
TeC: end-user development of software systems for smart spaces

João P. Sousa*, Daniel Keathley, Mong Le,
Luan Pham, Daniel Ryan, Sneha Rohira,
Samuel Tryon and Sheri Williamson

Computer Science Department,
George Mason University,
4400 University Drive 4A5,
Fairfax, 22030 Virginia, USA
E-mail: jpsousa@cs.gmu.edu
E-mail: dkeathle@gmu.edu
E-mail: mle8@gmu.edu
E-mail: lpham6@gmu.edu
E-mail: dryan4@masonlive.gmu.edu
E-mail: srohira@gmu.edu
E-mail: stryon1@gmu.edu
E-mail: swillif@masonlive.gmu.edu

*Corresponding author

Abstract: This paper presents TeC, a framework for end-user design, deployment, and evolution of applications for smart spaces. This work is motivated by the current gap between traditional software development approaches and end user desire to easily personalise and evolve their systems for smart spaces. TeC is precise enough to support the fully automated deployment of systems designed by end users, and it addresses important characteristics of ubiquitous computing, namely, the ability to describe dynamic adaptations and to relate system features to physical location and to the presence and identity of users.

TeC is described by example, with four home automation systems concerning surveillance and energy management. The paper also discusses the implementation of the TeC middleware and preliminary evaluation concerning usability and engineering effort.

Keywords: design methodology; end-user development; smart buildings; home automation; situated computing, spaced-based computing.

Reference to this paper should be made as follows: Sousa, J.P., Keathley, D., Le, M., Pham, L., Ryan, D., Rohira, S., Tryon, S. and Williamson, S. (xxxx) 'TeC: end-user development of software systems for smart spaces', *Int. J. Space-Based and Situated Computing*, Vol. X, No. Y, pp.000–000.

Biographical notes: João P. Sousa received his PhD in Computer Science from Carnegie Mellon in 2005. Currently, he is an Assistant Professor with the Computer Science Department at George Mason University. His research interests include software architectures, usability, security, and self-* properties for ubiquitous computing, in which he worked for the past ten years.

Daniel Keathley is a graduate student who contributed to different components of the implementation and user studies as part of his coursework at George Mason University.

Mong Le is a graduate student who contributed to different components of the implementation and user studies as part of her coursework at George Mason University.

Luan Pham is a graduate student who contributed to different components of the implementation and user studies as part of his coursework at George Mason University.

Daniel Ryan is a graduate student who contributed to different components of the implementation and user studies as part of his coursework at George Mason University.

Sneha Rohira is a graduate student who contributed to different components of the implementation and user studies as part of her coursework at George Mason University.

Samuel Tryon is a graduate student who contributed to different components of the implementation and user studies as part of his coursework at George Mason University.

Sheri Williamson is a graduate student who contributed to different components of the implementation and user studies as part of her coursework at George Mason University.

1 Introduction

The pervasion of computing into the spaces where we live, work, and travel is challenging conventional wisdom about software development. Application domains of ubiquitous computing, also known as ubicomp, include energy management in buildings, transportation, assisted living, home automation and surveillance, etc. In addition to mobile devices, ubicomp encompasses the use of devices embedded or scattered in spaces ranging from homes to subway stations to streets and farms.

Traditional software development has relied on design methodologies that expect stakeholders to agree on a set of requirements for a system before it is implemented by software engineers. In contrast, ubicomp often targets open systems in informal domains where users come and go freely, each user may have a different set of expectations, and no central authority dictates the desired system behaviours.

Context-aware systems, with their adaptability to the surrounding circumstances, are an important step forward. However, also their adaptive context-aware behaviours need to be agreed on by stakeholders beforehand, and the difficulty to change those decisions after being cast into software causes end users to feel at the mercy, rather than in control of technology (Barkhuus and Dey, 2003).

In these domains, system requirements are too personalised, too specific to circumstances, and may change too often to make the approach of hiring engineers to make every change economically viable or even fast enough to be useful.

A promising approach is to empower end users to design and deploy their own personalised systems. Today, however, mainstream methods and tools for software development are inaccessible to end users. Developing the kinds of systems described above entails programming for network communications, concurrent threads, timeouts, exceptions, and adaptive behaviours, all of which are notoriously hard to use effectively, even by professional engineers. Furthermore, current software design methods are mute about physical location, making it impossible to relate desired behaviours and spatial context at the design level. Today, that relation is hardwired into the code, and into the way systems are deployed and administered.

This paper presents TeC, a framework for end-user design, deployment, and evolution of applications for ubicomp. TeC is meant for a range of end users, from home owners to domain experts, such as facility administrators and health-care professionals.

To become accessible to end users, TeC departs from an algorithmic view of computing in favour of a declarative

view similar to spreadsheets. In the latter, there is no algorithm or ‘main’ programme: all formulas are asynchronously recalculated whenever the values they refer to are updated. Similarly, computation and communications in TeC are triggered asynchronously and overall system behaviour is emergent.

TeC is precise enough to enable the fully automated deployment of systems. For that, it builds on prior work that addressed the formal operational semantics and the automated verification of desired behaviours (Sousa, 2010). It also includes constructs that address important characteristics of ubicomp, namely:

- a the situatedness of features in physical locations meaningful to end users, such as rooms and buildings
- b awareness of user presence and identity
- c dynamic adaptation of application features in response to application-level events.

Broadly, this work builds on principles of service orientation (SOA) and its supporting mechanisms, such as automated service discovery (Zhu et al., 2005).

The contributions in this paper include descriptions of

- a the TeC language, by example
- b editors for personal computer and smart phone platforms
- c middleware that facilitates system deployment and evolution
- d an empirical study that helped tune and obtain preliminary results on the framework’s usability.

In the remainder of this paper, Section 2 introduces TeC, while Sections 3 and 4 describe constructs to address space and personalisation of behaviours, respectively. Section 5 discusses implementation, including the integration of legacy players, space-aware discovery, dynamic deployment of teams, and security and privacy issues. Section 6 describes the evaluation and Section 7 compares TeC with related work. Section 8 describes future work and Section 9 summarises the main points of the paper.

