Implicit Interaction

M.J O'Grady, J. Ye, G.M.P. O'Hare, S. Dobson, R. Tynan, R. Collier, C. Muldoon

CLARITY: Centre for Sensor Web Technologies,
School of Computer Science & Informatics,
University College Dublin,
Belfield, Dublin 4, Ireland.
{michael.j.ogrady, juan.ye, gregory.ohare, simon.dobson, richard.tynan, rem.collier,
conor.muldoon, }@ucd.ie

Abstract. Achieving intuitive and seamless interaction with computational artifacts remains a cherished objective for HCI professionals. Many have a vested interest in the achievement of this objective as usability remains a formidable barrier to the acceptance of technology in many domains and by various groups within the general population. Indeed, the potential of computing in its diverse manifestations will not be realized fully until such time as communication between humans and computational objects can occur transparently and instinctively in all instances. One step towards achieving this is to harness the various cues that people normally use when communicating as such cues augment and enrich the communication act. Implicit interaction offers a model by which this may be understood and realized.

Keywords: *Keywords: Implicit Interaction, interaction modalities, Ambient intelligence, embedded agents, context-aware computing*

1. Introduction

It is acknowledged that intuitive interaction is fundamental to the success of computing services. How such interaction is achieved is open to question. And the need for an answer to this question is increasingly urgent, given the paradigm shift that is ongoing towards pervasive computing. In the original manifesto for ubiquitous computing in the early 1990s, the need for seamless and intuitive interaction was explicitly acknowledged. How it was to be achieved was not stated. A decade later, the Ambient Intelligence initiative proposed that Intelligent User Interfaces (IUIs) would solve this problem. Another decade has passed and the question remains open.

In this paper, it is proposed that a holistic view of interaction be adopted, encompassing its explicit and implicit components. Realizing this in practice is computationally complex; nevertheless, developments in sensor and related technologies are enabling hardware and software platforms from which this vision of interaction may be attained in practice.

1.2 How People Interact

If Intelligent User Interfaces (IUIs) [1] that truly enables intuitive instinctive interaction are to be developed, an innate understanding of how people communicate is essential. It is useful to reflect on this briefly. Humans communicate using a variety of means - verbal being a prominent communication modality. Yet nonverbal cues (behavioral signals), for example frowning, have over four times the effect of verbal cues [2]. For interfaces to act intelligently and instinctively, non-verbal cues need to be incorporated into their design and implementation. One well known classification of non-verbal behavior is that of Ekman & Friesen [3] who have listed 5 categories:

- Emblems: actions that carry meaning of and in themselves, for example a thumbs up.
- Illustrators: actions that help listeners better interpret what is being said, for example, for example, finger pointing;
- Regulators: actions that help guide communication, for example head nods;
- Adaptors: actions that are rarely intended to communicate but that give a good indication of physiological and psychological state;
- Affect: actions that express emotion without the use of touch, for example, sadness, joy and so on.

Thought this categorization has proved popular, it does not capture all kinesic behaviors, eye behavior being one key omission. Nevertheless, the model does give some inkling as to the complexity of the problem that must be addressed if instinctive interaction between humans and machines is to be achieved.

Capturing all kinesic behavior is desirable as studies indicate that human judgments were more accurate when based on a combination of face, body, and speech than when just using face and body [4] though the contribution of each may be dependent on the prevailing context. In practice, it may not be feasible in all circumstances to capture all cues; thus it may be necessary to work with a subset of the available behavioral cues. This need not be an insurmountable problem as some cues may be more important than others. Certainly the visual channel, through the reading of facial expressions and body postures, seems to be the most important as the face represents the dominant means of demonstrating and interpreting affective state [5]. Though speech is essential for communication in normal circumstances, interpreting the affective state from both its linguistics and paralinguistic elements is inherently difficult and much work remains to be completed before the emotional significance of these elements can be extracted and identified with confidence [6].

Finally, the issue of physiological cues need to be considered briefly as these almost invariably indicate an affective state. Examples of such cues might include increased heart rate, temperature and so on. Unless having undergone specific training, most people would not be capable of sensing such cues. Wearable computing offers options for harvesting such cues as garments embedded with sensors for monitoring heart and respiratory rate amongst others are coming to market. However, whether people will wear such garments in the course of their normal everyday activities is open to question, as is the issue of whether they would share this data with a third-party.

2. Interaction Modalities

As has been described, human communication encompasses many modalities and the depth of complexity that requires significant research is still required if the underlying complexity is to be resolved. This poses significant challenges for those who aspire to develop computing systems that can successfully capture and interpret the various cues and subtleties inherent in human communications. As a first step towards this, many researchers have focused on multimodal human computer interaction and see it as a promising approach to facilitating a more sophisticated interaction experience.

2.1 Multimodal Interaction

Multimodal interaction in a HCI context harnesses a number of different communication channels or modalities to enable interaction with a computer, either in terms of input or output. The key idea motivating multimodal interaction is that it would appear to be a more natural method of communications as it potentially allows the capturing and interpretation of a more complete interaction context, rather than just one aspect of it. A key question then arises – does research support or contradict this motivation?

