Wise Objects for Calm Technology

Ilham Alloui, David Esale and Flavien Vernier
LISTIC lab., University Savoie Mont Blanc, 5 chemin de Bellevue, Annecy-le-Vieux, France

Keywords: Introspection, Intelligent Systems, Adaptive Systems, Autonomous Learning, Decentralized Control.

Abstract: In this position paper we identify the design of "wise systems" as an open research problem addressing new technology-based systems. Increasing complexity and sophistication make those systems hard to understand and to master. Human users are very often involved in learning processes that capture all their attention while being of little interest for them. To alleviate human interaction with such systems, as the foundation of our current research, we propose the concept of "wise object" as the building block. Software-based systems would then be able to autonomously learn on themselves and on the way humans use them. Humans would in turn be prompted only when necessary by the system.

1 INTRODUCTION

New technologies are usually designed for meeting some social/business/political needs or goals. Among notable new technologies we find Communicating Objects (COT) and Internet of Objects (IOT) that increasingly contribute to our daily life (mobile phones, computers, home automation, etc.). Systems based on those technologies become very sophisticated, even to experienced users. For instance, people at home usually face at least two problems with home automation systems: (1) instructions accompanying the devices are too complex and it is hard for non-expert users to master the whole behaviour and capabilities provided by the system; (2) such systems are usually designed to meet general requirements through a set of predefined configurations. Information needed by a user is not necessarily the same from one to another. A user may need a set of services in a given context and a different set of services in another context. A user does not need to use all what a system could provide in terms of information or services.

In this position paper, we claim that a system based on new technologies must be able to: (1) "know by itself on itself", i.e. to learn how it behaves, to consequently reduce the understanding effort needed by users (even experimented ones); (2) "know by itself on its usage" to adapt to users according to the way and to the context it is used in. In addition like any service-based system (3) such system should be capable of improving the quality of services it is offering.

We need "non-intrusive" systems that serve users while requiring "just some" (and not all) of their attention and only when necessary. This in a sense contributes to "calm technology" (Weiser and Brown, 1996) that "describes a state of technological maturity where a user's primary task is not computing, but being human". As claimed in (Case, 2010), new technologies might become highly "interruptive" in human's daily life. Though "Calm technology" has been proposed first by Weiser and Brown in early 90's (Weiser and Brown, 1996), it is more than ever, a challenging issue in technology design.

We need systems composed of "autonomous" entities that are able to independently adapt to a changing context.

Many approaches are proposed to design and develop the kind of systems we target: multi-agent systems (Wooldridge, 2009), intelligent systems (Roventa and Spircu, 2009), adaptive systems (Salehie and Tahvildari, 2009), self-X systems (Huebscher and Mccann, 2008). In all those approaches, a system entity (or agent) is able to learn on its environment (including the other entities) through its interactions. Our intention is to go a step forward by enhancing a system entity with the capability of learning by its own on the way it has been designed to behave in. We see at least two benefits to this: (a) a decentralized control: as each entity evolves independently from the others, it can control actions to perform at its level according to the current situation; (b) each entity can improve its performance and then the performance of the whole system.

Our work addresses those issues through the concept of "Wise Objects". We call "wise object", a software-based entity that is able to learn on itself and also on the others (e.g. its environment and users). "Wisdom" refers to the experience such object acquires by its own during its life. We intentionally use terms dedicated to human as a metaphor. When human better succeed in observing the others, a wise object would have more facility to observe itself by introspection. A wise object is for instance a vacuum cleaner that could learn how to clean a room depending on its shape and dimensions. In the course of time, the object would in addition improve its performance (less time, less energy consumption, etc.).

In section 2, through an illustrating example on home automation, we briefly present our approach, system requirements and design principles.

2 RESEARCH ISSUES

2.1 Requirements

To meet users' requirements cited so far, namely: (1) the ability of a system to reduce the effort needed by its users to understand system behaviour; (2) the capability of a system to adapt to its users according to the way and the context it is used in; (3) improving the quality of services it is offering; we adopt an approach founded on the concept of "Wise objects" (WO). Wise objects refer to objects that have the ability to learn on their behaviour and also on the behaviour of their users according to changing context. In this paper we use the following definition from (Dey and Abowd, 2000): "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*" This definition is generic enough to apply to software-based entities (implemented through class objects that represent the *"low level"* part of context).

To illustrate our purposes, we use a simple example in home automation domain. Let us consider a system composed of a roller shutter (actuator) and a control key composed of two buttons (sensors). In the very general case and in a manual mode, with a one-button control key, a person uses the button to: bring the shutter either to a higher or to a lower position. With a second button, the user can tune inclination of the shutter blades to get more or less light from the outside. As the two buttons cannot be activated at the same time, the user must proceed in two times: first, obtain the desired height (e.g. 70%) then the desired inclination (e.g. 45%). For such systems, three roles are generally defined: "System developer", "System configurator" and "End-user". Assume an end-user is at his office and that according to the moment and to the weather, his/her requirements for the shutter change (height and inclination). This involves the end-user all along the day.