2 TeC





A TeC system, called a *team*, consists of a collection of *players* with no central component responsible for coordinating each step of the action. Players are computing-enabled devices ranging from computers and smart phones, to smoke and motion detectors, to microwave ovens, clothes driers and smart power metres. A team’s

overall function results from the joint effect of the *activities* carried out by the individual players.

Activities generalise the concepts of process, computational service, and function of a device. Specifically, activities may be of short duration with a discrete output, like service invocation; they may be long-lasting with a succession of inputs and outputs, like processes; or they may last for months without producing an output, like the function of a smoke detector.

End users may design different teams to serve purposes such as ‘surveillance’, or ‘energy management’. A *team design* includes *activity sheets*, describing the role of each player in the team, and *communication* paths for channelling asynchronous messages and data streaming between players (more below).

Deploying a team consists of two operations: *discovery* and *briefing*. Concrete players for carrying out the activities in a team are discovered at run time by the TeC middleware, which then *briefs* the players about their role in the team. Once briefed, players interact with their environment and with each other with no further intermediation or central coordination.

Figure 1 illustrates a team design for surveilling the perimeter at a small farm belonging to a hypothetical end user, Anne. The team overview, on the top left, shows three activities: **monitor fence**, **film** and **phone**. The communication paths link *output* events to *input* events, respectively shown as  and , and output to input data streams, shown as  and . The figure also shows select details for each of the activity sheets.

Team designs are specified by end users by interacting with an editor (more in Section 5). If at any point Anne is not happy with the way the team in Figure 1 is operating, she might add more activities, rewire the routing of messages and data among players, or change the entries in

the activity sheets, which support a level of syntactical complexity similar to the formulas in a spreadsheet.

2.1 Activities

To add an activity to a team, end users customise an activity *sheet* starting from an activity *type*. Typically, activity types would result from a community standardisation effort and would be downloadable or shipped with TeC-enabled devices.

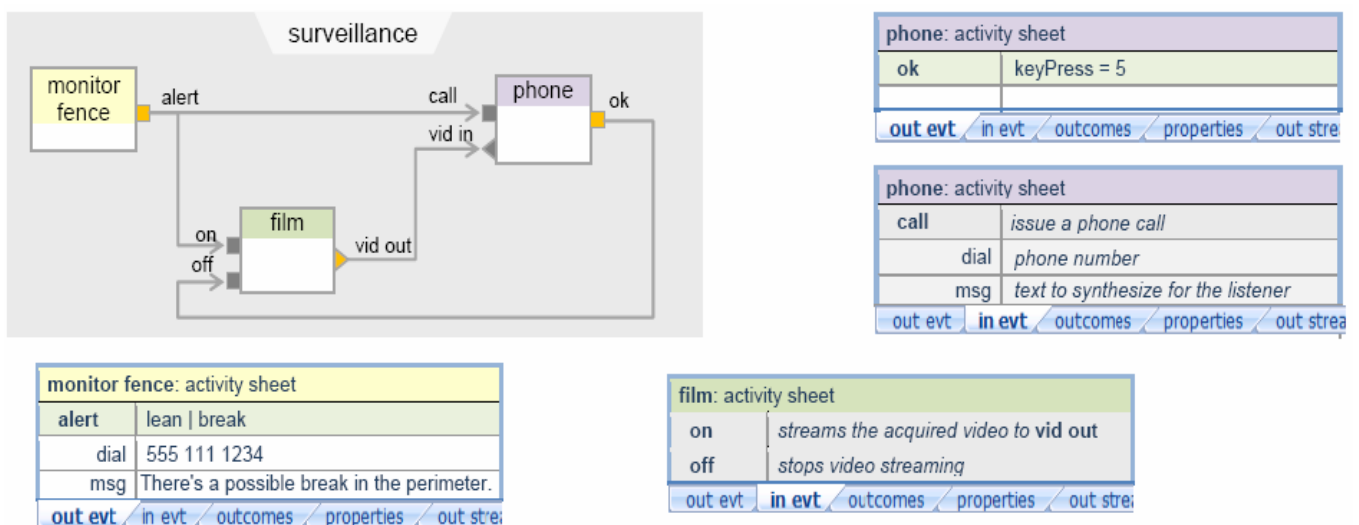
Activity types define *input events*, *data streams*, and internal *outcomes*: property values that may change during operation. For example, whether a motion sensor has been tripped, a Boolean outcome, or the temperature sensed by a thermometer, a numeric outcome.

When Anne added the film activity in the design shown in Figure 1, descriptions for the input events **on** and **off** and for output stream **vid out** are automatically copied from the activity type. These correspond to features that are intrinsic to a filming device, and are shown in the activity sheet but cannot be changed by Anne.

The tab **outcomes** (not shown for the sake of space) in the sheet for **monitor fence** includes descriptions for Boolean outcomes **lean** and **break**. As above, these descriptions are copied from the type and cannot be changed by Anne.

Users customise activity sheets by defining *output events*, the conditions that *trigger* such events, and message *payload*. In the **out evt** tab for **monitor fence**, Anne defined event **alert** and its triggering condition **lean | break**. Payload attributes are shown in rows with a name on the left and a value on the right. In the example, Anne added two payload attributes **dial** and **msg** that will travel with the message issued when the event occurs (more on communication below).

Figure 1 Design of a team for surveilling the perimeter at Anne’s small farm (see online version for colours)



Notes: The fence is equipped with sensors that detect animals leaning against the wire, or breaking it. If that happens, a call is issued to Anne’s cell phone, streaming video captured from the fence so that the situation can be assessed. Anne may press key 5 on her phone, indicating that no further action is required from the system.

Anne also bought a TeC-enabled phone at a local store, which came with an activity type describing input events such as **call**, input data streams such as **vid in**, and outcomes such as recognising key presses by analysing tones on the line. Anne customised the sheet with output event **ok**, to be triggered when outcome **keyPress** takes the value 5.

Sheet customisation may include defining new input events in addition to those that come with the activity type. The observation of such events and the values of their payloads may then be used anywhere in the **out evt** tab.