One well cited study by Oviatt [7], has systematically evaluated multimodal interfaces. It was shown that multimodal interfaces speeded up task completion by 10%, and that users made 36% fewer errors than with a unimodal interface. Furthermore, over 95% of subjects declared a preference for multimodal interaction. Though these results are impressive, it must be remembered this study focused on interaction in one domain, namely that of map based systems. Whether these results can be generalized to other domains and different combinations of modalities remains to be seen. To gain a deeper understanding of this issue, it is useful to reflect on how multimodal interaction is defined.

Sebe [8] defines modality as being "a mode of communication according to the human senses and computer input devices activated by humans and measuring human qualities". For each of the five human senses, a computer equivalent can be found. Microphones (hearing) and cameras (sight) are well established. However, haptics (touch) [9], olfactory (smell) [10] and taste [11] may also be harnessed, albeit usually in specific domains and circumstances. For a system to be considered multimodal, these channels must be combined. For example, a system that recognizes emotion and gestures using one or multiple cameras would not be multimodal, according to this definition, whereas one that used a mouse and keyboard would be. This definition is strongly influenced by the word "input". A less rigorous interpretation might be to consider not only modality combinations but also select attributes of individual modalities. In either case, the problem facing the software engineer is identical. Individual input modalities must be parsed and interpreted, and then the final meaning of the interaction estimated though a fusion process. How this may be achieved is beyond the scope of this discussion; however one technique proposed involves the use of weighted finite state devices, an approach that is lightweight from a computational perspective and is thus suitable for deployment of a range of mobile and embedded devices [12]. Ultimately, the question that is being posed for an arbitrary system is

what is it that the user intends (Fig. 1). While this may never be known with 100% certainty, the harnessing of select cues that are invariably used in communication may contribute to more complete understanding of the user's intent thereby leading to a more satisfactory interactive experience.

Fig. 1. At any time, a user may display a range of non-verbal cues that give a deeper meaning to an arbitrary interaction. Identifying and interpreting such cues is computationally complex but a prerequisite to instinctive interaction.

2.2 Interaction as Intent

In most encounters with computing, interaction is explicit – an action is undertaken with the expectation of a certain response. In computational parlance, it is event driven. The reset button is pressed and the workstation reboots. This is the default interaction modality that everyone is familiar with, even in non-computing scenarios. When designing interfaces, a set of widgets is available that operates on this principle. No other issue is considered. The application is indifferent to emotions and other contextual parameters. Should the context be available, a number of options open up but the appropriate course of action may not be obvious in all circumstances. If it is determined that the user is stressed for example, is the designer justified in restricting what they can do? Should certain functionality be temporarily suspended while certain emotions are dominant? If applications are to act instinctively, the answer is probably yes; thus embedded applications will have to support multimodal I/O.

Interaction may also be implicit, and it is here that non-verbal cues may be found. All people communicate implicitly. The tone of peoples' voices, the arched eyebrow and other facial expressions reinforce what they say verbally. Intriguingly, it can also contradict it. Though people can seek to deceive with words, gestures can indicate when they do so. Thus if we seek interactions that are based on truth, an outstanding challenge is to harness and interpret implicit interaction cues. This is computationally complex, requiring that such cues be captured, interpreted and reconciled in parallel with explicit interaction events.

One subtle point with implicit interaction is that it can, in certain circumstances, represent the direct opposite of explicit interaction. In short, what is NOT done, as

opposed to what is done, may indicate a choice or preference. For example, in ignoring an available option, users may be saying something important about their preferences. What this means is of course domain and context dependent.

In all but the simplest cases, implicit interaction is multimodal. It may require the parallel capture of distinct modalities, for example, audio and gesture. Or it may require that one modality be captured but be interpreted from a number of perspectives. For example in the case of the audio modality, semantic meaning and emotional characteristics be may extracted in effort to develop a deeper meaning of the interaction.

2.3 Models of Interaction

Various models of interaction have been proposed in computational contexts, for example, those of Norman [13] and Beale [14]. Ultimately, all frameworks coalesce around the notions of input and output, though the humans and computer interpretation of each is not symmetrical. Obreovic and Starcevic [15] define input modalities as being either stream-based or event based. In the later case, discrete events are produced in direct response to user actions, for example, clicking a mouse. In the former case, a time-stamped array of values is produced. In the case of output modalities, these are classified as either static or dynamic according to the data presented to the users. Static responses would usually be presented in modal dialog boxes. Dynamic output may present as an animation - something that must be interpreted only after a time interval has elapsed.

Fig. 2: Interaction may be regarded as constituting both an implicit and explicit component.

For the purposes of this discussion, interaction is considered as composing a spectrum that incorporates an implicit and explicit component (Fig. 2) but in which one dominates. An explicit interaction may be reinforced or augmented by an implicit

one, for example, smiling while selecting a menu option. Likewise, an implicit interaction may be supported by explicit one, an extreme case being somebody acting under duress where they are doing something but their body language clearly states that they would rather not be pursuing that course of action. Thus the difficulty from a computational perspective is to identify the dominant and subordinate elements of an interaction, and to ascribe semantic meaning to them. A key determinant of this is the context in which the interaction occurs.

