Building Intelligent Environments By Adding Smart Artifacts to Spaces: a Peer-to-Peer Architecture

Seng W. Loke
Department of Computer Science and Computer Engineering
La Trobe University
VIC 3086, Australia
Email: s.loke@latrobe.edu.au

***Abstract*—We envision an intelligent environment comprising collections of smart artifacts, each artifact with an embedded processor, networking and sensing capabilities. The interactive capabilities of the environment are due to the collective working of the smart artifacts. This position paper proposes a rule-based declarative programming model for programming intelligent environment behaviours involving a collection of smart artifacts. We outline our language, provide examples and highlight issues. We contend that a peer-to-peer architecture forms a useful approach to building (and extending, over time) intelligent environments, i.e., by adding (perhaps incrementally over time, a few artifacts at a time) cooperative programmable smart artifacts to physical environments.**

I. Introduction

Intelligent environments are "physical environments in which information and communication technologies and sensor systems disappear as they become embedded into physical objects, infrastructures, and the surroundings in which we live, travel, and work."[1] Such physical objects may be everyday objects such furniture, vases, picture frames, appliances or simply the walls of the room.

We take the perspective that, while modern intelligent environments can be built from scratch by embedding computers and sensors into the room itself (e.g., the floor and walls), a normal physical environment such as a living room can be turned into an "intelligent environment" by adding smart artifacts. For example, consider an ordinary living room in an old house, and add a collection of vases and digital picture frames, each vase and picture frame with an embedded processor and sensors, and can network/communicate with each other, cooperatively monitoring the activities of the inhabitants and adjusting the environment accordingly through the artifacts' capabilities (e.g., showing the right photographs and music according to sensed gestures of the user - where the artifacts are embedded with cameras, touch sensors, motion sensors and/or distance sensors, etc).

Programming such an intelligent environment reduces to programming the distributed collection of computers within the vases and the picture frames collectively forming "The Computer" with, effectively, a peer-to-peer system architecture. Such a peer-to-peer architecture is naturally extensible to new artifacts added to the environment - all it takes is for the new smart artifact (e..g., a new digital picture frame added to the wall of the room) to make itself known (as a friend) to one other artifact (e.g., via Bluetooth or other short range networking technology) and then the network of artifacts (including the new) is connected.

Further adding smart artifacts (e.g., smart furniture) into the room (such furniture with sensors and the capabilities to interact with the picture frames and vases) further enhances the living room environment, adding behaviours that can be described as intelligent - e.g., sensing the user sitting on the chair and adjusting the music and pictures displayed accordingly. The idea is that a collection of smart artifacts can be added to almost any physical environment in order to create intelligent environments. This idea is perhaps similar to the notion of piling up technologies (or devices) at a place McCullough [11], except that we also want these devices to cooperate whether in sensing to recognize human activity (e.g., [12], [17], [16]) or supporting the user in accomplishing tasks [8].

What are smart artifacts? There has been rapid recent developments in the notion of smart artifacts. Smart artifacts are capable of computationally based behaviours, either by virtue of sensing their physical property (e.g., position, presence, state, or how they are handled) and then intelligently responding to that via some computation, or they might be everyday objects (e.g., vases, books, furniture, clothings, etc) endowed with embedded processors, networking and sensing capabilities, or new types of devices [14]. The notions of *spime* [13], context-aware smart artifacts, smart objects, and things that think[2] (e.g., as in [5], [1], [2], [6]) are changing the way we view ordinary everyday objects and also defining new kinds of things with networking, computational and sensing capabilities.

A number of paradigms have been employed to program

[1] http://en.wikipedia.org/wiki/Intelligent_environments

[2] http://ttt.media.mit.edu/

such artifacts, including Web service based methods,[3] that assume each device has an embedded Web server that receives and responses to invocations, UPnP service styles,[4] JINI,[5] and peer-to-peer based styles such as JXTA,[6] and various visual editors such as [4], [3], [10]. Recent times have seen the application of rule-based programming for behaviours of collections of smart artifacts, such as [15], [18], [4]. Basically, context information gathered via sensors are reasoned with and appropriate action taken based on these rules.

While work such as [15] did use a Prolog-based language implemented on tiny computers, and reasoning was via Prolog based rules, they did not consider cooperative reasoning among artifacts by one artifact sending a query to another artifact to continue reasoning. But in [5], is briefly sketched the vision of goal evaluation using rules on one artifact that results in a subgoal (or query) being sent to another artifact, initiating another goal evaluation there. Subsequently, further subgoal evaluations might yield another query being sent to a third artifact and so on, resulting in a peer-to-peer goal evaluation among a collection of artifacts. However, there are no details of an implementation model or language.

