Situation Semantics for Things: Everyday Artifacts that Come with Pre-Specified Behaviours

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

ABSTRACT

This paper proposes the idea of providing artifacts with specifications of their situation transforming behaviour. Such specifications can be created by designers (and so would come together with the artifact on purchase) or end-users to implement smart things that transform their environment in meaningful ways. The specifications are based on situation semantics first developed for studies in linguistics.

1. INTRODUCTION

The notions of *spime* [13], context-aware smart artifacts, smart objects, and things that think[1] (e.g., as in [10, 8, 4, 5, 12]) are changing the way we view ordinary everyday objects and also defining new kinds of things with networking, computational and sensing capabilities.

We envision everyday objects with accompanied computational behaviour. Buying a new television and introducing it into the living room should automatically cause changes in or interactions with other appliances/devices in the living room (perhaps invisible to the end-user), or a new table (with an embedded computer such as the Microsoft surface computer) introduced into an office might begin to interact with other tables and perform various behaviours. In a similar way, any device or a coffee cup (with embedded computer and networking capabilities) introduced temporarily into a space might cause computational behaviour and underlying interactions automatically, whether for art, commerce, entertainment or other applications. And it may not only be the presence of an artifact that causes interactions but also the location of an object - e.g., placing a vase on the dining room table causes the dining room lights to turn on but placing the same vase on the living room coffee table causes a table lamp to turn on. In addition, placing a collection of vases (of a certain type) next to each other might cause certain songs to be playable from a set of nearby speakers (this could be used by businesses to reward buyers for buying a whole collection of vases, etc). Hence, a case can be made for packaging everyday objects (from furniture to vases to digital picture frames, etc) with accompanying computational behaviours as value-adding, triggered by

- their presence,
- their position,
- their movements,
- their proximity to other objects, or
- more generally, context changes

within their normally situated spaces.

People buy clothings and accessories to make a (fashion) statement or purchase things to transform their situation and lifestyle. In other words, an artifact, device, or object has the purpose (or intent) and/or ability to alter the current situation or environment in which they inhabit. For example, a lamp when switched on changes the environment from dark to bright, and a smart road sign (between two road segments with two different speed limits) is intended to change the speed of cars as drivers observe these signs.

A set of vases put together might begin to play orchestral music; arranging chairs together in a certain way in a cafe yield certain updates on public displays; a pill bottle at the right time and in the presence of the owner begins to glow till the owner takes the medication; throwing a ball among a group of people results in the ball changing colours; a new set of speakers introduced in the living room finds sounds from the TV or a radio being piped through it, and placing a plant in the middle of a shopping mall results in periodic reports about deforestation happening throughout the world. These are only a few examples of artifacts or things with situation transforming intent, whether the thing has its own capabilities to achieve the effect, the intent is achieved through a human in the loop influenced by the thing, or the thing effectively uses other resources, or triggers another system to try to accomplish its intent.

The general idea is that when an artifact is brought into a space (e.g., a living room), based on its current situation, the artifact causes a new situation to arise (which the artifact is then now in). If the same artifact is placed in another space (e.g., the kitchen), based on its new current situation, the artifact causes a new situation. The relationship and interaction between space and artifact has been explored in [7], but we highlight in this example that the artifact operationally represents a relationship between two situations, the situation which the artifact first occurs in and the situation which the artifact has caused. This by no means should terminate, the artifact may continuously transform the situation it is in, generating a sequence of time-ordered situations, but a simplification or a discretization (taking two snapshots in time) can be extracted, and so we deal with discrete situations in this paper.

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

MoMM2011, 5-7 December, 2011, Ho Chi Minh City, Vietnam.

In this paper, we present a perspective on an artifact based on its (designed or user-conceived) purpose to change its current situation. We define context to be any information that can be used to recognize the situation of an entity [3], and situation as a state of affairs in which an artifact finds itself. We formalize the fact that an artifact is intended to have some effect on situations; we provide a situation semantics for artifacts, where an operational meaning of an artifact can be given by a relation between a situation the artifact is in and a situation it is purposed to achieve. Certainly, a specification for the situation-changing behaviour of an artifact can be given separately, but we imagine that an artifact in the future will come together with its situation-transforming behaviour (specified in some standard language), only to be interpreted by the space (i.e., the space software) in which the artifact is placed. We provide a formalism for specifying the intended situation-changing behaviour of an artifact, effectively giving the artifact a semantics. Such specifications can then be executed to try to achieve the intended effects or used for reasoning about the behaviour of the artifacts. We describe this meaning of artifacts based on situation semantics [1] in Section 2, where we also explore the semantics of collections of things and consider modifiers on things. In Section 3, we briefly review implementation architectures. We conclude with future work in Section 4.

