Requirement Considerations for Ubiquitous Integration of Cooperating Objects

Stamatis Karnouskos*, Vladimir Villaseñor-Herrera†, Muhammad Haroon‡, Marcus Handte‡, Pedro José Marrón‡
*SAP Research, Germany. Email: stamatis.karnouskos@sap.com
†Tampere University of Technology, Finland. Email: vladimir.villasenorherrera@tut.fi
‡Universität Duisburg-Essen, Germany. Email: {muhammad.haroon,marcus.handte,pjmarron}@uni-due.de

***Abstract*—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. Finally we prioritize the requirements, and discuss on future directions that could be followed.**

I. Cooperating Objects

The core idea behind amalgamating the physical and virtual (business) world is to seamlessly gather any 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 should offer its functionality as one or more services, while in parallel it is possible to discover and invoke new functionality from other services on-demand. 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.

According to CONET [1], 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. 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 (object-to-object and object-to-business) will be able to engage into a conversation in a loosely-coupled manner.
- They will have a common understanding of well-defined communication patterns and protocols.
- They will be able to interchange data relevant to their capabilities and needs.
- They will share computational resources when needed by means of information migration or data mashups.
- They will be able to cluster in order to create distributed data gathering/processing platforms.

Another interesting trend is the evolution towards global service-based infrastructures. 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 domain of Cooperating Objects is still at its dawn; however, its impact is estimated to be so broad and significant that could change drastically the future applications and services. 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.

II. 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: (i) standalone, where devices discover and interact with each other on a standalone mode without significantly depending on the existence of third-parties, and (ii) infrastructure assisted, where devices cooperate with each other and third-parties by heavily depending on infrastructure services. Based on our experiences coming from multiple industry and R&D projects (an overview is depicted in Table I), we see several requirements for the Ubiquitous Integration of Cooperating Objects (UICO).

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.

Req. 1 – Dynamic collaboration: Devices with embedded intelligence and sensing/actuating capabilities 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).

Req. 2 – Extensibility: Flexible support for extending the capabilities of a device is needed. Cooperating Objects is a rapidly developing domain and implementation should take into consideration future growth. Since extensions can be made through the addition of new functionality or modification of existing one, change support should be there while minimizing impact to existing system functions.

Req. 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 the devices with more resources, and opportunistically take advantage of the resources in the surroundings if it makes sense from the strategy/performance viewpoint.

Req. 4 – Description of objects (interface): Implementation independent description of the object that can be used by both implementer and requester is needed. This will enable 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.

Req. 5 – Semantic description capabilities: Semantics and ontologies should be used to enforce the dynamic interpretation of things and used as 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. as well as describe constraints of capabilities of the specific cooperation.

Req. 6 – Inheritance/polymorphism: It would make sense to have a way to form new objects using objects that have already been defined. This will enable easy programming as code can be reused. 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.

Req. 7 – Composition/orchestration: Generation and execution of work-plans between objects, services and other resources in order to promote their interaction should be supported.

Req. 8 – Pluggability: 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.

Req. 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.

Req. 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. 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.

Req. 11 – (Web) service indirect device access (gateway): Gateways might glue to the Cooperating Objects infrastructure devices by hiding heterogeneity and resource scarceness. Further more, most efforts in the research domain today focus on how to open the device functionality to the enterprise systems; however, the opposite i.e. open enterprise systems to the devices might also be beneficial. 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.

Req. 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).

Req. 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.

Req. 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 to fit in the new service-enabled infrastructure. The latter is expected to be achieved via the wrapping of them, e.g. with Web Services.

Req. 15 – Historian: In an information-rich infrastructure, logging of 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.

Req. 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 there is a way to hide heterogeneity and provide uniform access to device's and infrastructure's life cycle.

Req. 17 – Security and privacy: Security 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. Similarly, the privacy should be preserved especially for devices operating in sensitive user areas, e.g. hospitals, households, etc.

Req. 18 – Service monitoring: As we anticipate a service based infrastructure, it should be possible to monitor the services and determine their status. Based on the continuous monitoring, key performance indicators can be acquired, e.g. responsiveness, reliability, performance, quality, etc.

III. 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 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 defined (as described in section II). 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) [2]; SOCRADES Integration Architecture (SIA) [3]; Global Sensor Network (GSN) [4]; Wireless Sensor Network Center with Gateway Abstraction Layer (WSN-C) [5]; Orchestration Engine with Petri Nets - Continuum (SE1) [6] and Simulation Framework using smart devices (SE2); as well as the SOCRADES Orchestration Tools (SOT) [7]. 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: The 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 the 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

Table I
OVERVIEW OF REQUIREMENTS COVERAGE BY UICO MIDDLEWARES

ID	UICO Requirement	Middleware Approaches							UICO Recommendation
		3PC	SIA	GSN	WSN-C	SE1	SE2	SOT	
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 and privacy	○	○	○	○	○	○	○	⊠
18	Service monitoring	○	●	◐	◐	◐	○	●	⊠

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

consider this as a "should" since its existence would facilitate the automatic creation of new objects and the code reuse. 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 can not 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 acknowledgement 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 and privacy: It seems that security and privacy issues are underestimated, since most approaches try to "put a workable" framework. 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 has a significant importance in non-isolated environments and privacy plays a key 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. It is clear that we need to investigate more how to satisfy these requirements efficiently, what their impact would be in specific application domains, and what is the risk associated with the respective degree of requirement fulfillment.

IV. Conclusion

It is clear that the field of Cooperating Objects [1], 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 modus.

Acknowledgment

The authors would like to thank the partners of European Commission funded project Cooperating Objects Network of Excellence (CONET - www.cooperating-objects.eu). Especially we would like to thank for the fruitful discussions: Fabio Bellifemine, Sami Bhiri, Armando Walter Colombo, Umer Iqbal, José Martínez Lastra, Paulo Leitao, Filipe Pacheco, Nataliya Popova, Paulo Gandra de Sousa, and Enrico Vinciarelli.

References

[1] P. J. Marrón, S. Karnouskos, and D. Minder, Eds., *Research Roadmap on Cooperating Objects*. European Commission, Office for Official Publications of the European Communities, July 2009, no. ISBN: 978-92-79-12046-6.

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

[3] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. d. Souza, and V. Trifa, "SOA-Based Integration of the Internet of Things in Enterprise Services," in *IEEE International Conference on Web Services, ICWS 2009 , Los Angeles, CA, USA*, Jul. 6–10, 2009, pp. 968–975.

[4] K. Aberer, M. Hauswirth, and A. Salehi, "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, 2006, pp. 1199–1202.

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

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

[7] J. Puttonen, A. Lobov, M. Cavia Soto, and J. L. Martínez Lastra, "A semantic web services-based approach for production systems control," *Advanced Engineering Informatics*, vol. 24, no. 3, pp. 285–299, Aug. 2010.