergence of solutions and products towards the SOA paradigm adopted for smart embedded devices contributes to the improvement of the reactivity and performance of industrial processes [Karnouskos et al. 2010], such as manufacturing, logistics, and others. This will lead to information being available in near real-time based on asynchronous events, and to business-level applications that are able to use high-level information for various purposes, such as diagnostics, performance indicators, traceability, etc. SOA-based vertical integration will also help to reduce the cost and effort required to realize a given business scenario as it will not require any device drivers or third-party solutions. To realize the next generation IoT applications, we need to focus on how to interact with cooperating objects i.e. on the Ubiquitous Integration of Cooperating Objects (UICO) which we will analyze in more detail in the subsequent sections.

The domain of Cooperating Objects is still at its dawn; however, its impact is estimated to be so broad and significant that it could change future applications and services drastically. Numerous market analyses also point out this direction. It is important to understand that Cooperating Objects is a huge domain with applications spawning several fields, and, therefore, it is very difficult to set the limits and estimate its total value. However, the issue of ubiquitous integration is common to all domains and seen as a key challenge that must be overcome to realize cooperation and collaboration.

2. REQUIREMENTS FOR COOPERATING OBJECTS

There are several requirements that have to be tackled at a sufficient level in order to enable easy integration of Cooperating Objects. Basically we see two modes of cooperation:

— *Standalone*: devices discover and interact with each other on a standalone mode without significantly depending on the existence of third-parties; this is expected to be the case for generic forms of interaction, mostly locally. Examples for past research projects that focused on enabling this type of cooperation are 3PC [Becker et al. 2003], [Becker et al. 2004], MundoCore [Aitenbichler et al. 2007], and P2PComp [Ferscha et al. 2004], to name a few.

— *Infrastructure assisted*: devices cooperate with each other and third-parties by heavily depending on infrastructure services; these can be sophisticated interactions not restricted at the local level or to the own device's capabilities. Past research projects focusing on this type of cooperation are GAIA [Román and Campbell 2000], AURA [Cheng et al. 2002], ALLOW [Herrmann et al. 2008], and O2S [Mazzola Paluska et al. 2008], for example.

Adding cooperation in the context we analyzed, makes it imperative to have a look from a different angle, i.e., that of integration with the goal of cooperation. Based on our experiences coming from multiple industry and R&D projects, we see several requirements for the UICO:

Requirement 1 – Dynamic collaboration. Devices with sensing and/or actuating capabilities and embedded intelligence should be able to dynamically collaborate in the environment and provide services to the user (e.g. a service, another device or an end-user). As dynamic collaboration is the foundation for any cooperation, this requirement is frequently tackled by existing middleware developments such as [Aitenbichler et al. 2007], [Becker et al. 2003], or [Román et al. 2001].

Requirement 2 – Extensibility. Flexible support for extending the capabilities of a device is needed. Cooperating Objects is a rapidly developing domain and implementation should take future growth into consideration. Since extensions can be made through the addition of new functionality or modification of existing one, support for change should be provided while minimizing impact to existing system functions. One possibility to achieve this is to support protocol composition [Handte et al. 2003] or to rely on adequate programming abstractions [Becker et al. 2004], [Ferscha et al. 2004].

Requirement 3 – Resource utilization. Optimal management of resources at the local (device) as well as non-local (groups, global view) level is needed. As most of the cooperating devices are expected to be resource-constrained, the resource utilization should be considered and possibly captured in a cooperation context. For instance, it should be possible that resource-scarce devices exploit the capabilities of devices with more resources, and opportunistically take advantage of the resources in the surroundings if it makes sense from the strategy or performance viewpoint. So far, such optimizations have been done only with respect to individual system functions, e.g. [Schiele et al. 2004] or [Handte et al. 2007].

Requirement 4 – Description of objects (interface). Another requirement is an implementation independent description of the object that can be used by both implementers and requesters. This will facilitate decoupling of design and actual implementation, which will enable cooperating concepts to be developed in a loosely coupled way with respect to the actual software and hardware available. Due to its general usefulness, this requirement is realized by many existing integration systems, e.g. [Object Management Group 2004] or [Sun Microsystems 2004].

Requirement 5 – Semantic description capabilities. Semantics and ontologies should be used to enforce the dynamic interpretation of things and as an input for reasoning systems. An object should be able to not only understand that cooperation is possible, but also to assess what impact the cooperation might have, e.g., on the resources, time, processor utilization, etc. Thereby, it should describe constraints of capabilities of the specific cooperation. So far, only few existing integration approaches such as PECES [Haroon et al. 2009] attempt to tackle this requirement; however, it is important to notice that the use of semantic descriptions has been long researched at industrial scale, for example in the manufacturing industry [**?**], [**?**], from where it has been possible to obtain experience about its potential application in embedded devices.

Requirement 6 – Inheritance/polymorphism. To simplify programming via code reuse, it would make sense to have a way to form new objects using objects that have already been defined. At a later stage one can move towards the Composite Reuse Principle which enables polymorphic behavior and code reuse by containing other classes which implement the desired functionality which is partially addressed by [Mazzola Paluska et al. 2008] or [Becker et al. 2004].

Requirement 7 – Composition/orchestration. As basis for cooperation, generation and execution of work-plans between objects, services and other resources in order to promote their interaction should be supported. One example for this type of orchestration are the adaptable flows implemented in ALLOW [Herrmann et al. 2008].