2. SITUATION SEMANTICS FOR THINGS

What is a thing? As mentioned earlier, various terminology including smart objects, Internet of Things, and context-aware artifacts have been used to describe everyday objects with one or more of the three capabilities: computation, networking and sensing. Our semantics applies to "things" which, here, refer to not just these objects with these capabilities but also those without such capabilities but tagged (e.g., RFID-tagged or Bluetooth tagged) so that they can be recognized and used to trigger situation changes. Situation semantics was first developed to study the semantics of natural languages [2, 1]. We use here the notion of situation from [2]: "Situations are parts of the world and the information an agent has about a given situation at any moment will be just a part of all the information that is theoretically available." Context, being information that can be used to determine context, can then be aggregated in a model of situations.

2.1 Things as Relations

With situation semantics, roughly speaking, the meaning of an expression ψ is taken to be a relation: $d, c[\![\psi]\!]e$ between an utterance or discourse situation d, a speakerÕs connection function c (which refers to objects mentioned), and a situation e described by the utterance. Here, the situations d and e can be finitely described by a set of non-contradictory facts (describing different types of context attributes and values), since they are only descriptions of some $part$ of the world. Correspondingly, we give the semantics of a thing t as a relation: $s, I[\![t]\!]s'$, between the situation s in which the thing is embedded, an interpreter function I which discerns if s occurs, and a situation s' representing the intended effect of t which is ascertained by I. Sometimes, we will leave I out in our expressions when that is implicit. Given specifications of the form $s, I[\![t]\!]s'$ as above, we identify six stages a system realizing I should support for their execution:

1. *determine if the situation s has occurred*: we consider the current situation of the artifact, and the system may use sensors to determine if s has occurred;
2. *explore the effects of s'*: it is useful to understand the effects of the intended artifact before they are executed; the idea is that the system can reason with these specifications and their possible effects before actually executing them, for example, either as a safeguard against undesired behaviours or to determine the likelihood of success;
3. *execute the specifications*: the system needs to figure out the appropriate operations to perform in order that s' may be effected, based on the current situation s;
4. *ascertain if the intended situation has been achieved*: the system can check to see if the operations produced the intended effects, perhaps by perceiving the situation via sensors after the operations complete, or by querying the status of device(s);
5. *update self-knowledge*: the system might then update its own knowledge base, either to record the success (or failure) of the specified artifact or to record the effects of a successfully achieved situation;
6. *inform the user*: we assume here some mechanism of feedback to users; sometimes this might not be necessary if the user can simply observe the new situation.

Our specifications describes an artifact as a relation between two situations, and goes beyond simple Event-Condition-Actions (ECA) rules by (i) representing (effectively) conditions as the prerequisite situation s which forms a set of coherent facts about the world, instead of a simple Boolean condition, (ii) mapping to situations instead of actions, allowing an interpreter I to figure out operations required to achieve the situation. We contend that, for situations, mapping between situations is at a higher level of abstraction than mere ECA rules. This also facilitates specifying things with *chained* effects. For example, for things t_1 and t_2, we have $s_1[\![t_1]\!]s_1'$ and $s_2[\![t_2]\!]s_2'$ and $s_2 \subseteq s_1'$ so that the resulting situation of t_1 enables t_2 to have effect. We can also have *ramified* effects, where we might also have thing t_3 so that $s_3[\![t_3]\!]s_3'$ and $s_3 \subseteq s_1'$ so that not just t_2 but also t_3 is enabled. We plan to exploit such *chained* and *ramified* effects in collections of things mediated by the physical world. (This idea also allows us to generalize the semantics to relations between *situation-types* instead of situations, where we define a situation-type as a set of situations.)