LogicPeer [7] is a Prolog-based peer-to-peer distributed programming model. In this paper, we propose *ActiveArtifacts-L*, an adaptation of LogicPeer for smart artifacts as one way of realizing the vision in [5]. The advantage is a semantically sound basis for cooperative rule-based query processing for a collection of artifacts, where each artifact has its own set of rules (a logic program), and so, the artifacts are viewed effectively as a collection of logic programs.

II. A Language for Active Artifacts

LogicPeer is an extension of Prolog with operators that pass subgoal evaluations to peers. We provide the operational semantics of *ActiveArtifacts-L*, which is a simple variation of that of LogicPeer, with the main difference that each peer is a smart artifact; *ActiveArtifacts-L* has two operators, logic program *union* and *peer-switching*. First, the syntax of rules are as follows:

$$\mathcal{A} : -\mathcal{G}$$

where $\mathcal{A}$ is an atom, and goal $\mathcal{G}$ is given by:

$$\mathcal{G} ::= \mathcal{A} \mid \mathcal{E} * \mathcal{G} \mid (\mathcal{G}, \mathcal{G})$$

where "$*$" is the *peer-switching* operator which redirects subgoal evaluations to a different peer, and $\mathcal{E}$ identifiers one or more peers given by:

$$\mathcal{E} ::= \mathcal{P} \mid \mathcal{E} + \mathcal{E}$$

where $\mathcal{P}$ is a peer identifier, which is unique (at least within the collection under consideration) for a given artifact (this could map to a low-level identifier such as an IP address, a Bluetooh ID, an RFID tag ID, or a friendly name resolved to a low-level identifier).

The Plotkin-style rules provide the semantics as follows which extends that of pure Prolog:

$$[true]\frac{}{E \vdash_\epsilon true}$$

$$[atom]\frac{E \vdash_\theta H : -G \ \wedge \ \gamma = mgu(A, H\theta) \ \wedge \ E \vdash_\delta G\theta\gamma}{E \vdash_{\theta\gamma\delta} A}$$

$$[peer - switching]\frac{F \vdash_\theta G}{E \vdash_\theta F * G}$$

$$[conjunction]\frac{E \vdash_\theta G_1 \ \wedge \ E \vdash_\gamma G_2\theta}{E \vdash_{\theta\gamma} G_1, G_2}$$

$$[union]\frac{E \vdash_\theta H : -G}{E + F \vdash_\theta H : -G}, \frac{F \vdash_\theta H : -G}{E + F \vdash_\theta H : -G}$$

$$[clause \ from \ peer]\frac{(H : -G) \ found \ at \ P}{P \vdash_\epsilon H : -G}$$

The union operator selects, nondeterministically, one of the peers to which goal evaluation is directed. In practice, several peers might be selected until one succeeds. The peer-swtiching operator effectively passes (in a message) evaluation of the goal G from E to another peer(s) F and then retrieves the results. Repeated uses of peer-switching leads to reasoning chains in [5], but in our model, with multiple subgoals performing peer-switching or the use of union, a reasoning tree results involving multiple artifacts (or peers). We also assume, for practical purposes, that conjunction proceeds one goal at a time, first do G_1 with E and then on completion (and success), do G_2 with the same composition E. Also, in practice, for union, we might use short-circuit evaluation from left to right, stopping when one succeeds.

III. An Illustrative Example

Consider a collection of vases, each with an embedded processor fitted with an interpreter for *ActiveArtifacts-L*. We assume that the vases can communicate with each other via Bluetooth (A vase might not be able to maintain two Bluetooth connections at a time, in which case, it needs to disconnect from one, and reconnect to another, and then reconnect later). Also, Bluetooth has a limit of up to seven other friends that a peer can simultaneously connect to in a piconet, but this can be overcomed (albeit with delays) by disconnecting and connecting.