3. Reasoning with Context

If the context in which the user operates is fully understood, a successful interaction can take place. In practice, an incomplete state of the prevailing context is all that can be realistically expected in all but the simplest scenarios. Indeed, it is questionable as to whether it is possible to articulate all possible contextual elements for an arbitrary application [16]. Usually, software engineers will consider simpler forms of context, normally those that are easy to capture and interpret, and incorporate them into their designs. Though useful, these are just proxies for user intent [17] and are used in an effort to remedy the deficiency in understanding of what it is that the user is try to achieve. This deficiency obliges the software engineer to use incomplete models to best estimate the prevailing context, and to adapt system behavior accordingly.

3.1 Context Reasoning

Not every piece of information about a user has an equal effect on the interaction. Low-level context can be enormous, trivial, vulnerable to small changes, and noisy. Therefore, higher-level contexts (or situations) are needed to derive from an amount of the low-level context, which will be more accurate, human-understandable, and interesting to applications.

Earlier research on context attempted to use first-order logic to write reasoning rules, for example the work by Gu et al. [18], Henricksen et al. [19], and Chen et al. [20]. More recently, ontological reasoning mechanisms have been adopted as the technology of choice to make reasoning more powerful, expressive, and precise [21, 22]. Currently, research focuses more on formalizing situation abstraction in terms of logic programming. Loke presents a declarative approach to representing and reasoning with situations at a high level of abstraction [23]. A situation is characterized by imposing constraints on the output or readings returned by sensors (Fig. 3). A situation occurs, when the constraints imposed on this situation are satisfied by the values returned by sensors. For example, an “in_meeting_now” situation occurs when a person is located with more than two persons and there is an entry for meeting in a diary. These constraints are represented as a logic program. This approach is based on the logical programming language LogicCAP that embeds situation programs in Prolog, and provides a high level of programming and reasoning situation for the developers. The logical theory makes it amenable to formal analysis, and decouples the inference procedures of reasoning about context and situations from the acquisition procedure

of sensor readings. This modularity and separation of concerns facilitates the development of context-aware systems.

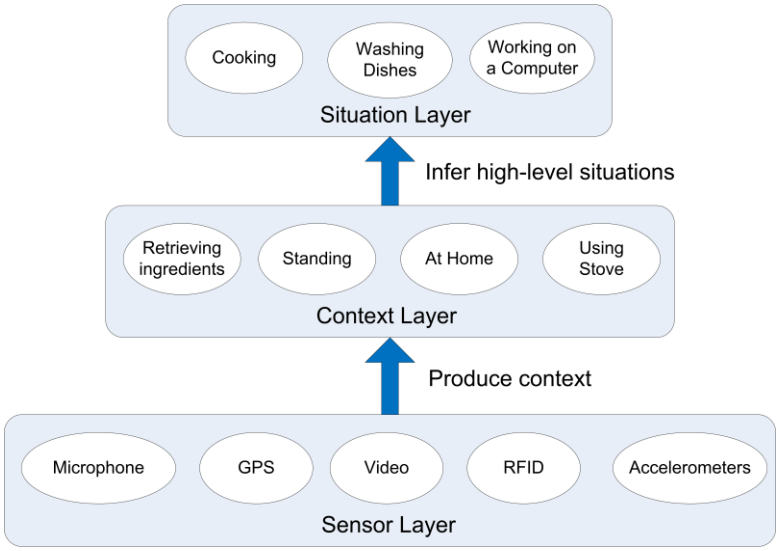

Fig. 3: Situations can be inferred from individual contexts harnessed from a suite of sensors.

3.2 Context Uncertainty

In terms of software, the error-prone nature of context and contextual reasoning alter the ways in which we must think about *interaction* and *adaption*. If a context is incorrectly reported, or is considered irrelevant to users, a problem will occur when a system makes a responsive action adapting to real-time contextual changes [24]. Resolving uncertainty in context has been a hot research topic in recent years. Henricksen et al. [25] refine the quality of context into five categories:

1. *incompleteness* – if a context is missing;
2. *imprecision* – if the granularity of a context is too coarsed;
3. *conflicting* – if a context is inconsistent with another context;
4. *incorrectness* – if a context contradicts with the real world state;
5. *out-of-dateness* – if a context is not updated in response to changes.

Any decision may be made incorrectly on account of any type of poor input data quality. Beyond the quality in context, oversimplified reasoning mechanisms could introduce extra noise to inferred results.

Anagnostopoulos et al. [26] define a fuzzy function to evaluate the degree of membership in a situational involvement that refers to the degree of belief that a user is involved in a predicted situation. They define Fuzzy Inference Rules (FIR) that are used to deal with imprecise knowledge about situational context and the user

behaviour/reaction and historical context. Similarly Ye et al. [27] use the fuzzy function to integrate and abstract mulitple low-level contexts into high-level situations. The fuzzy function is used to evaluate how much the current context satisfies the constraints in a situation's specification.

Machine learning techniques are widely applied to deal with uncertainty issues in the inferring process. Bayesian networks have a causal semantics that encode the strength of causal relationships with probabilities between lower- and higher-level. Bayesian networks have been applied by Ranganathan et al. [28], Gu et al. [18], Ding et al. [29], Truong et al. [30], Dargie et al. [31], and Ye et al. [32]. For example, Gu et al. encoded probabilistic information in ontologies, converted the ontological model into a Bayesian network, and inferred higher-level contexts from the Bayesian network. Their work aimed to solve the uncertainty that is caused by the limit of sensing technologies and inaccuracy of the derivation mechanisms. Bayesian networks are best suited to applications where there is no need to represent ignorance and prior probabilities are available [33].