Consider a simple example of the intended effect of a $table\_lamp$:
$< dark = true >, I[\![table\_lamp]\!] < dark = false >$
where we rely on some interpreter I to elaborate on the exact meaning of $dark$ and not dark. This specifies the $table\_lamp$ as an artifact that turns a dark place to a place that is not dark. Given a system realizing I that can execute this specification, we effectively have an automatic table lamp. A light sensor could be used with a table lamp that is computer controllable (e.g., via an UPnP[2] or Web service interface). Note that a comparatively simple implementation of this is if a normal table lamp is used, we can implement this automatic behaviour by RFID tagging the table lamp and then having an RFID reader that detects the presence of the table lamp, and then when the light sensor readings are obtained, use an X10 controller to switch the lamp on or off. Hence, there are numerous ways to implement such a specification. But the idea is that the expression above is an abstract specification of the functional meaning of the table lamp, which is a relation between the situation $< dark = true >$ and $< dark = false >$.

Of course, one can use a more elaborate description of the situation $< room\_dark = true >$ and $< room\_dark = false >$ which refers to my room where the table lamp is located (this interpretation of the context $room\_dark$ is supplied by I which will

[2]http://www.upnp.org

map it to an appropriate range of readings of a light sensor in my room). There could be multiple table lamps with a similar specification because of their similar effects on dark rooms, and so such specifications are considered *abstract*, applicable to a variety of similar (at least with respect to their situational effects) things.

We also note that the situations to which a thing relates may have nothing to do with the capabilities of the thing. For instance, we may have

(1) $< room\_dark = true >, I[\![book]\!] < room\_dark = false >$

to mean that when a particular book is brought into the room, the book's presence triggers some light in my room to turn on so that my room is no longer dark. Effectively, we specified a situation altering thing: the book changes the lighting condition in my room. This could be implemented by, say, RFID tagging the book, and when an RFID reader in my room detects the (tag on the) book, the system then turns a light on. What we have done is then assigned situation semantics to the book, but this semantics does not relate to the book's capabilities. We could also assign a different semantics to the same book, as follows:

(2) $< quiet = false >, I[\![book]\!] < quiet = true >$

which makes a room quiet when the book is brought in. Now, both specifications can be composed without problems here, but this yields a slightly different meaning:

$\{< room\_dark = true >, < quiet = false >\}, I[\![book]\!]$
$\{< room\_dark = false >, < quiet = true >\}$

The enabling conditions are now both $< my\_room\_dark = true >$ and $< quiet = false >$, and so, both have to be true before the $book$ has any effect, in contrast to the two separate specifications (1) and (2) above, which takes effect on one or both situations happening. But in general, the designer should consider compatibility of multiple enabling situations and multiple goal situations.

2.2 Collections of Things

There has been much work on aggregating devices and artifacts for computational purposes (e.g., [6], [9], and e-gadgets[3]), including aggregating things such as grocery items on a kitchen table [11] and clothings in a wardrobe. Consider an electronic orchestra (or band) formed by a collection of vases, each vase contains a mini-computer and speakers to emit music corresponding to a particular type of instrument: we have one vase emitting guitar music, another five vases emitting violin music, another vase emitting piano music and another emitting bass music, but the vases wirelessly communicate with one another and are synchronized by a vase playing the role of a conductor. A user might buy a collection of such vases (containing at least the conductor vase) and then when the vases are set in close enough proximity to each other and the atmosphere is sufficiently quiet, they begin to play a musical piece. Given a collection of seven such vases $v_1, \ldots, v_7$, we can write a specification for this collection as follows, for some interpreter I, and the result is to play a piece by Mozart:

$\{< quiet = true >, < v_1, \ldots, v_7$ `in close proximity` $>$
$\}I[\![v_1, \ldots, v_7]\!]\{< Mozart\_playing = true >\}$

A system implementing this specification should be able to track the vases or sense their collocation and proximity with one another, and should also be able to detect the current sound level where the vases are. An easy variation is where appropriate vases are present so that a particular kind of music can be played (say, Bach), and say this requires only $v_1, \ldots, v_4$ to be present:

$\{< quiet = true >, < v_1, \ldots, v_4$ `in close proximity` $>$
$\}I[\![v_1, \ldots, v_4]\!]\{< Bach\_playing = true >\}$