Our idea is that sort of system could be designed to alleviate its interactions with the end-user. In our example, the "wise" system would use some knowledge from past experiences to change the shutter height and inclination when needed. Moreover, before the first use of the system by end-users, the "wise" system could propose to the "system configurator" a first "picture" of the behaviour of system components. Such picture is the result of an introspection process done by each component of the system (i.e. control key and shutter). Each component, i.e. "wise object", has the ability to learn on its behaviour. The system configurator could then complete and/or correct information provided by the "wise" system so that the home automation system could perform. S/he in particular defines "valid" coordination rules among system components; for example, a *switch on* action on the control key must be followed by a *raise* action on the shutter.

The design of "wise" systems raises many open research issues, among them:

- How to design such systems with the minimum of "intrusion" in the way home automation systems are usually developed?
- How could individual components learn on their behaviour?
- How to put together knowledge coming from autonomous components?
- How could the automation system learn about the way it is used?

In the following section, we give an overview of the approach we are working on.

2.2 Approach

Our approach is based on the concept of "wise object" as the building block for "wise" systems. We address open issues cited above as follows:

- To design "non intrusive" systems, both for users (with different roles: system developers, system configurators and end-users), we propose a framework of "wise objects" from which a system inherits its "wisdom";
- Each system entity inheriting from Wise Object (WO) class will have the ability to learn on itself and on its usage by others.
- In the system a particular object called *Assembly Object* is in charge of putting together individual WOs behaviours. A WO instance does not know the other WO instances in the decentralized system.

As already said, a Wise Object (WO) is an object that knows itself by its own, i.e. its knowledge is not obtained from an external database. This acquisition is performed by *introspection* and *monitoring*.

As depicted by Figure 1, a WO life-cycle involves two main steps: *Configuration* and *Operation*. When an instance of WO Class is created the object has no knowledge about the services it is expected to provide.

At *Configuration* step, a WO acquires knowledge about its capabilities (i.e. services implemented as methods) thanks to introspection mechanisms we defined in WO class. Thus, a WO object discovers services it is intended to offer and constructs a behaviour graph of all its possible states and all its possible transitions when it invokes those services. Transitions in the behaviour graph correspond to the object method invocations. A WO object can easily obtain the set of its methods by *introspection*. A state in the graph behaviour is defined by the attribute values of the object. When a WO instance is created, the object is in its initial state. The other states are computed by method invocation. Each invocation can move the object into a new state or let the object into its current state. When all methods are invoked on all known states, the behaviour graph is considered as complete. What is worth noting here is that in "Learning on itself" sub state, a WO is able to act in an autonomous way, i.e. with no external interaction. This results a behaviour graph that could be either incomplete (e.g. because it requires external information) or not valid (e.g. because some transitions are not realistic). A "validation" sub state involving users is required for those reasons. This sub state is in particular necessary for assembling behaviour graphs of system WO instances. Indeed up to now, a WO instance has learnt only about its behaviour.

In addition to WO objects, we designed an Assembly Object that puts together graph behaviours of participating WO instances. An Assembly Object assembles behaviour graphs in a way similar to process composition in FSP (Finite State Processes) algebra (Magee and Kramer, 2006). In a system at work, each service invocation is followed by the requested service delivery (i.e. executing the corresponding object method). We then can view object method execution as an atomic action, and, coordination among concurrent WO instances as a composition of their behaviour graphs from a process perspective (i.e. ordering constraints on object method execution). It is in the charge of the system configurator to define the valid "assembly" or coordination rules. In our illustrating example, System configurator defines the following rule: a *switch on* must be followed by *shutter roll down*. According to this rule, the Assembly Object deactivates all transitions that do not conform to the expected rules.

When the *Operation* step starts, a WO instance is ready to learn about its usage. It collects data (*Collecting usage data*) each time a service is invoked. Those data correspond to the statistics on state changes or the discrete-time Markov chain of the usage. As the behaviour graph is known (*Configuration* step), the Markov chain is computed by monitoring method invocations on the object. This computation is done by the WO instance when it is in *idle*, i.e. it is not executing a service (*Learning on its usage*). In this step, when an uncommon case occurs (e.g. a service that has never been invoked by a user before), the WO instance handles this situation in the *Managing emotion* sub state. The word "emotion" is another metaphor to qualify unusual situations.