Any decision may be made incorrectly on account of errors in input data, and we simultaneously cannot blame poor performance on poor input data quality: we must instead construct models that accommodate uncertainty and error across the software system, and allow low-impact recovery [34].

3.3 Intelligibility of Context Reasoning

Interaction can be more useful if it is scrutable or intelligible. Intelligibility is defined as "*an application feature including supporting users in understanding, or developing correct mental models of what a system is doing, providing explanations of why the system is taking a particular action, and supporting users in predicting how the system might respond to a particular input*. " [35]. On one hand, a system will make decisions by taking all input, explicit or implicit to users, from sensors embedded in an environment. It uses its knowledge base in reasoning, while it has limited ability in ruling out random input or understanding which input is more important than another in determining actions. On the other hand, a user may have little understanding of what a system considers to be input and why a particular action is taken. Making a system intelligible will benefit both the system and end-users. The system will provide a suitable interaction interface to users so as to explain its actions, while users can provide feedback through the interface so that the system can adjust its behavior and provide services that match users' desire much better in the future.

4. Embedded Agents

Embedded agents [36] offer an effective model for designing and implementing solutions that must capture and interpret context parameters from disparate and distributed sources. Such agents have been deployed in a variety of situations including user interface implementation on mobile devices [37], realizing an intelligent dormitory for students [38] and realizing mobile information systems for the tourism [39] and mobile commerce domains [40] respectively.

4.1 The Agent Paradigm

Research in intelligent agents has been ongoing for over two decades now. What it is that defines an agent is open to debate. For the purposes of this discussion, agents are considered, somewhat simplistically perhaps, to be one of two varieties – reactive and deliberative. The interested reader is referred to Wooldridge and Jennings [41] for a more systematic treatment of agents and agent architectures.

Reactive agents respond to stimuli in their environment. An event occurs, the agent perceives it and responds using a predefined plan of action. Such agents are quite simple to design and implement. A prerequisite for their usage is that the key events or situations can be clearly defined and that an equivalent plan of action can be constructed for each circumstance. Such agents can be easily harnessed for explicit interaction modalities as their response time is quick.

Deliberative agents reflect and reason before engaging in an action. Essential to their operation is a reasoning model; hence they may be demanding from a computational perspective and their response time may be unacceptable. Such agents maintain a model of both themselves and the environment they inhabit. Identifying changes in the environment enables them both to adapt to the new situation and affect changes within the environment in certain circumstances. Such agents are useful for implicit interaction in that they enable transparent monitoring of an end-user and their inherent reasoning ability allows to come to some decision about as to if and when an implicit interaction episode has occurred, and what the appropriate course of action might be.

4.2 Coordination & Collaboration

As has been discussed, all interaction takes place within a context and an understanding of the prevalent context can usually make the meaning of the interaction itself more clear. However, gathering and interpreting select aspects of the prevalent contexts is process fraught with difficulty. And it is in addressing this that the agent paradigm can be harnessed to most benefit.

Agents are inherently distributed entities. Coordination and collaboration are of fundamental importance to their successful operation. To this end, all agents share a common language or Agent Communication Language (ACL). Indeed, the necessity to support inter-agent communication has resulted in the development of an international ACL standard, which has been ratified by the Foundation for Intelligent Physical Agents (FIPA). FIPA has recently been subsumed into the IEEE computer society, forming an autonomous standards committee seeking to facilitate interoperability between agents and other non-agent technologies.

4.3 The Nature of the Embedded Agents

Embedded Agents are lightweight agents that operate on devices of limited computational resources. Ongoing developments in computational hardware have resulted in agents become viable on resource limited platforms such as mobile telephones and

nodes of Wireless Sensor Networks (WSNs). While such agents may be limited in what they can do on such platforms, it is important to remember that they can call upon other agents and resources if the physical communications medium supports an adequate QoS. Thus a multi-agent system may itself be composed of a heterogeneous suite of agents – some significantly more powerful than others but all collaborating to fulfill the task at hand. Such a model of an MAS is reflective of the diverse suite of hardware that is currently available and may be harnessed in diverse domains.

As an example of how embedded agents might collaborate, consider the following scenario. While observing how a user interacts with an arbitrary software package, the user wipes their brow. This gesture, done subconsciously, is observed and identified. However, what does it mean in this context? It may indicate that the user is stressed or it may be a cue to indicate that the ambient office temperature is too high. In the later case, it would not be difficult to confer with an agent monitoring a temperature sensor to identify the current temperature and check whether it an average figure or maybe too high. If considered high, a request could be forwarded to the agent in charge of air-conditioning to reduce the ambient temperature. In the case where the user is stressed, and there may other cues to affirm this, the situation is more complicated. Is the user stressed because of the software or hardware itself? or because of the task they are trying to accomplish? or because of some other circumstance? And does it really matter? In some cases, it may be quite important to know if a user is stressed, especially if they operating a vital piece of equipment, for example in a medical theatre or air control context. As to what the equipment should do if it senses that its operator is under stress is debatable, and may even raise ethical issues. However, it can be reasonably conjectured that the team leader or project manager might find it useful to know that one of their team members could be having difficulty and that some active intervention, though precautionary, might be a prudent course of action.