An interesting issue that arises here is that enabling situation

$\{< quiet = true >, < v_1, \ldots, v_7$ `in close proximity` $>$
$\}$

subsumes the second enabling situation

$\{< quiet = true >, < v_1, \ldots, v_4$ `in close proximity` $>$
$\}$

so that two possible situations might be intended here. The system must then choose one of the musical pieces randomly, choose to satisfy the most specific situation, or the more general one - a policy is needed for such a case. Alternatively, if there is a user interface, the user can be asked for his/her choice. However, there would be situations where the resulting intended situation do not conflict and both can be carried out. For example, suppose we have "opening the window" instead of "playing Mozart" with the seven vases:

$\{< quiet = true >, < v_1, \ldots, v_7$ `in close proximity` $>$
$\}I[\![v_1, \ldots, v_7]\!]\{< open\_window = true >\}$

Then, both having Bach playing and an open window can co-occur. In general, given two specifications $s_1[\![C_1]\!]s_1'$ and $s_2[\![C_2]\!]s_2'$ where $C \cap C' \neq \emptyset$, a designer needs to consider disjoint, overlap, and subset relationships between enabling situations and situations to be achieved, i.e. $s_1 \cap s_2 = \emptyset$, $s_1 \cap s_2 \neq \emptyset$ (and if $s_1 \subseteq s_2$ or $s_2 \subseteq s_1$), and $s_1' \cap s_2' = \emptyset$, $s_1' \cap s_2' \neq \emptyset$ (and if $s_1' \subseteq s_2'$ or $s_2' \subseteq s_1'$).

2.3 Command-Controllable Things, Stateful Things, and Thing-Currying

Currying in functional programming refers to turning an n-argument function to a $(n-1)$-argument function. A thing that can receive commands (and not all things can do this) can be "curried" in that the thing itself may have a function, but the thing with a command issued on the thing (written like a method on an object) might have a slightly different function, and the thing with a command issued on the thing and additional parameters on the command might have yet another function.

$\{< room\_dark = true >\}, I[\![drapes.open]\!]\{< room\_dark = false >\}$

and an example with a parameter:

$\{< room\_noisy = false >\}, I[\![tv.set\_volume(high)]\!]$
$\{< room\_noisy = true >\}$

Things even if not command-controllable can be described with states. States of things can be described as part of an enabling situation, but could specify a curried version of things - we basically employ appropriate adjectives (written in bold) with things. For example, we might then have

$< quiet = false >, I[\![\mathbf{opened}\ book]\!] < quiet = true >$

Coupled one of the above specifications, we may have a book being brought into my room causing the light to come on, and when the book is opened, the radio in the room is turned off.

3. PROOF-OF-CONCEPT

As noted earlier, a thing (a) may have its own capabilities to achieve the situation changes, (b) may involve a human in the loop influenced by the thing to help it achieve its intent, or (c) the thing effectively uses other resources, or triggers another system to try to accomplish its intent. In other words, the thing might be stand alone and does not involve any surrounding infrastructure or it may connect to and utilize a surrounding infrastructure. The thing may or may not involve a user in its achieving of an intended situation. Below, we describe a prototype using Bluetooth discovery/scanning for Bluetooth tagged devices and a linear array of Phidgets[4] dis-

[3]http://www.extrovert-gadgets.net/

[4]http://www.phidgets.com

tance sensors, which allows thing identity (via Bluetooth addresses) and presence to be detected, and position with respect to the array of sensors to be detected.

Figure 1 illustrates the array of eight distance sensors (small boxes numbered 1 to 8) connected to a computer. The large rectangle represents an area where the position of an object (illustrated as a circle) can be detected. The large rectangle can be vertical (on a wall) or horizontal (on a floor, carpet, or table). A distance sensor can detect the distance of an object from it roughly up to 0.7metres away. In the figure, the object is a distance x metres away from