Requirement 8 – Pluggability. Due to the continuous evolution of future systems, ubiquitous integration will require the dynamic interaction with newly plugged-in and previously unknown objects. This refers not only to software but also to hardware; typical examples include communication, computation, behavior, etc. and calls for a component-based approach where things can be combined to customize existing behavior or to deliver more complex ones. Cooperating objects supporting pluggability will enable third-party developers to create capabilities to extend them, easy ways of adding new features, reduced size and independent application development, etc. On the software side, this can be addressed by approaches such as Speakeasy [Edwards et al. 2002].

Requirement 9 – Service discovery. Cooperating Objects must support a mechanism for each node to make its services known to the system and also to allow querying for services. Automatic service discovery will allow us to access them in a dynamic way without having explicit task knowledge and the need of a priori binding. The last would also enable system scalability and support the composable approach of services. However, existing approaches such as [UPnP Forum 2008] or [Sun Microsystems 2006] mostly focus on low level aspects of this process.

Requirement 10 – (Web) service direct device access. Enterprise applications must be able not only to discover but, in many cases, also to communicate directly with devices, and consume the services they offer [Karnouskos et al. 2010]. The need to bypass intermediates and directly acquire specific data from the device may offer business benefits and rapid development, deployment, and change management.

Additional support, e.g. the capability of event notifications from the device side to which other services can subscribe, may provide optimization advantages.

Requirement 11 – (Web) service indirect device access (gateway). Gateways might glue to the Cooperating Objects infrastructure devices by hiding heterogeneity and resource scarceness. However, most efforts in the research domain today focus on how to open the device functionality to the enterprise systems, yet, the opposite, i.e., the opening of enterprise systems to the devices, might also be beneficial [Karnouskos et al. 2010]. For instance, devices should be able to subscribe to events and use enterprise services; this can be achieved by creating "virtual devices" that proxy an enterprise service. Having achieved that, business logic executing locally on devices can now take decisions not only based on its local information, but also on information from enterprise systems.

Requirement 12 – Brokered access to events. Events are a fundamental pillar of a service-based infrastructure; therefore access to these has to be eased. As many devices are expected to be mobile, and their on-line status often changes (including the services they host), buffered service invocation should be in place to guarantee that any started process will continue when the device becomes available again (opportunistic networking). Also, since not all applications expose (web) services, a pull point should be realized that will offer access to infrastructure events by polling [Spiess et al. 2009]. Minimized resource usage on the device by delegating access to a more powerful device/system will be beneficial.

Requirement 13 – Service life-cycle management. In future infrastructures, various services are expected to be installed, updated, deleted, started, and stopped. Therefore, the requirement is to provide basic support on-device/in-infrastructure that can offer an open way of handling these issues [**?**], [**?**].

Requirement 14 – Legacy device integration. Devices of older generations should be also part of the new infrastructure. Although their role will be mostly providing (and not consuming) information, we have to make sure that this information can be acquired and transformed [Karnouskos et al. 2010] to fit in the new service-enabled infrastructure. The latter is expected to be achieved via the wrapping of them, for example, using web services. An alternative to this is to use extensible protocol composition [Handte et al. 2010], [Aitenbichler et al. 2007].

Requirement 15 – Historian. In an information-rich infrastructure, a continuous logging of relevant data, events, and the history of devices is needed. The historian is needed to offer logging of information to services, especially when an analysis of up-to-now behavior of devices and their services is needed, for example, to support system audits.

Requirement 16 – Device management. Service-enabled devices will contain both, static and dynamic data. This data can now be better and more reliably integrated, e.g. into enterprise systems. However, in order to manage large infrastructures, a common way of applying basic management tasks is needed [**?**], [**?**]. The device management requirement makes sure that at least on the middleware side, there is a way to hide heterogeneity and provide uniform access to a device's and infrastructure's life cycle.

Requirement 17 – Security, trust and privacy. Security, trust and privacy mechanisms should be considered. Access to the devices and their services will depend on the deployed security context and, therefore, basic functions should be supported. Trust relationships will need to be considered and built upon. Similarly, privacy should be preserved especially for devices operating in sensitive user areas, e.g. hospitals, households, etc. This requires the development of new or the adaptation of existing methods to new application areas [Apolinarski et al. 2010], [?].

Requirement 18 – Service monitoring. Anticipating that the overall infrastructure will rely on services, it should be possible to monitor these services and determine their status [Karnouskos et al. 2010]. Based on their continuous monitoring, key performance indicators can be acquired, e.g. responsiveness, reliability, performance, quality, etc.

As we see for the cooperating objects domain we deal with a wide range of requirements. In the next section we propose a reference architecture which could be used in order to define the necessary elements for enabling cooperating objects. This reference architecture has been specified taking into consideration the list of requirements described previously.

3. UICO ARCHITECTURE

As fundamental basis for a ubiquitous integration of cooperating objects, we envision an overall abstract architecture. Embedded devices are depicted at the lowest layer, i.e. the device layer. They are composed of a hardware as well as a software part that enables low level programmability. On the top layer, i.e., the application layer, we have various (enterprise) services and applications that can form mashups. Between the two, there is a middleware layer partially at infrastructure level and partially at device level.