While agents encompass a suite of characteristics that make them an apt solution for identifying episodes of implicit interaction, a further level of abstraction would be desirable if implicit interaction is to become incorporated in mainstream computing. In the next section, we consider how this might achieved, specifically through the middleware construct.

5. Characteristics of a Middleware for Implicit Interaction

In the previous sections, the issues of reasoning with uncertain contexts was discussed. Likewise, the agent paradigm was considered in light of its inherent distributed nature as a means for capturing and interpreting contextual states. Except in the simplest cases, interaction cannot be divorced from the context in which it occurs. Thus the key challenge that must be addressed is how to incorporate implicit interaction into the conventional software development lifecycle.

Requirements Analysis

Requirements analysis is concerned with the identification of what it is that either a new system or modified system is expected to do. Various techniques have been proposed for eliciting user requirements. In particular, the key stakeholders are identified

and their needs specified. The question of how issues relating to interaction may be addressed depends on the approach adopted. During interviews, there is scope for identifying how users perceive interaction occurring and opportunities for incorporating alternative interaction modalities, including implicit modalities. Given the time and budgetary constraints that a project may labor under, it may be questionable as to what scope software engineers have to do this. Not only must they obtain a thorough understanding of what a proposed system must do but they must also gain and in-depth of the target user group including their needs, backgrounds and expectations.

Should the requirements phase of a project include rapid prototyping, a greater understanding of how potential users envisage interaction with the proposed system may be gleaned. In such circumstances, a mockup is constructed resulting in users seeing clearly how the interaction is planned and enabling the system designers to ascertain the possibility and desirability of incorporating an implicit interaction component. Whether the average software engineer is the best person for this task is an open question. In principle, such a task would be undertaken by usability professionals. In practice, such people may not become involved in the project until the next stage, if indeed at all.

Design & Specification

The objective here is to provide a systemic description of what a system will do. Naturally, all elements of how interaction will occur need to be agreed at this stage. First of all, there needs to be agreement on whether the interaction requirements would be best served by harnessing implicit interaction, or indeed, other interaction modalities. Various factors will influence this decision, for example, will the user base accept what they might perceive as non-conventional interaction modalities? More importantly, there may be a trade-off between system performance or responsiveness and what interaction modality is adopted. Such a trade-off needs to be quantified. The implications for project planning must also be considered. Though there may be an excellent usability case for supporting an arbitrary interaction modality, the time-scale, budget or deployment configuration may preclude their realization in the project.

Implementation

At this stage, all the key decisions have been made. It only remains for them to be implemented. From an implementation perspective, realizing implicit interaction is just another programming task that must be completed such that it adheres to the design. However, it must be stated that programmers and designers currently have little to aid them should a request to include implicit interaction be forthcoming. Thus the resultant solution, which may operate perfectly, is really an ad-hoc solution. If such interaction is to be incorporated into mainstream software development, a prerequisite will be that this process is transparent and easy to manage. At present, that is not the case. How this deficiency may be remedied is considered next.

5.1 Making Implicit Interaction Mainstream

Conventional software development environments include a range of widgets with associated behaviors that programmers can use in their designs and implementations. Such widgets can be customized and adapted according to a range of policies and application-specific requirements. This is the prevalent approach adopted in mainstream computing where the interaction modality is predominantly explicit. Thus the principles underpinning this approach are well understood, and codes of best practice have been identified. This is not the case with implicit interaction.

Once a decision has been made to either discard the classic WIMP interface, or even augment such interfaces with additional modalities, the creativity and ingenuity of the programmer will be required to craft a solution. It is worth reiterating that the model of interaction being adopted, at this stage of the software development process, will have been agreed and its behaviors specified. Thus the programmer has the singular task of implementing the design without necessarily being concerned with the merits or otherwise of the interaction modality being used. Their problem is that the lack of widgets will oblige them to create new solutions – a creative endeavor perhaps but one which may be costly in terms of time. Such a scenario may well be replicated in diverse projects; thus a key challenge is develop a framework that allow software developers incorporate implicit interaction seamlessly into their products.

5.2 Towards a Middleware for Implicit Interaction

Implicit interaction may be unimodal or multimodal. Though an implicit interaction "event" may be said to have occurred, the event-driven model adopted in conventional software systems is not appropriate in this circumstance, at least not without modification, as users are not directly interacting with the system but rather doing so indirectly through a variety of behavioral cues. Though such cues are initiated by the user, they will not be communicated directly to the software system. Rather the software must act in a proactive manner to capture and interpret such cues, rather than just react to user stimuli. Thus developing a suite of APIs for capturing and interpreting a range of implicit interaction behaviors, though attractive, is not an option. A robust solution is called for and to this end, it is proposed that one based on the middleware concept offers one approach for enabling the seamless integration of implication interaction into conventional computing.