Let `v1`, `v2` and `v3` be three vases and suppose that each is able to play a guitar chord: "d", "g", and "a" respectively. A song typically comprises a sequence of such chords, each chord being played for a certain period of time (say, in this example four beats). Assume that we have a song "s" with the following sequence "daadgdad", each chord played for the same duration of four beats. We can represent this by the following rule:

```
play_song(s) :-
  play(d), play(a), play(a),
  play(d), play(g), play(d),
  play(a), play(d).
```

[3]See http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01 and [9]
[4]See http://www.upnp.org/
[5]See http://www.jini.org/wiki/Main_Page
[6]See https://jxta.dev.java.net/

where the play/1 predicate plays the given chord for four beats (of a predefined tempo). With our three vases each playing a different chord, our rule should then be:

```
play_song(s) :-
  v1*play(d), v3*play(a), v3*play(a),
  v1*play(d), v2*play(g), v1*play(d),
  v3*play(a), v1*play(d).
```

Then, if one of the vases, say v1 plays the "lead" role which receives a goal from the user: ?- v1*play_song(s), then the song is played given the rule above residing in the knowledge base of v1. But suppose that play(d) only succeeds with v1, play(g) only succeeds with v2 and also, play(a) only succeeds with v3, the rule could also be written as:

```
play_song(s) :-
  (v1+v2+v3)*(play(d), play(a), play(a),
              play(d), play(g), play(d),
              play(a), play(d)).
```

which is simpler for the user as s/he does not need to know (or to specify) which vase plays which chord, and the system "works out" which vase plays what (but with redundant wasted messages to vases that don't play the right chord). This is in accordance with the semantics of conjunction above which effectively means the evaluation goes

```
(v1+v2+v3)*play(d)
```

then

```
(v1+v2+v3)*play(a)
```

then

```
(v1+v2+v3)*play(a)
```

and so on.

This becomes a lot more useful if we have say twenty chords and twenty vases, and have a song involving ten different chords. Also, the expression (v1+v2+v3) is effectively the logic program for the collection of three vases, formed from the individual logic programs of the vases.

However, it could be that the lead vase v1 does not know of all the other vases. Then, a peer-to-peer model is useful. For example, consider another song represented by this rule:

```
play_song(s2) :-
  play(d), play(a), play(a),
  play(c), play(e), play(d),
  play(g), play(a), play(d).
```

and suppose each vase has the additional rule for chords it cannot play, say for v1:

```
play(Chord) :-
   Chord \= a,
       // chord is not ``a''
   friends(Friends),
   (+Friends)*play(Chord).
       // pass on the subgoal
```

which determines that if the chord being asked to be played is not "a", it will retrieve a database of peers that it knows about (or its *friends*, who are assumed to be stored in the relation friends/1), and then passes the evaluation to the union of its friends (the prefix notation +Friends means a union of the friends in the list Friends).

The song s2 involves chords "c" and "e" which are not playable by v1, v2 and v3. Hence, the rule on v1

```
play_song(s2) :-
  (v1+v2+v3)*(play(d), play(a),
              play(a), play(c),
              play(e), play(d),
              play(g), play(a),
              play(d).)
```

would now result in friends of v1, v2 or v3 being consulted for the two chords.

However, note that now, it is possible that the same chord be played more than once. To ensure that every goal (or chord) is only executed (or played) once, we can have each subgoal sent from the lead vase v1 assigned a unique identifier in the lead vase, so that a subgoal that is received and passed on to friends is passed on with the same identifier. This means that a fourth vase, v4 say, on receiving a subgoal (to play chord "c" say) checks to see if it has executed the subgoal, and if not, it plays the chord, and if it has, it drops the subgoal.

Vase v1 may have motion sensors and distance sensors that recognize specific user movements and gestures, which result in the goal play_song/1 being invoked.

In a living room with such vases placed around the room (effectively now an intelligent environment), different user gestures and movements can be recognized and appropriate *ActiveArtifacts-L* rules triggered to coordinate a response to the user.

IV. Conclusion

We have proposed a flexible approach to turning a normal physical environment into an intelligent environment, that is, by adding a collection of smart programmable artifacts to the physical environment. We have also sketched *ActiveArtifacts-L*, an extension of pure Prolog, based on LogicPeer, to program behaviours of a collection of artifacts. The advantage is the declarative programming style, and the simple semantics: the program over a collection of artifacts can be viewed as the program (formed by the union of the individual logic programs residing in the collection of artifacts). Issues emerging include:

- *trade-off efficiency and costs for abstraction*: we saw that the union operator can provide simplicity for programming but involves overheads of messaging to implement.
- *handling redundancies*: our scheme works as we know a priori that a chord can only be played by exactly one

vase (but would still work if a vase could play multiple chords), but more general scenarios must be investigated, certainly drawing on the existing extensive work on peer-to-peer computing.