This middleware implements functionality that assists towards the integration of and collaboration among cooperating objects. Figure 1 presents an overview of an architecture we follow within the CONET. The focus is on the middleware implemented on-device and in-infrastructure. The selection of the components was done with the requirements of Section 2 in mind, in order to sufficiently tackle them. Since we want to enable both device-only and device-to-infrastructure services, some functionality is unavoidably duplicated at infrastructure and device level; however with different degrees of support and capabilities. Examples of implementation of this architecture can be realized, for instance, by the SOCRADES Integration Architecture [Spiess et al. 2009] for the infrastructure part, while for the device specific part the BASE [Becker et al. 2003] can be used.

Applications may use the infrastructure services in order to realize sophisticated approaches and still be lightweight; however this is not a must as they can use also the devices directly. Security, trust and privacy are critical issues spawning both the application, infrastructure and device layer.

3.1 Infrastructure middleware

At the infrastructure level, the aim is to enable enterprise-level applications to interact with and consume data from a wide range of networked devices in a service-based manner. The infrastructure should enable and promote the bidirectional collaboration among its services, enterprise applications and devices in a cross-layer way. As depicted in Figure 1 several components can be identified; a detailed description

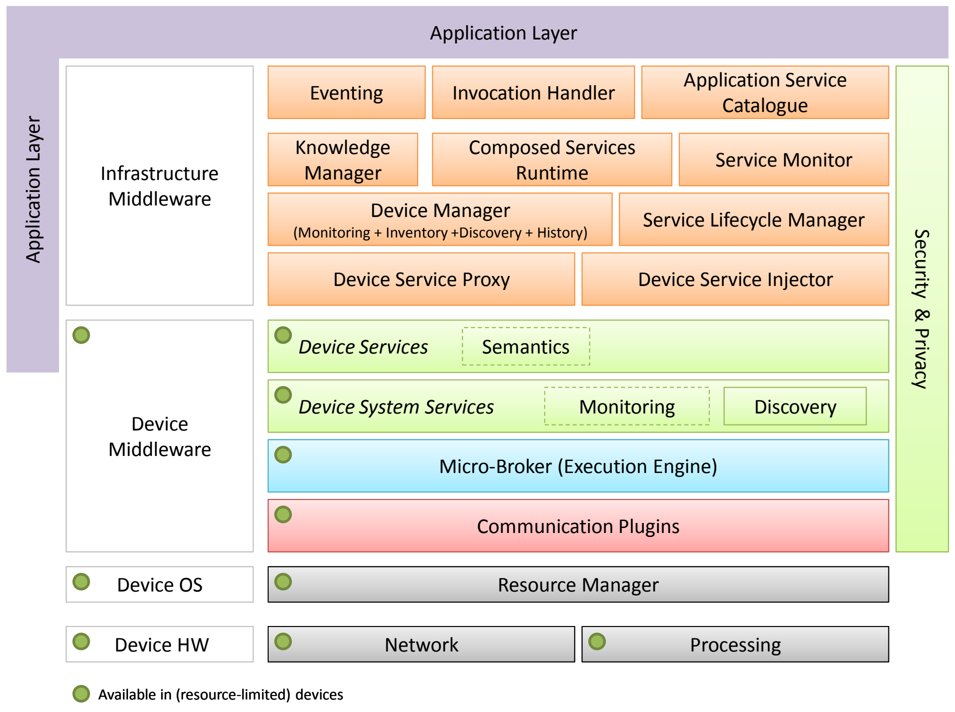

Fig. 1. UICO Reference Architecture. A minimal device middleware enables the ad-hoc cooperation of objects whose capabilities can be enhanced with an infrastructure middleware when available. This enables uniform support for a broad spectrum of application scenarios while being suitable for resource-constrained embedded systems. Security and privacy are cross-cutting concerns that must be handled at both infrastructure and device middleware levels in an adequate manner.

[Spiess et al. 2009] as well as hands-on experiences based on the implementation and usage in prototypes are available in [Karnouskos et al. 2010].

A messaging (or eventing) system allows an application to consume any events whenever it is ready to and not when a low-level service happens to send them. A so-called invocation handler allows buffering invocations to devices that are only intermittently connected. Finally, an application service catalog enables human users and applications to find service descriptions and pointers to running service instances. Both atomic services hosted by the devices and higher-level composed services can be listed here.

All functionality offered by networked devices is abstracted by services. The notion of devices is abstracted and the only visible assets are services. A repository of all currently connected service instances is provided by the service monitor. A runtime for the execution of composed services is available to support composition of business processes by offering an execution service, for example, for specified BPEL processes, meaning that service compositions can be modeled as business processes where the involved partners do not need to be explicitly specified at design time. The knowledge manager introduces the semantics and supports the collaboration among devices by semantic arguing means.

Devices are dynamically discovered, monitored and their status is available to the enterprise services. This is done via the device manager. Additionally the service lifecycle manager provides a unified view on remotely installing or updating the software (via the device service injector) that runs on devices. The device service proxy eases the integration of several categories of heterogeneous devices. Plug-ins can be written that uniformly address devices based on their capabilities, which enables the devices to communicate natively, i.e. in their own protocol.