Defining new input events is especially useful when users wish to include a *generic* activity in a team. Unlike specialised activity types, the generic type includes no definitions of input events, streams, or outcomes, and may be played by any computing-enabled device.

The team in Figure 2 includes a generic activity **make calls** which facilitates making calls to different destinations depending on circumstances and in response to alerts issued by any sensor *s*. Anne defined two input events for **make calls**, **fence** and **handled**, and made an explanatory note for **handled** but not for **fence**. In the example, output event **call Anne** is triggered as soon as **fence** is observed.

2.2 Triggering output events

Event **call 911** in Figure 2 illustrates a less trivial triggering condition, meant to wait a short while for user reassurance, and lacking that, issue a call to the emergency number.

Triggering conditions may include the operators:

- $P(e, t)$, that *postpones* the validity of e by time t
- $T(e_1, e_2)$, that *toggles* between true and false as e_1 and e_2 become true, respectively

- $R(t)$, that becomes valid with e and then *rests* for time t , ignoring e until that time is elapsed.

In Figure 2, $P(\text{fence}, 0:01)$ sets a one minute time out after a message is received from the fence, and $T(\text{fence}, \text{handled})$ disables the reaction to the fence alert by becoming false once **handled** is observed.

The *R* operator can be used in monitor fence to reduce repeated alerts, for example, due to a horse repeatedly leaning against the fence for scratching its hindquarters. Specifically, by making **lean | break R(0:02)** the trigger condition for **alert**, a sensor will refrain from issuing further alerts for two minutes after being activated the first time.

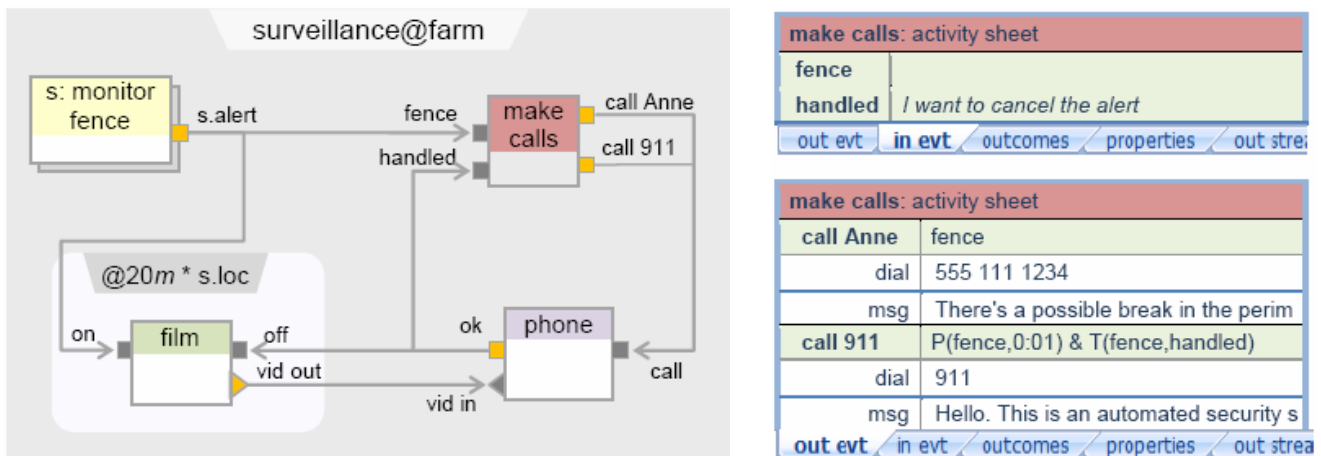
In general, triggers are Boolean expressions that may include logical operators and, **&** or **|**, not, **!**, as well as predicates on values using comparison operators such as equals, greater than, etc. For example, if **tempF** is an outcome of a temperature sensor measured in degrees Fahrenheit, then **tempF > 50 & tempF < 80** is a valid expression. The name of an input event is an expression that *becomes* true each time the event is observed.

Because output events occur at discrete points in time while conditions may remain true for a long time, an event occurs when its trigger *becomes* true. The trigger needs to become false before the event can occur again.

2.3 Communication

The occurrence of an output event causes a message to be sent along the communication path(s) attached to it, and in turn that will cause the occurrence of the input event(s) at the end of the path(s). In Figure 1, the occurrence of event **ok** on **phone** causes (sending a message whose reception causes) event **off** on **filming**. Event **alert** causes *both* **call** on **phone** and **on** on **film**.

Figure 2 End user Anne redesigned the team in Figure 1 to support a large fence with multiple sensors and multiple cameras (see online version for colours)



Notes: When a sensor is activated, a camera within 20 metres of that sensor is turned on. Also, a call to 911 will be issued if Anne cannot be reached or does not handle the alert.

To keep matters simple for end users, paths may connect events freely and the matching of attributes in the payload is best effort, based on attribute name. In Figure 1, attributes `dial` and `msg` on the outgoing message are matched to attributes `dial` and `msg` on input event `call`. Since on has no attributes, the payload of the message is discarded in `film` upon triggering the input event.

While best effort matching saves users from the tedious work required by method invocation in mainstream languages, it may lead to unintended errors. To help users manage this tradeoff, upon user request, the editor highlights which attributes are being matched in a connection. Additionally, players should have reasonable defaults to all attributes of input events.

3 Space

Teams may be widely distributed, with some players deployed in one space, such as Anne's farm, and others somewhere else, such as her friend Bob's house, or from Anne's point of view "where I am now".

To make it easy for end users to design and deploy distributed teams, TeC combines automated discovery of players with explicit spatial constraints in team design. Such constraints are used to guide the discovery mechanisms, i.e., to *scope* discovery to the spaces of interest for the user. For that, all players are made aware of their location (more in Section 5) and the TeC language defines the property `loc` for all entities.

Spatial constraints are indicated in the form `name @ space`, to be read `name at space`, and may show in one of two ways. First, constraints matching human-perceived boundaries, such as a room, building, or piece of property. These are shown in the team design by a shaded rectangle enclosing the activities that take place at the named space. All activities in Figure 2 take place within the `farm`.