- *small footprint interpreters*: we have assumed vases with the ability to contain an *ActiveArtifacts-L* interpreter, but the impact of resource limitations on expressivity of programs must be investigated further.
- *incremental development*: the programming model is scalable in that individual programs need not change as long as the peers' individual friend databases are updated to record new artifacts - basically, an artifact need only to make itself known to one other artifact to join the peer network; however, bounds on execution times are then needed to ensure that subgoal evaluations are bounded by time and resources.

Future work will involve implementations and further examples of the above model, perhaps based on a JavaME or Android platform. We aim to further prototype and explore our notion of intelligent environments (as spaces endowed with collections of programmable smart artifacts) in settings such as shops and offices. While this paper proposes a peer-to-peer approach, hybrid peer-to-peer and centralized infrastructure techniques can also be employed.

REFERENCES

[1] Hans W. Gellersen, Albercht Schmidt, and Michael Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. *Mob. Netw. Appl.*, 7(5):341–351, 2002.

[2] Neil Gershenfeld. *When Things Start to Think*. Henry Holt and Co., Inc., New York, NY, USA, 1999.

[3] C. Goumopoulos and A. Kameas. Ambient ecologies in smart homes. *Comput. J.*, 52(8):922–937, 2009.

[4] Achilles Kameas, Irene Mavrommati, and Panos Markopoulos. Computing in tangible: using artifacts as components of ambient intelligence environments. In *In Ambient Intelligence: The evolution of Technology, Communication and Cognition (G.Riva, F.Vatalaro, F.Davide and M.Alcaniz,eds)*, pages 121–142. IOS Press, 2004.

[5] Gerd Kortuem, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14:44–51, 2009.

[6] Seng W. Loke. Context-aware artifacts: Two development approaches. *IEEE Pervasive Computing*, 5(2):48–53, 2006.

[7] Seng W. Loke. Declarative programming of integrated peer-to-peer and web based systems: the case of prolog. *J. Syst. Softw.*, 79(4):523–536, 2006.

[8] Seng W. Loke, Sea Ling, Gerry Butler, and Brett Gillick. Levels of abstraction in programming device ecology workflows. In *ICEIS (4)*, pages 137–144, 2005.

[9] Seng Wai Loke. Service-oriented device ecology workflows. In *ICSOC*, pages 559–574, 2003.

[10] Nicolai Marquardt, Tom Gross, M. Sheelagh T. Carpendale, and Saul Greenberg. Revealing the invisible: visualizing the location and event flow of distributed physical devices. In Marcelo Coelho, Jamie Zigelbaum, Hiroshi Ishii, Robert J. K. Jacob, Pattie Maes, Thomas Pederson, Orit Shaer, and Ron Wakkary, editors, *Tangible and Embedded Interaction*, pages 41–48. ACM, 2010.

[11] Malcolm McCullough. *Digital Ground: Architecture, Pervasive Computing, and Environmental Knowing*. MIT Press, Cambridge, MA, USA, 2004.

[12] Matthai Philipose, Kenneth P. Fishkin, Mike Perkowitz, Donald J. Patterson, Dieter Fox, Henry Kautz, and Dirk Hahnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3(4):50–57, 2004.

[13] Bruce Sterling. *Shaping Things*. MIT Press, 2005.

[14] Norbert A. Streitz, Carsten Rocker, Thorsten Prante, Daniel van Alphen, Richard Stenzel, and Carsten Magerkurth. Designing smart artifacts for smart environments. *Computer*, 38(3):41–49, 2005.

[15] Martin Strohbach, Hans-Werner Gellersen, Gerd Kortuem, and Christian Kray. Cooperative artefacts: Assessing real world situations with embedded technology. In *Ubicomp*, pages 250–267, 2004.

[16] Martin Strohbach, Gerd Kortuem, Hans-Werner Gellersen, and Christian Kray. Using cooperative artefacts as basis for activity recognition. In *EUSAI*, pages 49–60, 2004.

[17] Emmanuel Munguia Tapia, Stephen S. Intille, and Kent Larson. Portable wireless sensors for object usage sensing in the home: Challenges and practicalities. In *AmI*, pages 19–37, 2007.

[18] Tsutomu Terada and Masahiko Tsukamoto. Smart object systems by event-driven rules. In *Proc. of the 1st International Workshop on Smart Object Systems*, pages 100–109, 2005.