3.2 Device middleware

At the device level, the goal is to support (enterprise) applications running in the infrastructure with an appropriate interface to the capabilities of networked and possibly mobile devices. In addition, the middleware on the devices should support the collaboration between different devices in a peer to peer fashion. This reduces the dependencies on the infrastructure in cases where the collaborative tasks are simple enough to be done by several devices. To fulfill both goals simultaneously, the architecture depicted in Figure 1 introduces four layers that are stacked on top of the device hardware and operating systems. In the following, we briefly outline their functionality and interaction from top to bottom.

At the highest layer, device manufacturers and application developers configure devices with local applications and services. The services make the device-specific functionality available to other applications that may be running locally on the device or remotely on another device or in the infrastructure. In order to be useful for arbitrary applications, the services are described not only in terms of interfaces but also with respect to their semantics. The semantic description simplifies the integration of previously unknown services by making their functionality "visible" to other services. This is important since it helps to answer two fundamental questions frequently asked when working on heterogeneous environments using services: *what can be done with the service?* and *how to interact with it?* [**?**].

Semantic descriptions are made possible by creating taxonomies of indivisible classes (or concepts), also known as ontologies. The ontologies are intended to represent knowledge pertaining to a specific domain of discourse in a rational, bounded way. There exist many languages used to define ontologies, but at the present the most commonly used are rooted on Description Logic; such is the case of the Web Ontology Language [**?**]. A class may contain one or more individuals (or instances) which inherit membership and properties of the parent class. The facts about the classes and their members are asserted by means of properties describing binary relationships.

In this sense, a service provided by a device could be described semantically by means of ontologies containing classes which define different aspects about the service: for example: type of service, types of input/output data expected/produced by the service[1], types of preconditions required before using the service, types of

[1]This differs from the interface description, which is only used as a reference during the development stage of other services and/or objects which may need to interact with the service in question. In this case, the semantic description about the data types can be used to infer if a translation procedure is required before engaging in a collaboration.

postconditions (i.e. change produced on a certain state) generated by the utilization of the service, etc. It is foreseen that by using ontologies in order to describe the semantics about the services, it is possible to infer new knowledge about them which has not been stated a priori. In other words, by *understanding* what is the *meaning* of a service, it could be possible for other entity to use it whether the service's interface definition matches 100% what was originally expected or not.

Underneath the device services, system services provide the basic functionality to support interaction, such as device discovery and service monitoring. The device discovery service announces the device and monitors the network for announcements from other devices or the infrastructure. If an infrastructure is available, the discovery service might shift the responsibility to the infrastructure in order to minimize resource usage. Via the device discovery, the service monitoring service can determine the availability of services that are needed by applications running on the device.

To abstract from different software interfaces, the device services and system services are using an execution engine provided by a so-called micro-broker. The micro-broker is responsible for forwarding service invocations to remote services and for dispatching invocations to local services. Thereby, the broker provides interfaces to register local services, decoupling the interaction pattern of the applications from the interaction pattern of the communication.

In the spirit of micro kernels, the functionality of the micro-broker can be extended with communication plug-ins that encapsulate different communication protocols and communication technologies [Handte et al. 2010] . As a result, they shield the micro-broker from any device- and technology-specific code which enables platform-independent brokers. Thus, instead of sending remote invocations directly by means of some fixed communication stack, the micro-broker can flexibly compose stacks that satisfy different application requirements. As a side effect, this also increases the interoperability and reduces the size of the on-device middleware.

4. COLLABORATION EXAMPLES

To clarify the concepts of the architecture presented in the previous section, we briefly introduce two collaboration examples. The first one describes device-only collaboration. Such collaboration examples often arise in settings where an infrastructure is not available, e.g. on the road or in a train, or where the cost of maintaining it cannot be justified. The second one describes infrastructure-assisted collaboration which is often used in settings where the cost for maintaining the infrastructure can be justified by the achievable gains. Both scenarios have already been implemented and evaluated.

4.1 Device-only Collaboration

As an example of device-only collaboration, let us consider the automation domain and more specifically the case where two modular material handling units need to interact in order to provide an adequate transportation service for a pallet carrying a product. In this example, the collaboration is carried out exclusively between the devices and there is no need for infrastructure support. Although, it could be possible to perform the collaboration between the material units using information channels provided by an underlying infrastructure, the main goal in this case is to

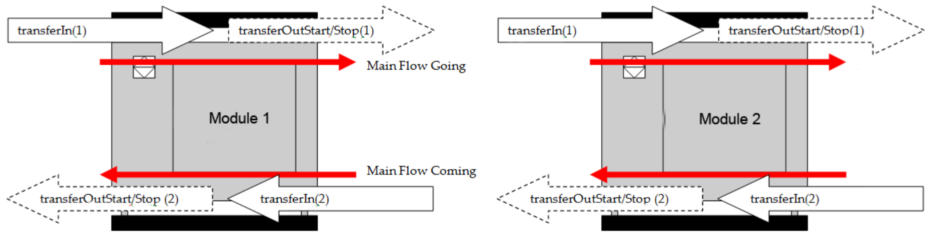

Fig. 2. Example of device-only collaboration in the automation domain. Two modular material handling units can collaborate and negotiate during runtime the transfer of a pallet thanks to a well-defined communication interface based on Web Services. This provides a new way for material handling units from different manufacturers (heterogeneous) to collaborate without a prior knowledge of each other.

provide better scalability to the system by modularizing the units, getting rid of unnecessary wiring, and reducing the complexity of the communication interfaces.