Middleware has been conventionally viewed as a service provision layer that sits above the OS and networking layers but below domain specific applications [42]. Frequently seem as framework for ensuring interoperability, the middleware construct has been adopted in a diverse range of applications and domains, for example smart phones [43] and wireless sensor networks [44] offers a useful mechanism for providing a higher level of abstraction than that offered by conventional APIs. In the context of this discussion, it is instructive to note that middleware has been harnessed in the HCI domain. For example, Yaici and Kondoz [45] describe a middleware for the generation of adaptive user interfaces on resource-constrained mobile computing devices. Likewise Repo and Riekki [46] adopt a middleware approach for realizing context-aware multimodal user interfaces.

Middleware offers an attractive framework for incorporating implicit interaction into mainstream computing. The framework itself may be implemented in a variety of ways. However, in light of the discussion on agents, it can be seen that agents encompass a suite of characteristics that make them a suitable basis for such a framework. Indeed, the framework could be designed such that it acts as a wrapper for a Multi-Agent System. In this way, a standardized interface to the middleware could be provided to the software developer while the developers themselves are shielded from the intricacies of both MAS development and the effort required to capture and classify instances of implicit interaction.

5.3 Case Study: The SIXTH Middleware Architecture

An ongoing project in our laboratory concerns the deign and development of an middleware for sensor networks. SIXTH takes a broad interpretation of what a sensor might actually entail. At its simplest, a sensor network might compromise a network of nodes, for example motes. However, sensors can vary significantly in their capabilities, and might include a range of artifacts that on first sight might appear to have little in common with the conventional view of what a sensor actually is. For example, a surveillance camera network is essentially a sensor network. Likewise, fabrics imbued with heart rate monitors and other physiological measuring instrumentation might comprise a sensor network.

SIXTH is motivated by two observations:

1. Practical sensor networks will be heterogeneous. This heterogeneity will be expressed in a number of ways. Specifically a range of sensors differing in capability, communications mechanisms and supporting a range of sensed modes will form networks that support a range of diverse applications. Only in specialized sensor applications, for example, environmental applications, will homogeneous networks be the norm. In the case of implicit interaction, it can be seen that a network of cameras and audio receivers would be essential just to capture vocal cutes, gestures and facial expressions.
2. Sensor networks must be usable. In essence, the functionality encapsulated in sensors must be abstracted in an intuitive fashion such that it can be harvested and used by a variety of service providers. Only in this way, will sensor networks become incorporated into mainstream computing applications and services.

Thus SIXTH aims to encapsulate the following characteristics:

- scalability;
- reusability;
- flexibility;
- openness;
- extensibility;
- modularity.

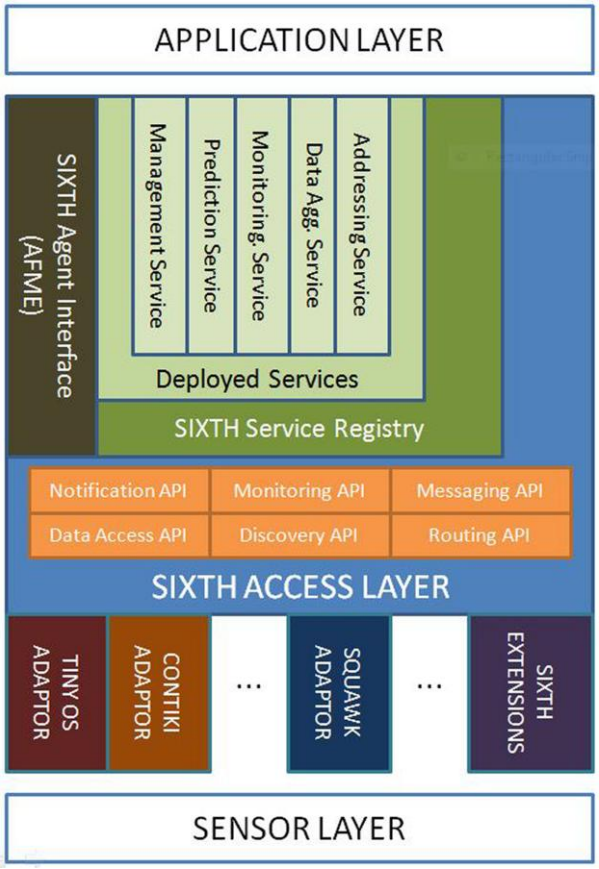

Fig. 4: Constituent components of the SIXTH middleware architecture.

Figure 4 illustrates the key components of the SIXTH architecture. It comprises three core layers:

1. Adaptor Layer: This layer contains device specific adaptors that utilize the native resources on the individual sensor itself and exposes them to the higher layers of the middleware.
2. API Layer: This layer implements a set of device agnostic APIs that can be used (in principle) to interface with any deployed sensor device. It provides support for addressing, (re-) programming of sensors; discovery of devices, monitoring of devices, and data access.
3. Service Layer: This layer augments the basic functionality provided by the API Layer to deliver higher level services that are tailored to the specific applications that require access to the underlying devices.

Layers 1 and 2 are designed to address the issue of heterogeneity. Layer 3 provides a mechanism for integrating new services, enabling their transparent and intuitive use in a range of applications.

At each layer, the components can be reused in many contexts to delivery multiple applications without the requirement for redevelopment of lower level functionality. The interface between the Adaptor and API Layers has been designed to embrace a multiplicity of abstractions that facilitate diverse modes of interaction with the embedded devices.