Second, a range around a location of interest. These are indicated by a shaded area with round corners labelled `name @ radius * location`, to be read, `name at radius around location`. In the figure, `film` takes place within a 20 metre range of the activated sensor, `s`.

Location in team design is specified either by reference to the location of a known entity, e.g., `s.loc` or `Anne.loc`, or by indicating a human-readable address of the form `city/street-address/room`. For example, `Alice-Springs.NT-0870.au/456.windy-road/kitchen.2` indicates the kitchen in Apartment 2 at 456 Windy Rd in Alice Springs. Users may define aliases for a location, such as `farm` (specific address not shown here), or a list of locations, such as `trusted spaces`, listing the addresses where a user is comfortable deploying his or her teams. The discovery mechanisms described in Section 5 are capable of reasoning about containment and proximity of such location expressions.

In Figure 2, the activity `monitor fence` is carried out by *all* the players (fence sensors) found within the boundaries of the `farm`. In TeC, finding *one* player to carry out an activity is indicated by a single box, and finding *all* such

players is indicated by stacked boxes. In the latter case, a label precedes the activity name and output events emanating from the stacked boxes, e.g., `s` in `s:monitor fence` and `s.alert`. This label refers to the specific player emitting the event and can be used elsewhere in the design. In the example, `s` is used to constrain the discovery of a camera to a vicinity of `s.loc`.

The set of players carrying out an activity described by a stacked box may change dynamically as players enter or leave the prescribed space. For example, if Anne buys extra sensors of type `monitor fence` and deploys them at the farm, they will be automatically discovered and instantaneously incorporated into the `surveillance` team, since that is the intent expressed in the team design.

The example in Figure 2 illustrates

- the placement of teams within spaces, with players being discovered both within a fixed location and at a vicinity that follows the occurrence of events at run time
- incorporating all players within a space that can perform that activity
- the specification of timeouts.

4 Personalised behaviours

An important feature of teams is the ability to adopt different configurations and behaviours depending on which users are present. For example, suppose that a hypothetical user, Bob, installed a smart power metre at his home that issues pricing signals reflecting the load on the power grid (<http://www.smartmeters.com/>). To reduce the energy bill, Bob would like the clothes dryer, a heavy energy consumer, to pause drying during peak rates and to automatically resume during lower rates.

Figure 3 shows Bob's initial design to save energy at Bob's house (Bob's alias for his home address). The team includes two unnamed sub teams: one at the laundry room, and the other at the electric cabinet. Operator in the space constraint for the sub teams refers to the space of the enclosing team: `Bob's house`.

However, Bob's wife Mary is not happy with the new feature, since she would like to have her drying cycles run uninterrupted. To accommodate that, Bob changed his design to Figure 4. Now, the sub team in the laundry room includes activity `track(Bob)`.

Figure 3 When the smart metre detects rate changes, it informs the clothes dryer, which pauses drying at peak rates (see online version for colours)

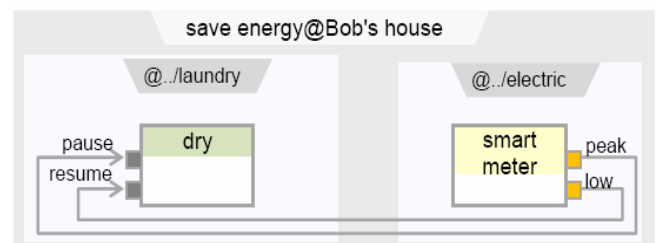
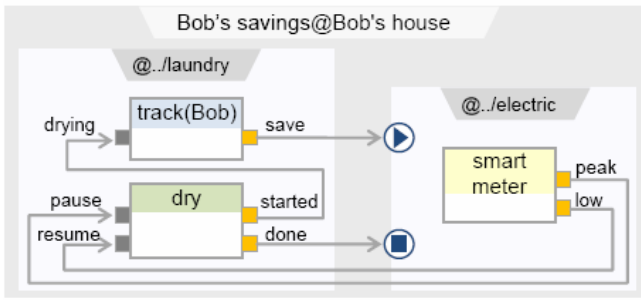


Figure 4 If Bob is present in the laundry room when the dryer is started, the smart metre joins the team and sends pricing signals to the dryer (see online version for colours)

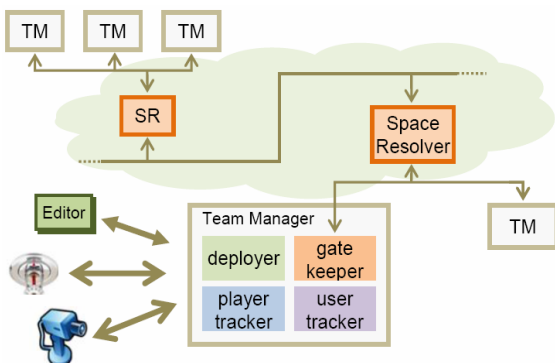


Activities of type *track* take as a parameter the id of the entity to track and may use a variety of means to do so (Hightower and Borriello, 2001; Covington et al., 2002; Asmidar and Jais, 2009). In the sheet for *track(Bob)*, not shown for the sake of space, the trigger for event *save* is set to *T(enter, leave) & drying*, which captures the intention to issue event *save* if the dryer is started while Bob is in the laundry room. Outcomes *enter* and *leave* are defined in the *track* type, and correspond to the tracked entity entering and leaving the space where the *track* activity is deployed. Operator *T* works as described in Section 2.

Events *save* and *done* are used to control, respectively, starting \blacktriangleright and stopping \blacksquare the sub team at the electric cabinet. A sub team *without* such operators is deployed and retired with the enclosing team, and ultimately at the users request. A sub team with these operators is deployed once an event reaches \blacktriangleright , and retired either with the enclosing team, or when an event reaches \blacksquare . In the example, the electric sub team is deployed once *save* is announced: the metre is then briefed for activity *smart metre* and starts sending *peak* and *low* events to the dryer. Whenever this sub team is stopped, the smart metre device keeps metering and possibly participating in other teams: it is just not part of *save energy*.