If every module is considered as a single device it will be necessary to coordinate the execution of operations provided by every device. From Figure 2, it can be seen for example that a single unit may offer services such as *transferIn*, *transferOutStart*, and *transferOutStop*. In this case, module 1, which precedes module 2, should exchange information about the pallet that is about to transfer, and should request from module 2 to start transferring the pallet in. Module 2 will inform that it will accept the pallet only if it is free and, at the same time, it will request from module 1 to start transferring out the pallet that it currently holds. Since it might be the case other modules request the same services simultaneously, every single unit provides in addition operations for reserving and releasing the execution.

4.2 Infrastructure-assisted Device Collaboration

Consider that we have also infrastructure support as we run a multi-site enterprise in which the assembly of electromechanical components could be allocated to a corresponding site. This is done as an evaluation result of the best production facility available when a production request is made, or when — due to external factors — the production should be shifted to another location.

The idea is that similar production facilities are available in remote locations. Both facilities provide electromechanical assembly capabilities, and the components of the production systems in these locations are abstracted and perceived externally as Web Services (WS). At local level, each one of the facilities acts independently and can coordinate its WS-enabled production system by using a middleware. At global level, both facilities can connect to a WS-enabled ERP module provided by a third-party, which is used for coordinating the production in the remote locations. This integration architecture is shown in Figure 3.

In addition, the devices in a location could collaborate with devices in another location, but in this case they are assisted by the infrastructure. This interaction is achieved by means of a component called Local Discovery Unit (LDU) which provides a way for connecting and managing devices and WSs from different remote locations, without needing special private network connections or proprietary protocols between the premises and the enterprise system. Through LDUs, the

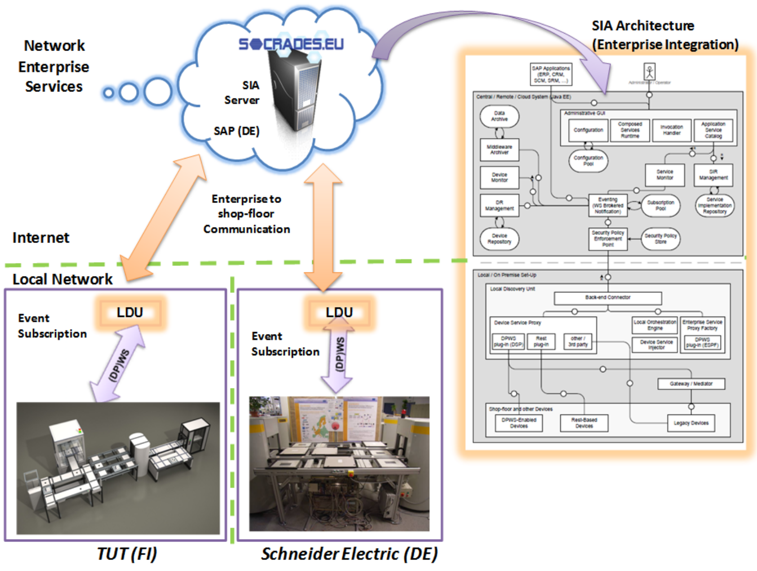

Fig. 3. Example of device collaboration assisted by the infrastructure. Devices at the same site cooperate with each other either directly or via their surrounding infrastructure. Multiple local infrastructures may cooperate to optimize the resource utilization. A local discovery unit enables the interaction between devices at different sites via an enterprise Intranet or via the Internet.

WSs from a remote location appear as if they were locally available. In this case, the infrastructure is providing the proxying mechanisms necessary to do this, and a caching support in order to accelerate the discovery of remote devices and their WSs.

Next, we discuss our experiences towards assessing the requirements outlined previously in Section 2, as well as a view on the development priorities for each one of the requirements.

5. EVALUATION AND DISCUSSION

Several approaches exist, some of which focus only in enhancing networked embedded devices with on-device software, enabling the direct device-to-device collaboration; while others focus on partially device-agnostic approaches, making the infrastructure smarter in order to extract information and feed it to the appropriate applications. Our view is that an amalgamation of both approaches might bring significant benefits to all future IoT players. Therefore, we have focused on approaches where we had hands-on experiences developing them in the last years, and attempted to investigate commonalities such as design directions, requirement coverage, implementation methods, etc.

Table I. Overview of requirements coverage by UICO middlewares

ID	UICO Requirement	Middleware Approaches 3PC	SIA	GSN	WSN-C	SE1	SE2	SOT	UICO Recommendation
1	Dynamic collaboration	◐	◐	◐	◐	◐	◐	◐	⊠
2	Extensibility	●	●	○	●	◐	○	○	⊠
3	Resource utilization	○	○	○	○	◐	○	○	⊠
4	Description of objects (interface)	●	◐	◐	●	●	●	●	⊠
5	Semantic description capabilities	○	○	◐	○	○	○	●	☑
6	Inheritance/Polymorphism	◐	◐	◐	◐	◐	◐	◐	☑
7	Composition/Orchestration	●	●	◐	○	●	◐	●	⊠
8	Pluggability	◐	●	○	●	◐	○	●	☑
9	Service discovery	●	●	◐	●	●	●	●	⊠
10	(Web) service direct devices access	◐	●	○	◐	●	●	●	☑
11	(Web) service indirect devices access (gateway)	○	●	○	○	●	○	●	☑
12	Brokered access to events	◐	●	◐	◐	◐	●	◐	☑
13	Service life-cycle management	◐	◐	○	○	◐	◐	◐	☑
14	Legacy device integration	◐	●	◐	◐	◐	●	●	⊠
15	Historian	○	◐	◐	●	○	○	○	□
16	Device management	○	◐	◐	◐	◐	○	○	☑
17	Security, trust and privacy	○	○	○	○	○	○	○	⊠
18	Service monitoring	○	●	◐	◐	◐	○	●	⊠