Up to now, we considered atomic objects (i.e. not composed of other objects). One more important issue is then: in a hierarchical system of WOs (i.e. a WO composed of WOs), how can knowledge from low-level WOs be managed by high-level WOs? The amount of knowledge can be important but not always relevant to high-level WOs, in particular if this does not bring new information. Thus, it is more relevant for the system to translate knowledge from low to high-level WO only if knowledge evolves or if the usage of WO changes. If we consider that the capabilities of a WO cannot change, two questions are raised:

- how can a WO detect a change on its usage?
- is this change relevant to the high-level WO?

We see the former as a fuzzy problem where the change can be expressed as a distance to a common usage reference. Regarding the second question, we consider that a low-level WO cannot "say" if a change on its usage has an impact on its high-level

WO. Only a high-level WO can define if a change in its low-level WOs affects it. Thus, when a usage-related change appears, a WO must send information to its high-level WOs. These changes can be of a different nature: change on the frequency of usage (objects are more or less frequently used), change on the used capabilities... We refer to this nature of changes as *emotion*. A WO is *stressed* if its use is more frequent than its common use. A WO is *surprised* if a capability is uncommonly used. This approach raises a new question: how emotions can be merged into high-level WO? This last problem requires an information fusion solution.

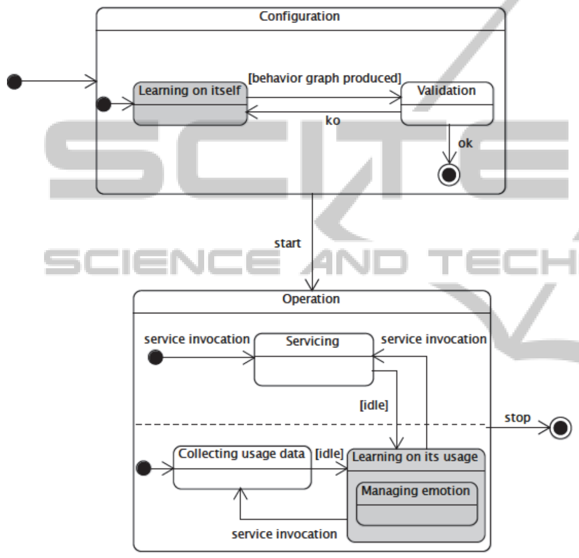

Figure 1: Wise Object behaviour.

WO instance gets out from this sub state each time a service is invoked, and, it returns into it each time the WO instance is idle. It is worth noting that the *service invocation* event and the *idle* state are two synchronisation points among the concurrent states of *Operation*. We have separated the "wise" part of a WO instance from its "common usage" part. We consider that this is essential to meet "non intrusiveness" requirement. Another design issue is that we have highlighted states where a WO instance needs introspection (grey coloured states in Figure 1). We use the metaphor "dream" for those states to distinguish them from "real" states (white states in Figure 1) where the WO instance is delivering requested services. An important issue is that when the object dreams, it cannot affect its environment. Thus, a WO must manage its interactions with the other objects. One of the best ways, in our point of view, to manage these interactions is to use an event bus. A WO instance can then activate or not the events according to its state.

3 CONCLUDING REMARKS

Our current research addresses the problem of how to design autonomous systems that limit the involvement of their users to what is necessary. We propose the concept of "wise object" as the building block of such systems. As proof of concept, we are currently developing a Java framework for implementing this kind of systems with the minimum intrusion in the application code. Object classes produced by a developer inherit the behaviour of "Wise object" (WO) class. An instantiated system is then a "wise system" composed of "wise objects" that interact through an event bus according to "publish-subscribe" design pattern. We believe that "wise systems" is a promising approach to help humans serenely integrate new technologies in their daily life.

REFERENCES

Dey, A. K., Abowd, G. D., 2000. Towards a Better Understanding of Context and Context-Awareness. In *CHI 2000. Workshop on the What, Who, Where, When, and How of Context-Awareness*.

Weiser, M., Brown, J. S., 1996. Designing Calm Technology. In *PowerGrid Journal, v 1.01* (http://powergrid.electriciti.com/1.01).

Case, A., 2010. We Are All Cyborgs Now. In *TED: Ideas Worth Spreading* (http://www.ted.com/talks/amber_case_we_are_all_cyborgs_now.html).

Wooldridge, M., 2009. *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2nd edition.

Roventa, E., Spircu, T., 2009. Management of Knowledge Imperfection in Building Intelligent Systems. In *Studies in Fuzziness and Soft Computing*, Springer, v 227.

Salehie, M., Tahvildari, L., 2009. Self-adaptive software: Landscape and research challenges. In *ACM Transactions on Autonomous and Adaptive Systems (TAAS),* v 4.2, No. 14.

Huebscher, M. C., Mccann, J. A., 2008. A survey of Autonomic Computing – degrees, models and applications. In *ACM Computing Surveys (CSUR)*, v40.3, No. 7.

Magee, J., Kramer, J., 2006. *Concurrency: state models and java programs*. Wiley, 2nd edition.