This example illustrates features to detect the presence and identify specific users in a space, and the inclusion of rules in a team design to automatically start and stop sub teams. These two constructs were used to design a team that personalises its behaviour at run time, depending on who is present in a space.

Figure 5 Informal picture of the TeC infrastructure (see online version for colours)



5 Implementation

In addition to the players, which take centre stage, support for TeC relies on three kinds of components: *editors* of team designs, *team managers* (TM) present at each space, and *space resolvers* (SR) deployed throughout the internet: see Figure 5. Having infrastructural components such as TMs and SRs, as opposed to adopting a purely peer-to-peer approach, makes it easier for end users to control the participation of players in privately owned spaces.

- Editors enable end users to edit team designs, and include commands to deploy and retire a team.
- TM oversee the set of players within a geographical area and may coordinate the deployment of distributed teams with other TMs. Their role is divided into:
 - 1 player tracker registers players and monitors their physical location and performance
 - 2 deployer facilitates deploying, updating, and retiring teams, briefing players as needed
 - 3 user tracker identifies users entering and leaving the space, keeping track of their location within the space
 - 4 gatekeeper facilitates communicating with other TMs while controlling access to the space as well as the release of any information to the outside.
- SR map space constraints, such as specified in teams designs, to the TMs that will help discover the desired players. For this role, SRs take into account security and privacy policies transmitted by gatekeepers.

The communication between distributed components, including the communication of events between the players in a team, is carried in XML over the network transport (currently TCP/IP).

5.1 Players and activities

For becoming TeC-enabled, players need the computational capability to interpret activity sheets, following the operational semantics in Sousa (2010), and the ability to communicate over the network.

Two kinds of players can be distinguished: players that require specialised devices, such as motion detectors, and those which can be realised by generic computing devices, such as *make calls* in Figure 2. For players that require specialised hardware, we are investigating the use of low-power system-on-chip, (e.g., http://www.gainspan.com/products/GS1011_single_chip.php), although we have done functionally equivalent prototypes using inexpensive sensors with device drivers running on a computer (<http://www.phidgets.com/>).

Turning such devices into TeC players consists of wrapping their device drivers with code that implements the TeC protocols (more below). These wrappers have the ability to run multiple activity sheets simultaneously; for example, a smart metre may participate in several teams,

each with its own conditions to trigger different output events (e.g., Figure 4).

Players deployed at fixed locations, such as smoke detectors and smart metres, have their location set with the TM during deployment. Players that run in mobile devices such as cell phones leverage the device's own location awareness mechanisms, such as GPS, and send updates to the TM every time their location changes past a radius set during deployment.

5.2 Editors

An initial implementation of an editor works over the Eclipse platform (<http://www.eclipse.org/modeling/gmp/>) and can be used to produce team designs with the core features shown in this paper. An additional implementation was developed over Android™ (<http://www.android.com/>), which is shown in Figures 6 and 7.

Several instances of editors may run on different computers/phones within a space: each editor is registered with the local TM following a protocol similar to registering players (Section 5.4). Team designs reside with the editors, and are shared with TMs for deployment purposes. Editing team designs is supported while a team is deployed, and the team is re-briefed with the changes upon explicit user indication.

To help users design their teams, editors proactively query TMs for the players available within a space and the activity types they support: exploratory discovery.

5.3 Space-aware discovery

Discovery requests have type constraints, either a specific activity type or *any*, and spatial constraints. For example, Anne may want to know about all monitor fence sensors

deployed within her farm, and Bob may want to know about all players of any type deployed at his house. In addition to exploratory queries for editing purposes, the TM confirms player availability during team deployment.

Specifically, the deployer (in the TM) examines the spatial constraints for the team and directs discovery requests to either the local player tracker or to the gatekeeper. A team employs local players by default, e.g., Figure 1, or when a constraint refers to a part of the local space, e.g., subteams in Figures 2 through 4.

Figure 6 Screenshots of the Android editor, (a) part way through designing the team in Figure 1, and (b) associating message payload to event alert (see online version for colours)

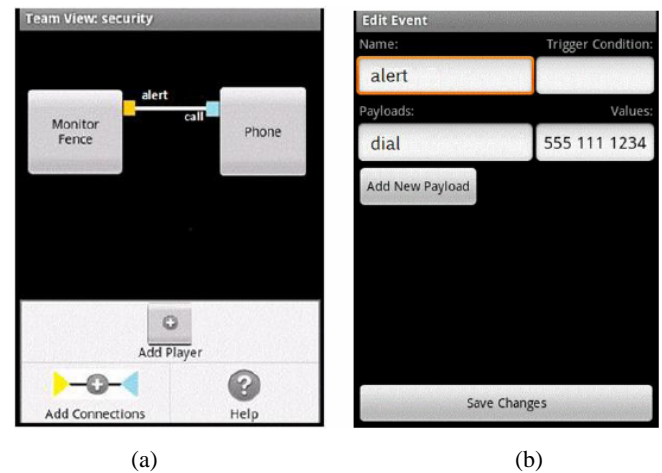


Figure 7 Continuing the design in Figure 6, (a) adding the film activity (b) adding a data stream between film and phone and (c) connecting the ends of the stream to the data ports (see online version for colours)



Discovery may be done *after* team deployment, when spatial constraints contain a variable, e.g., s in $@20m * s.loc$ in Figure 2. For simplicity, we do not distinguish the case where s refers to one of a set of fixed players, from the case where s is mobile. Both cases are handled by having the TM broker the delivery of events. In the example, the activity sheets for **monitor fence** instruct the sensors to send **alert** events to the TM. Upon receiving an **alert** from a player s , the TM discovers a player for **film** that satisfies the location constraint, briefs it, and then relays the event. Additionally, when asked to discover *all* players for an activity within a space, e.g., **monitor fence**, the deployer registers a callback with the player tracker for being notified when a player for that activity enters the space.