● Covered ⊠ Must be included
◐ Partially covered ☑ Should be included
○ Not covered □ Could be included

Table I shows an overview of the middleware systems that have been developed by the members of the CONET consortium, and a comparison based on the requirements that have been jointly derived (as described in Section 2). Thereby, we identify gaps and possible migration paths in order to provide true support for the ubiquitous integration of cooperating objects. The following middleware systems are included for comparison in Table I: Peer-to-Peer Pervasive Computing (3PC) [Becker et al. 2004]; SOCRADES Integration Architecture (SIA) [Spiess et al. 2009]; Global Sensor Network (GSN) [Aberer et al. 2006]; Wireless Sensor Network Center with Gateway Abstraction Layer (WSN-C) [Guerra et al. 2010]; Orchestration Engine with Petri Nets - Continuum (SE1) [Mendes et al. 2009] and Simulation Framework using smart devices (SE2); as well as the SOCRADES Orchestration Tools (SOT) [Puttonen et al. 2010]. Finally Table I also depicts our view on the significance of the requirements and whether these must, should, or could be present in the future cooperative IoT.

Dynamic collaboration. This functionality is partially covered by all approaches. However, we still feel it is necessary to specify until which extent an embedded device is "intelligent"; as well as to define the meaning of "collaboration" among objects and types of collaborations. In our opinion this is a "must" for any approach that will deal with a dynamic Cooperating Objects infrastructure.

Extensibility. This is partially covered mainly due to the fact that some approaches are very task or domain specific. Extensibility is a "must" and should cover not only communication protocols (the main focus of today's approaches) but be embedded on the architecture based on open standards. This implies standard interaction interfaces, and a modular structure with no hard bindings.

Resource utilization. We can see that this high-impact and important requirement is not covered by existing approaches. This is also attributable to the lack of common methods as well as the difficulty of assessing the resource impact during execution. This feature "must" be supported to enable resource-driven approaches in a resource-constrained infrastructure.

Description of objects (interface). This feature is quite well covered; however, still within specific implementations and is not open in standardized way. An intermediate mechanism for designing and deploying general types of descriptions widely understandable is still missing. We consider this a "must" feature because only implementation independent descriptions can allow the interaction between objects without the need of constant reconfiguration or re-programming.

Semantic description capabilities. There is partial support for semantics in our middlewares; however, the majority of embedded devices are not yet capable of embedding complete semantic descriptions about their capabilities. We consider this as a "should" feature as it will improve the dynamic knowledge based collaboration and enhance the cooperation capabilities of the objects.

Inheritance/polymorphism. Although partially supported, the focus is mostly on the inheritance in implementation rather than polymorphism. Cooperating Objects should inherit features from other objects, both physically and logically. We consider this as a "should" since its existence would facilitate the automatic creation of new objects and the reuse of code. In conjunction with semantics it would give us new capabilities for knowledge extraction.

Composition/orchestration. This is supported by the majority of middlewares as most of them assume service-based infrastructures where composition and orchestration are common. Dynamic workflows are not embeddable, so it is necessary to have a mechanism for representing complex workflows in embedded devices. Additionally, embedded devices do not associate automatically or engage in collaboration without a broker guiding the interactions; therefore, it is necessary to define standard ad-hoc communication patterns that two unknown embedded devices can follow in order to determine if it is possible to become associated in some manner. However, resource-constrained devices can also identify their own association capabilities by requesting support from infrastructure services. We consider this as a "must" as collaborations will be defined possibly as dynamic workflow interactions among objects.

Pluggability. Partially supported by existing approaches, we consider this as a "should" for future infrastructures. We cannot fully envision future capabilities of devices, however, it should be possible to add modules (both in software and hardware) to enhance or provide new functionality needed to realize a cooperation scenario.

Service discovery. This is one of the key requirements (therefore a "must") as devices will need to discover each other and their capabilities before initiating collaborations. As we see almost all of the existing middlewares tackle this. Focus should be on approaches that provide discovery in a global way (and not only on local networks).

(Web) service direct device access. Most of the middlewares assume direct access to the devices and their functionalities. As we mentioned, this is of benefit to specific enterprise scenarios and, therefore, it is a "should" for the future infrastructures. However, we have to point out that the target here is mostly resource-rich devices, or devices that provide very lightweight methods of accessing their itineraries such as REST.

(Web) service indirect device access (gateway). This requirement is partially supported by existing middlewares. A significant majority of devices (especially due to the miniaturization trend) is and will remain resource-constrained, incapable of accommodating direct access (or it does not make sense to realize that functionality). Gateways hiding the heterogeneity of hardware, software, communication protocols, etc. will mediate access to their features and enable easy integration. We consider this a "should" as it will allow us to integrate any kind of device to the global infrastructure.