Finally, SIXTH supports embedded intelligence, that is, support for in-situ reasoning via the deployment of intelligent agents. Any agent platform that can operate on a Java 2, Micro-edition (Java ME) platform, for example, Agent Factory Micro Edition (AFME) [47] will work with SIXTH.

6. Conclusion

As computation technologies permeate more areas of everyday life, the need for a range of interaction modalities will become increasingly urgent, particularly if the promise of seamless and intuitive interaction is to become a reality rather than the aspiration it is at present. This paper explored the concept of implicit interaction, explaining its genesis and reflecting on how it might be incorporated into mainstream, computing.

Further basic research is needed into understanding what it is that defines implicit interaction. As a start, it may be feasible to develop a classification of non-verbal cues that people normally use and attempt to attach semantic meaning to them. A cultural perspective on these needs to be maintained also. Furthermore, the computational effort that must be expended in capturing implicit interaction needs to be quantified, particularly if a range of embedded artifacts are used for interaction capture. Likewise the time expended both in capturing and interpreting must be quantified so that an adequate response time can be estimated thus ensuring the quality of the user experience is maintained. Only when a more thorough understanding of the underlying principles is obtained can the practical issue of service implementation be considered.

Acknowledgements. This work is supported by Science Foundation Ireland (SFI) under grant 07/CE/I1147.

References

1. Maybury, M.T., Wahlster, W. (Eds.): Readings in Intelligent User Interfaces, Morgan Kaufmann Publishers Inc. (1998)
2. Argyle, M., Slater, V., Nicholson, H., Williams, M., Burgess, P.: The Communication of Inferior and Superior Attitudes by Verbal and Non-verbal Signals, British Journal of Social and Clinical Psychology, 9, 221-31. (1970)
3. Ekman, P., Friesen, W.V.: The Repertoire of Nonverbal Behavior: Categories, Origins, Usage, and Coding. Semiotica, 1, 49–97. (1969)

4. Ambady, N., Rosenthal, R.: Thin Slices of Expressive Behavior as Predictors of Interpersonal Consequences: A meta-analysis. Psychological Bulletin, 1(11), 256-274 (1992)
5. Ekman, P., Rosenberg, E.L. (eds.): What the Face Reveals: Basic and Applied Studies of Spontaneous Expression using the FACS. Oxford University Press, Oxford. (2005)
6. Cowie, R., Douglas-Cowie, E., Tsapatsoulis, N., Votsis, G., Kollias, S., Fellenz, W., Taylor, J.: Emotion Recognition in Human-Computer Interaction. IEEE Signal Processing Magazine, 18(1), 32-80. (2001)
7. Oviatt, S.: Multimodal Interactive Maps: Designing for Human Performance. Human-Computer Interaction 12(1), 93-129 (1997)
8. Sebe, N.: Multimodal Interfaces: Challenges and Perspectives. Journal of Ambient Intelligence and smart environments, 1(1), 23-30. (2009)
9. Kuchenbecker, K.J., Fiene, J., Niemeyer, G.: Improving Contact Realism through Event-based Haptic Feedback. IEEE Transactions on Visualization and Computer Graphics, 12 (2), 219-230, (2006)
10. Charumporn, B., Omatu, S.: Classifying Smokes using an Electronic Nose and Neural Networks. Proceedings of the 41st SICE Annual Conference, 5, 2661-2665, (2002)
11. Ciosek, P., Wróblewski, W.: Sensor Arrays for Liquid Sensing – Electronic Tongue Systems, Analyst, 132, 963-978 (2007)
12. Johnston, M., Bangalore, S.: Finite-state Multimodal Integration and Understanding. Journal of Natural Language Engineering 11(2), 159–187 (2005)
13. Norman, D. A.: The Design of Everyday Things. Doubleday. (1989)
14. Dix, A., Finley, J., Abowd, G., Beale, R.: Human–Computer Interaction. 3rd Edn. Prentice-Hall (2004)
15. Obrenovic, Z., Starcevic, D.: Modeling Multimodal Human–Computer Interaction, IEEE Computer, 37 (9) 65-72 (2004)
16. Greenberg, S., Context as a Dynamic Construct. Human-Computer Interaction, 16 (2–4), 257–268. (2001)
17. Dey, A. Modeling and Intelligibility in Ambient Environments. Journal of Ambient Intelligence and smart environments, 1(1), 57-62. (2009)
18. Gu, T., Pung, H.K., Zhang, D.Q.: A Bayesian Approach for Dealing with Uncertain Contexts. Proceedings of Advances in Pervasive Computing. pp. 205-210 (2004)
19. Henricksen, K., Indulska, J.: Developing Context-aware Pervasive Computing Applications: Models and Approach. Pervasive and Mobile Computing 2(1) 37-64 (2006)
20. Chen, H., Finin, T., Joshi, A.: An Ontology for Context-Aware Pervasive Computing Environments. Knowledge Engineering Review 18(3) 197--207. (2004)
21. Ranganathan, A, Campbell, R.: An Infrastructure for Context-awareness Based on First Order Logic. Personal Ubiquitous Computing, 7(6) 353-364 (2003)
22. Ye, J, Coyle, L., Dobson, S. Nixon, P.: Ontology-based Models in Pervasive Computing Systems. Knowledge Engineering Review. 22(4) 315--347 (2007)
23. Loke, S. W. Representing and Reasoning with Situations for Context-aware Pervasive Computing: A Logic Programming Perspective. Knowledge Engineering Review, 19(3) 213--233. (2004):
24. Schilit, B., Adams, N., Want, R.: Context-Aware Computing Applications. Workshop on Mobile Computing Systems and Applications. pp. 85-90, IEEE, New York (1994)
25. Henricksen, K., Indulska, J.: Modelling and Using Imperfect Context Information, pp. 33-37, IEEE, New York (2004)
26. Anagnostopoulos, C.B., Ntarladimas,Y., Hadjiefthymiades, S.: Situational Computing: An Innovative Architecture with Imprecise Reasoning. System and Software 80(12) 1993-2014 (2007)
27. Ye, J, McKeever,S., Coyle, L., Neely, S., Dobson, S.: Resolving Uncertainty in Context Integration and Abstraction. Proceedings of the International Conference on Pervasive Services, pp. 131-140, ACM, New York (2008)