In its simplest form, discovery of *one* player within a space results in an arbitrary player capable of carrying out the desired activity. Frequently, there will be only one such player, e.g., a smart metre in the electric cabinet. However, there are also cases where the user is not best served by an arbitrary choice among several candidates. In the example above, the choice of a camera $@20m * s.loc$ might be refined with the notion that *closer* to s is better. More general optimality criteria might also include desired features and quality of service preferences, e.g., resolution or colour. Prior work by the first author included sophisticated discovery criteria (Sousa et al., 2006, 2009), and we are currently investigating how to extend TeC with optional optimality criteria, without losing its intended simplicity of use.

When the spatial constraints for an activity extend beyond the local space, the deployer directs discovery requests to the gatekeeper. The gatekeeper relays the request to a SR, which, akin to a name server in internet's domain name system (DNS), identifies and relays the request to the one or more TMs that satisfy both the spatial constraints and mutual trust policies.

5.4 Security and trust

Security and trust are addressed at two levels:

- a within a space
- b across spaces, for purposes of team deployment and interaction.

Protection within a space starts with player deployment (Kawsar et al., 2008). Users control the admission of players into a space by explicitly swiping an RFID tag associated with each player by a reader attached to the TM. The tag contains a public key for the player, which is used by the player tracker to initiate communication and set up a secure channel between the player and the TM. Fresh symmetric keys are generated by the deployer for communication within each team, which are passed during briefing.

In addition to players, the identity of the users themselves needs to be verified by the space. Doing so unobtrusively is a separate and active area of research (Sabzevar and Sousa, 2011). For simplicity, the current prototype uses RFID technology (Sousa et al., 2005).

Protection across spaces is coordinated by the gatekeeper. For example, suppose that Anne is visiting her friend Bob, consults him about the team design, and then decides to deploy the team in Figure 2. Because that team includes activities at the farm, a discovery request is directed by the TM at Bob's home to the TM at the farm (facilitated by SRs, as described in 5.3).

In this example, the security issues are

- 1 the TM at Bob's home becomes aware that Anne is deploying a team at her farm
- 2 the TM at the farm becomes aware of Anne's presence at Bob's
- 3 the TMs at both spaces become aware of Anne's design
- 4 the TM at the farm needs to recognise Anne and grant the request.

Issues (1–3) could be alleviated if an editor running on Anne's phone were to communicate directly with the TM at the farm, for example, via a cellular telephony link secured using previously shared encryption keys. The flip side of this security strategy is that users have to become aware of, and manage information release to remote TMs vs. the local TM. This may become complex when users intend to leverage local players, or to deploy teams with a mix of local and remote players.

To make it simpler for end users to manage security, editors currently register with the local TM at a space and rely on it to facilitate all discovery and deployment requests. The flip side of this strategy is that users need to decide whether to trust a space before editing/deploying their team designs at that space.

With either security strategy, the fourth issue concerns Anne's identification and access control at the farm. To facilitate that, the editor identifies the request as coming from Anne and signs it using her (private key) credentials. If Anne is known to the farm and authorised to deploy teams, which she is, the gatekeeper at the farm relays the request to the player tracker and subsequent briefings to the players themselves.

5.5 Briefing

The deployer briefs each player during team deployment, and may do so again, if the user subsequently changes the team design and requests the changes to come into effect.

Figure 8 shows the briefing transmitted to **monitor fence** in Figure 1, which includes the activity sheet and communication paths that start or end in **monitor fence**. In this example, the briefing includes the specification of output event **alert**, with expressions for its trigger and payload, and the communication paths to events **call** and **on**. Specifically, each target element identifies the IP address, port, and symmetric encryption key to use in that channel (elided in the figure). While in transit, this briefing is itself encrypted with the symmetric key shared between the player for **monitor fence** and the TM.

Figure 8 (a) Briefing for the activity sheet in Figure 1 and
(b) message sent towards phone when lean | break is
satisfied

```
<activity-sheet activity="monitor fence">
  <out-evt>
    <evt name="alert" trigger="lean | break">
      <att name="dial" value="555 111 1234"/>
      <att name="msg"
        value="There's a possible break in the
        perimeter. "/>
      <target in-evt="call" ip="phone ip" port="*****"
        key="*****"/>
      <target in-evt="on" ip="film ip" port="*****"
        key="*****"/>
    </evt>
  </out-evt>
</out-stream/> <in-evt/> <in-stream/>
</activity-sheet>
```



(a)

```
<evt name="call">
  <att name="dial" value="555 111 1234"/>
  <att name="msg" value="There's a possible break in the
  perimeter."/>
</evt>
```

(b)

The briefing protocol described above supports stateless activities, such as the ones throughout the examples in this paper. Or to be more precise, activities which state can be discarded when the team is retired and when the activity passes to a different player during deployment (e.g., film in Figure 2). Should it become necessary to preserve an activity's state in such circumstances, then this briefing protocol could be extended so that the TM facilitates the transmission of state snapshots between players (Sousa et al., 2005; Sousa, 2005).

5.6 Dynamic deployment and adaptation

For team designs that include rules for starting  and stopping  sub teams, e.g., Figure 4, the discovery and briefing of players may be done after a team is deployed. Similarly to the case of space constraints with variables discussed in Section 5.3, the TM registers to receive the events leading to the start and stop operators, and reacts to the reception of those by deploying or retiring the sub team, as requested.

As in prior work (Sousa et al., 2006), the TM could be extended to play an active role in fault tolerance and adaptation to performance degradation of players. However, the design herein reflects the goal of minimising the role of infrastructural components after the team is deployed. Currently, the TM only intervenes during deployment and retirement of a team/subteam.

6 Evaluation

The evaluation of this work has so far covered the following two aspects: first, the usability of the editor, and second, the effort associated with bootstrapping TeC towards building real systems for a smart home. As the implementations mature, more encompassing evaluations will be carried out (Section 8).

6.1 Usability

The usability of the Android editor was evaluated in two rounds of user studies.