Brokered access to events. An event-based cooperating objects infrastructure seems to be eminent in most of the middlewares. Event notifications are partly handled by the middlewares; however, the successful delivery of events is not guaranteed at the moment, unless it is explicitly stated. This is especially the case for constrained devices, which might not request any kind of acknowledgment for the events that are sent to subscribers in order to save resources. In this case, it would be necessary for such kind of devices to have access to a broker service supported by the infrastructure, which could take care of the guaranteed delivery. The broker service will be preferably distributed and/or federated. We consider this a "should" as it would enable the realization of the event-based infrastructure without depending too much on the device.

Service life-cycle management. Cooperating Objects are expected to provide a variety of services which could be modified dynamically. In this sense, some middlewares offer basic service life-cycle management capabilities which are not available on a general level, and which are application specific. We consider this as a "should" and see that it is necessary to establish a minimal set of standardized service management operations that would be available by default on all Cooperating Objects and the support infrastructure.

Legacy device integration. Today's devices are the legacy devices of tomorrow. All middlewares partially tackle this requirement. However, the main issue is that all the approaches solve the integration problem only with a reduced set of specific legacy devices known a priori. In reality, Cooperating Objects are expected to interact with a very large diversity of legacy devices currently deployed. A solution to this problem could be to provide the legacy devices with a sufficiently abstract interface that could allow them to expose their services in a more generic way. We consider this requirement a "must" as it will heavily dictate the success of Cooperating Objects, especially in long-living setups such as the industrial domain.

Historian. This is partially tackled by some middlewares, especially the ones that provide infrastructure services. Selected key data produced by the objects should be automatically logged by the infrastructure services. This would enable historic views and statistics in order to evaluate an approach. We consider this a "could" as it is of low priority in a collaboration.

Device management. Currently there are no unified methods to handle the life-cycle of heterogeneous devices especially in large-scale heterogeneous infrastructures. Although partially tackled by existing middlewares, more effort will be needed once large-scale systems become operational. We consider that this is a "should" and would enhance the collaboration capabilities.

Security, trust and privacy. It seems that security and privacy issues are underestimated, since most approaches try to "put a workable" framework. Trust is hardly investigated at all. We consider this as a "must" as in some contexts sensitive information can be interchanged between the objects, and the objects should be prepared for this. For instance, security and trust have significant importance depending on the operational environments and privacy plays a major role in user-centric environments, e.g. hospitals, homes, etc.

Service monitoring. Most middlewares do not monitor or provide very limited support for this requirement. However we consider it as a "must", especially in a service-based infrastructure where complex composable services will exist. Since cooperating objects might co-exist in a distributed fashion and might interact asynchronously, it is important to define a general event-based model that could be used to monitor any kind of service offered by the objects.

As we can see there are several considerations to be tackled for realizing a cooperating objects infrastructure and existing middleware systems do not cover all requirements. It is clear that we need to investigate more how to satisfy all requirements efficiently, what their impact would be in specific application domains, and what is the risk associated with the respective degree of requirement fulfillment.

6. CONCLUSION

The field of Cooperating Objects [Marrón et al. 2011] is a very dynamic one that has the potential of drastically changing the way people interact with the physical world as well as how business systems integrate it in their processes. We are still at the dawn of an era, where a new breed of applications and services, strongly coupled with our everyday environment will revolutionize our lives even in a deeper way than the Internet has done in these past years.

Seamless cooperation and collaboration is necessary to realize an environment where the user services are provided in a distraction free manner. Traditional models support the cooperation either by providing peer-to-peer communication between devices or by utilizing an infrastructure. We believe that combining both of these approaches provides several advantages. We have investigated based on our hands-on experiences the requirements that should be tackled in order to achieve collaboration both standalone and infrastructure-assisted mode. Based on these, we have proposed an architecture and depicted a collaboration scenario that has been implemented by using components of this architecture. Finally, we have prioritized the requirements and investigated how these are tackled in a mix of industrial and academic middlewares while in parallel providing future directions that in our view are worth following. It is important to mention that although the list of requirements described in this paper is extensive, it may not be complete. Furthermore, depending on the scenario, there exist different views on the prioritization. The main intention of this article is to set a starting point for defining the set of elements that will enable the creation of cooperating objects.

Acknowledgment

The authors would like to thank the partners of European Commission funded project CONET (`www.cooperating-objects.eu`), SOCRADES (`www.socrades.eu`) and PECES (`www.ict-peces.eu`).

REFERENCES

Aberer, K., Hauswirth, M., and Salehi, A. 2006. A middleware for fast and flexible sensor network deployment. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 1199–1202.

Aitenbichler, E., Kangasharju, J., and Mühlhäuser, M. 2007. Mundocore: A light-weight infrastructure for pervasive computing. *Pervasive Mobile Computing 3,* 4, 332–361.

Apolinarski, W., Handte, M., and Marron, P. 2010. A secure context distribution framework for peer-based pervasive systems. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on.* 505 –510.

Becker, C., Handte, M., Schiele, G., and Rothermel, K. 2004. PCOM - A Component System for Pervasive Computing. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*. IEEE Computer Society, Washington, DC, USA, 67.

Becker, C., Schiele, G., Gubbels, H., and Rothermel, K. 2003. BASE " A Micro-Broker-Based Middleware for Pervasive Computing. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*. IEEE Computer Society, Washington, DC, USA, 443.