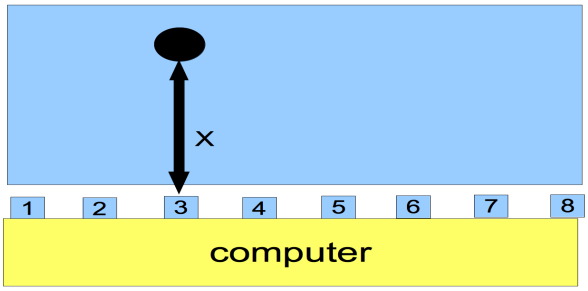

Figure 1: An illustration of our simple testbed of detecting the position of objects on a rough grid using an array of distance sensors numbered 1 to 8.

sensor 3, and so would have coordinates (3,x). Hence, the number of the sensors and the distance measures provide a rough grid for the object's position. It is possible that the object is detected by two sensors but we avoid that by having any two sensors more than a certain distance apart (determined by the dimensions of the object or thing) - but which also creates gaps in the detectable regions when the object is "between" two sensors. What is not shown in the diagram is that the object is Bluetooth-tagged (RFID can be used too) so that the computer can also sense the identity and presence of the object. We used this testbed to determine the current situation of the thing being tracked (presence/absence, position relative to our coordinate system), and output situations can range from displaying different content on a large screen to changing the music currently being played. We could have a specification such as the following for vase $v1$ on a fairly large table, with I denoting the system:
$\{< v1 \texttt{ at position } (3,0.5m) >\}, I[\![v1]\!]$
$\{< Mozart\_playing = true >\}$
And by simple Euclidean distance calculations on coordinates of two vases, we could have:
$\{< v1 \texttt{ and } v2 \texttt{ in close proximity} >\}, I[\![v1, v2]\!]$
$\{< Bach\_playing = true >\}$
The above specifications are general and can be used in other environments. For example, the way in which music is played in one environment is via computer speakers and another via sending commands to a hi-fi system, but the goal is to have Bart or Mozart played.

4. CONCLUSION AND FUTURE WORK

We have introduced ideas for specifying situation semantics for things, and illustrated our ideas with several examples. Such specifications, we assume, can then be executed by a system that will proceed through the six stages we outlined. We envision such a system being end-user programmable so that users can assign situation semantics to his/own own things, according to their own fancy. We think that the end-users, not only designers, should have a means to define the situation semantics of artifacts according to their own preferences, and create specifications in a standard XML language (which we have not defined in this paper) so that such specifications can be shared and reused.

We envision a generic interpreter which can execute these standard XML specifications, just like how a Web browser interprets standard HTML, e.g. user buys a vase and puts it in the living room and starts to see the vases' behaviour manifest, as specified. An ontology of situations and context ontologies (e.g., as reviewed in [14]) might be helpful to aid users in this regard. Descriptions of situations can then make use of context attributes from referenced ontologies.

Future work will extend the range of sensors used and define a collection of typical situation semantics specifications for everyday objects.

5. REFERENCES

[1] Jon Barwise and John Perry. *Situations and Attitudes*. MIT Press, Cambridge, MA, 1983.

[2] Keith Devlin. Situation theory and situation semantics. In John Woods and Dov M. Gabbay, editors, *Handbook of the History of Logic*, pages 601–664. 7 edition, 2006.

[3] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[4] 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.

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

[6] Andreas Heil, Mirko Knoll, and Torben Weis. The internet of things - context-based device federations. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 58, Washington, DC, USA, 2007. IEEE Computer Society.

[7] Ashraf Khalil and Kay Connelly. Context-aware configuration: A study on improving cell phone awareness. In *CONTEXT*, pages 197–209, 2005.

[8] Gerd Kortuem, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2010.

[9] Rajnish Kumar. User-centric framework for device aggregation. Master's thesis, 2003. http://scholarship.rice.edu/handle/1911/17600?show=full.

[10] Mike Kuniavsky. *Smart Things: Ubiquitous Computing User Experience Design*. Morgan Kaufmann, 2010.

[11] Marc Langheinrich, Friedemann Mattern, Kay RŽmer, and Harald Vogt. First steps towards an event-based infrastructure for smart things. In *Ubiquitous Computing Workshop (PACT 2000)*, Philadelphia, PA., October 2000.

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

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

[14] Juan Ye, Lorcan Coyle, Simon Dobson, and Paddy Nixon. Ontology-based models in pervasive computing systems. *Knowl. Eng. Rev.*, 22(4):315–347, 2007.