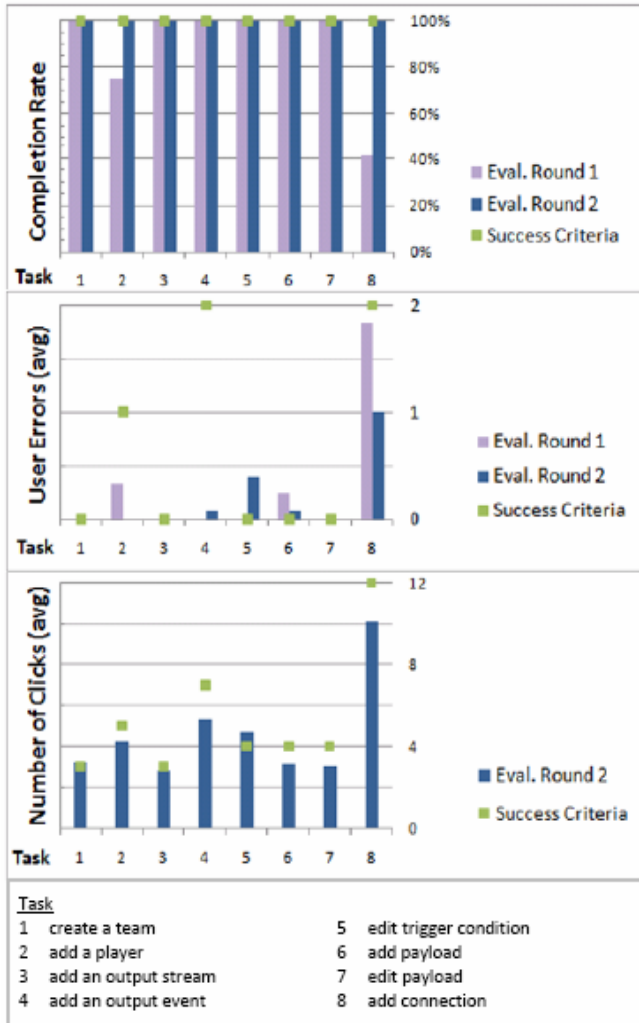
The first round emphasised *formative evaluation*: it was carried out in the early stages of development, with a prototype. The study population consisted of 12 volunteers among the graduate students taking a user interface design course. These students were totally unfamiliar with TeC, and after an explanation of five minutes were asked to carry out concrete tasks concerning the creation and modification of team designs. Users performed these tasks unassisted.

Feedback from users was helpful in identifying unclear aspects and awkward interactions, which led to some redesign and polishing of the interface. The interface in Figures 6 and 7 results from these changes.

The second round took place a few weeks later, with the same population but now with the polished interface. This round emphasised usability metrics defined for each concrete task, including *completion rate* (percentage of successful attempts at completing tasks), average number of *user errors* (users taking an unintended action), and average *number of clicks*, i.e., screen touches, to complete each task. The evaluators set *success criteria* for each metric based on their understanding of what would be acceptable by users, and according to the complexity of each task.

The measurements for eight simple tasks are shown in Figure 9. The ideal completion rate for tasks is 100%, corresponding to users always succeeding in finishing what they set out to do. Although the rates for tasks 2, 75%, and 8, 42%, had been significantly below this goal during the formative evaluation, they were brought all the way up to 100% with the revised interface.

The ideal number of user errors would be zero, corresponding to users always knowing exactly what to do to accomplish their intent. However, this ideal is unlikely to be attainable by absolute beginner users such as those in these studies. Therefore, the evaluators set non-zero criteria for less trivial tasks such as 2, 4, and 8. Adding a connection, task 8, had the highest number of user errors due to users having trouble locating the command. The redesigned interface in round 2 reduced the trouble on this task by almost half.

Figure 9 Results of users studies (see online version for colours)

User difficulty on editing trigger conditions, task 5, came as a surprise: users became confused by the popping up of the on-screen keyboard as soon as they clicked the editable field. Because the appearance of the keyboard reshuffled the screen layout, users could only see all the fields again by hitting the back button to remove the soft keyboard. This problem also raised the number of clicks for task 5.

Ideally, the average number of clicks to complete a task would be below a threshold that reflects the task's complexity. Except for task 5, due to the issue above, all others came on or below this threshold.

In addition to quantitative metrics, users were also surveyed with questions honed to identify points for improvement and preferred designs among alternatives. The answers to these questions already proved useful in making the improvements for round 2. After round 2, 83% of participants still indicated task 8 as the most troublesome. The interface design for tasks 5 and 8 is being revised for the next round of evaluations.

6.2 Building systems

Once a variety of TeC-enabled devices, i.e., players, are available, the TeC middleware enables users to easily

assemble home automation systems to their own desires. To assess the engineering effort associated with making this possible, we set off to develop a number of players and demonstrate their use in working applications.

We wrapped inexpensive devices for:

- motion detection, using infrared
- light sensing, which will detect lights being turned on
- touch sensing, which can be attached to commonly stolen objects such as TV sets
- filming, using a USB camera
- user identification, using RFID technology.

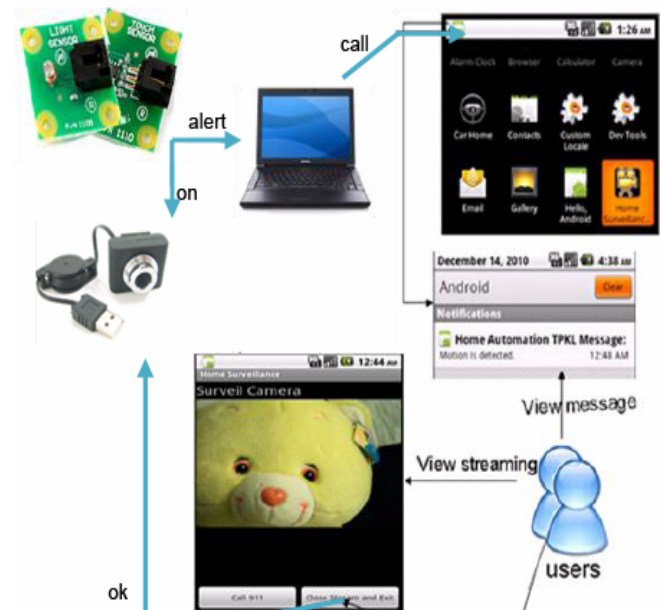
We also built emulators running on a PC for:

- smart electric metres, which announce energy rates
- thermostats, to control house heating and cooling
- laundry dryers.