Cheng, S.-W., Garlan, D., Schmerl, B., Sousa, J. P., Spitznagel, B., Steenkiste, P., and Hu, N. 2002. Software architecture-based adaptation for pervasive systems. In *International*

Conference on Architecture of Computing Systems (ARCS'02): Trends in Network and Pervasive Computing, H. Schmeck, T. Ungerer, and L. Wolf, Eds. Vol. 2299. Published in Lecture Notes in Computer Science, 67–82.

Edwards, W. K., Newman, M. W., Sedivy, J., Smith, T., and Izadi, S. 2002. Challenge: recombinant computing and the speakeasy approach. In *Proceedings of the 8th annual international conference on Mobile computing and networking*. MobiCom '02. ACM, New York, NY, USA, 279–286.

European Commission DG Information Society & Media. 2008. Monitoring and control: today's market, its evolution till 2020 and the impact of ICT on these. `http://www.decision.eu/smart/SMART_9Oct_v2.pdf`. Workshop presentation.

Ferscha, A., Hechinger, M., Mayrhofer, R., and Oberhauser, R. 2004. A light-weight component model for peer-to-peer applications. In *24th International Conference on Distributed Computing Systems Workshops*. Washington, DC, USA, 520–527.

Guerra, S., Fici, G. P., and Borean, C. 2010. Wireless Sensor Network Center: a ZigBee Network Management System. In *ZigBee European Developers Conference, Munich, Germany, 27-28 April 2010*.

Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., and Savio, D. 2010. Interacting with the SOA-based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services. *IEEE Transactions on Services Computing*. (accepted for publication).

Handte, M., Becker, C., and Schiele, G. 2003. Experiences - extensibility and flexibility in base. In *Workshop on System Support for Ubiquitous Computing (UbiSys) at Ubicomp 2003*. Seattle, USA.

Handte, M., Herrmann, K., Schiele, G., and Becker, C. 2007. Supporting pluggable configuration algorithms in pcom. In *Proceedings of the Workshop on Middleware Support for Pervasive Computing (PERWARE), International Conference on Pervasive Computing and Communications (PERCOM), to appear.*

Handte, M., Wagner, S., Schiele, G., Becker, C., and Marrón, P. J. 2010. The base plug-in architecture - composable communication support for pervasive systems. In *7th ACM International Conference on Pervasive Services*. Newport Beach, CA, USA.

Haroon, M., Handte, M., and Marron, P. J. 2009. Generic role assignment: A uniform middleware abstraction for configuration of pervasive systems. In *IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA, 1–6.

Herrmann, K., Rothermel, K., Kortuem, G., and Dulay, N. 2008. Adaptable pervasive flows - an emerging technology for pervasive adaptation. In *2008 Second IEEE Int. Conference on Self-Adaptive and Self-Organizing Systems Workshops*. Washington, DC, USA, 108–113.

Karnouskos, S., Savio, D., Spiess, P., Guinard, D., Trifa, V., and Baecker, O. 2010. Real World Service Interaction with Enterprise Systems in Dynamic Manufacturing Environments. In *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*, L. Benyoucef and B. Grabot, Eds. Number ISBN 978-1-84996-118-9. Springer. (in press).

Marrón, P. J., Karnouskos, S., Minder, D., and Ollero, A., Eds. 2011. *The emerging domain of Cooperating Objects*. Number ISBN: 978-3-642-16945-8. Springer.

Mazzola Paluska, J., Pham, H., Saif, U., Chau, G., Terman, C., and Ward, S. 2008. Structured decomposition of adaptive applications. *Pervasive Mobile Computing 4,* 6, 791–806.

Mendes, J. M., Bepperling, A., Pinto, J., Leitao, P., Restivo, F., and Colombo, A. W. 2009. Software Methodologies for the Engineering of Service-Oriented Industrial Automation: The Continuum Project. *Computer Software and Applications Conference, Annual International 1*, 452–459.

Object Management Group. 2004. The common object request broker: Architecture and specification, revision 3.0.3. online publication. http://www.omg.org/.

Puttonen, J., Lobov, A., Cavia Soto, M., and Martínez Lastra, J. L. 2010. A semantic web services-based approach for production systems control. *Advanced Engineering Informatics 24,* 3 (Aug.), 285–299.

Román, M. and Campbell, R. H. 2000. GAIA: Enabling active spaces. In *Proceedings of the 9th ACM SIGOPS European Workshop*. ACM, ACM Press.

Román, M., Kon, F., and Campbell, R. H. 2001. Reflective middleware: From your desk to your hand. *IEEE Distributed Systems Online Journal, Special Issue on Reflective Middleware*.

Schiele, G., Becker, C., and Rothermel, K. 2004. Energy-efficient cluster-based service discovery for Ubiquitous Computing. In *EW 11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*. ACM, New York, NY, USA, 14.

Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Souza, L. M. S. d., and Trifa, V. 2009. SOA-Based Integration of the Internet of Things in Enterprise Services. In *IEEE International Conference on Web Services, ICWS 2009 , Los Angeles, CA, USA*. 968–975.

Sun Microsystems. 2004. Java remote method invocation specification. online publication. http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf.

Sun Microsystems. 2006. Jini technology surrogate architecture specification, v1.0. online publication. http://surrogate.dev.java.net/specs.html.

UPnP Forum. 2008. Universal plug and play device architecture, version 1.0, document revision date 24 april 2008. online publication. http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20080424.pdf.