28. Ranganathan, A., Al-Muhtadi, J., Campbell, R.: Reasoning about Uncertain Contexts in Pervasive Computing Environments. IEEE Pervasive Computing 3(2) 1536-1268, (2004)
29. Ding, Z., Peng, Y.: A Probabilistic Extension to Ontology Language OWL. Proceedings of the 37th Hawaii International Conference on System Sciences, pp. 1-10 (2004)
30. Truong, B.A., Lee,Y-K, Lee, S-Y.: Modeling Uncertainty in Context-Aware Computing" Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05), pp. 676-681 (2005)
31. Dargie, W.: The Role of Probabilistic Schemes in Multisensor Context-Awareness. Proceedings of Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops. pp. 27-32 (2007)
32. Ye, J., Coyle, L.,Dobson, S. Nixon, P.: Using Situation Lattices to Model and Reason about Context. Proceedings of the Workshop on Modeling and Reasoning Context, pp. 1-12 (2007).
33. Hoffman, J.C., Murphy, R.R.: Comparison of Bayesian and Dempster-Shafer theory for sensing: a practitioner's approach. Proceedings of Neural and Stochastic Methods in Image and Signal Processing II, pp. 266-279 (1993)
34. Ye, J., Dobson, S., Nixon, P.: An Overview of Pervasive Computing Systems. In Augmented Materials and Smart Objects: Building Ambient Intelligence through Microsystems Technology, 3-17. Springer-Verlag (2008)
35. Dey, A.: Modeling and Intelligibility in Ambient Environments. Journal of Ambient Intelligence and Smart Environments, 1, 57-62. (2009)
36. O'Hare, G.M.P., O'Grady, M.J., Muldoon, C., Bradley, J.F., Embedded Agents: A Paradigm for Mobile Services, Int. Journal of Web and Grid Services 2(4) 379-405 (2006)
37. O'Hare, G.M.P., O'Grady, M.J., Addressing Mobile HCI Needs through Agents. In: Paterno. F. (Ed.) LNCS, vol. 2411, pp. 311–314, Springer, Heidelberg (2002)
38. Hagras, H., Callaghan, V., Colley, M., Clarke, G., Pounds-Cornish, A., Duman, H.: Creating an Ambient-Intelligence Environment Using Embedded Agents. IEEE Intelligent Systems 19(6) 12-20, 2004)
39. O'Grady, M.J., O'Hare, G.M.P., Sas, C.: Mobile Agents for Mobile Tourists: A User Evaluation of Gulliver's Genie, Interacting with Computers, 17 (4) 343-366 (2005)
40. Keegan, S. O'Hare, G.M.P., O'Grady, M.J.: EasiShop: Ambient Intelligence Assists Everyday Shopping, Information Sciences, 178 (3) 588-611 (2008)
41. Wooldridge, M. Jennings, N.R. Intelligent Agents: Theory and Practice. The Knowledge Engineering Review 10(2) 115-152. (1995)
42. Bernstein, P. A.: Middleware: A Model for Distributed System Services. Communications of the ACM 39(2), 86-98 (1996)
43. Riva O, Kangasharju J.: Challenges and Lessons in Developing Middleware on Smart Phones. Computer 41, 23–31 (2008)
44. Fok,C., Roman,G., Lu,C.: Mobile Agent Middleware for Sensor Networks: An Application Case Study, IPSN (2005)
45. Yaici, K.; Kondoz, A.: Runtime Middleware for the Generation of Adaptive User Interfaces on Resource-constrained Devices. Third International Conference on Digital Information Management, 2008, pp.587-592. (2008)
46. Repo, P. and Riekki, J.: Middleware Support for Implementing Context-aware Multimodal User Interfaces. In: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia(MUM '04), pp. 221-227. ACM, New York (2004).
47. C. Muldoon, G. M. P. O Hare, R. W. Collier, and M. J. O Grady.: Towards Pervasive Intelligence: Reflections on the Evolution of the Agent Factory Framework. In: Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, E.F. (eds.) Multi-Agent Programming: Languages, Platforms and Applications, pp. 187-210. Springer, Heidelberg (2009)