Additionally,

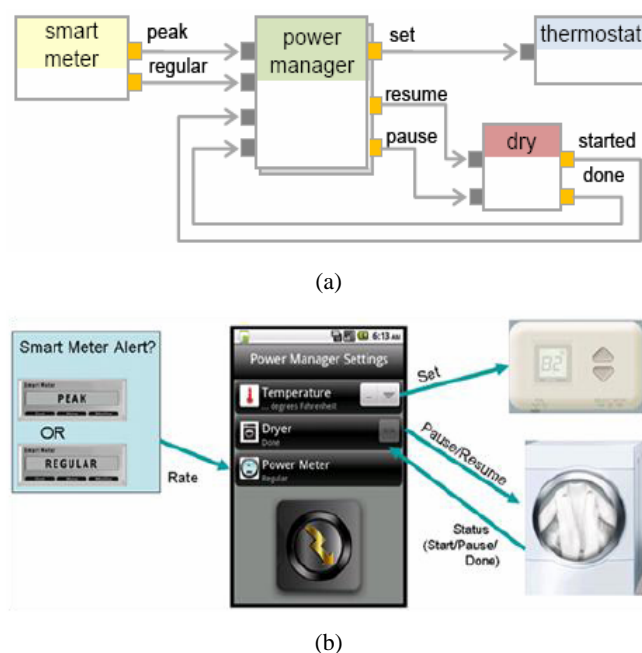
- we turned Android phones into players by developing a small application that receives call notifications and video streams (see *phone* in Figures 1 and 2)
- built a *power manager* player that runs on Android and that enables users to monitor and complement automated energy management behaviours such as the ones in Figure 4.

These players were demonstrated in the applications shown in Figures 10 and 11.

Figure 10 Home surveillance system following the design in Figure 2, except that monitor fence sensors were replaced by three other kinds: touch, light, and infrared/motion (see online version for colours)

Note: Activity make calls runs on the PC's TeC middleware.

Figure 11 Energy management system: design (a) and user interfaces (b) (see online version for colours)



Notes: Multiple users may run instances of power manager on their phones, thus sharing status messages and concurrently controlling appliances. The thermostat supports a request averaging policy, while the drier recognises user priority.

The most significant implementation effort concerning the players was the communications library for messages and, especially, video streaming. We investigated several off-the-shelf products to stream video, but these turned out to be too unwieldy for our simple purposes. We ended up writing a simple piece of code that captures and sends a sequence of still images over regular TCP sockets, and that has a much smaller memory footprint and better frame rate and jitter than the products we had tried.

Concerning messages, the communications library offers a simple API to send and receive messages, taking care of parsing the XML for both briefings and application events. Parsing was implemented over the packages supported by different Java platforms: XMLBeans for Java standard edition, and XMLSpy for Android.

Once the communications library was in place, the effort of implementing each player by wrapping the corresponding device driver was measured in hours. For those players that include a user interface, that effort was proportional to the desired sophistication.

7 Related work

Other work has addressed end-user development of ubicomp systems. We distinguish two groups of contributions. A first group offers languages to interconnect devices in a smart home. These languages range from written sentences with a very restricted English vocabulary, InterPlay (Messer et al., 2006), to graphical metaphors such as jigsaw puzzles, Jigsaw (Humble et al., 2003), to specific

diagrammatic languages, PIP (Chin et al., 2006), to tangible interfaces using physical cubes and RFID proxies for the devices themselves, respectively AutoHan (Blackwell and Hague, 2001) and FedNet (Kawsar et al., 2008). A second group of contributions adds language constructs to describe contextual conditions. These conditions are expressed either as restricted English sentences, CAMP (Truong et al., 2004), as if-then rules expressed using a succession of menus on a cell phone, HYP (Barkhuus and Vallgård, 2003), as orchestration rules over a taxonomy of features and context, Pantagruel (Drey et al., 2009), or using a diagrammatic notation to express spatial co-location and temporal order, iCAP (Sohn and Dey, 2003).

Concerning the choice of medium for the language, TeC is closer to the contributions that employ a diagrammatic language (Humble et al., 2003; Chin et al., 2006). Diagrammatic languages are considerably more expressive than tangible languages and are more precise than restricted sentences in English (Ko and Myers, 2004). While other diagrammatic languages support connecting existing ports in devices, TeC supports a spreadsheet-like metaphor that allows end users to define new ports, i.e., events and data streams, conditions to be monitored and transmitted payload. Concerning context-awareness, TeC's expressiveness is closer to iCAP, and richer than the non-diagrammatic languages (Truong et al., 2004; Barkhuus and Vallgård, 2003). While iCAP allows the expression of rules where a condition triggers an action, TeC additionally allows the easy expression of rules where a contextual condition triggers the deployment or reconfiguration of an entire team, with its set of features and behaviours. TeC goes further than existing work in supporting the design of systems distributed across multiple spaces, all in a secure way.

The implementation of TeC builds on lessons learned in other software infrastructures for ubicomp, namely the authors' prior work in Project Aura (Sousa et al., 2006, 2008; Sousa, 2005), and others (Román et al., 2002; Ponnekanti et al., 2001). An especially relevant area is location-aware service discovery. Broadcast-based discovery addresses location by administratively configuring network routers and bridges to limit the broadcast of service requests to a network partition, e.g., (Chakraborty et al., 2006). This becomes awkward when the intended scope of discovery does not match a particular network partition.

Work in directory-based discovery adopts a representation of location based on attribute-value pairs (Adjie-Winoto et al., 1999; IETF, 1999; Campo and Garcia-Rubio, 2006), possibly complemented by indicating a specific directory host (Waldo, 2000; Raverdy et al., 2006), or hierarchy of hosts (Czerwinski et al., 1999), to further scope the discovery. However, attribute-value matching in the discovery mechanism is purely syntactic and any reasoning about proximity or containment is left to the application code. In contrast, the TeC discovery infrastructure understands physical location at a semantic level, and is able to reason about proximity and containment

among two intermixable representations: human-readable physical addresses and coordinates generated by systems such as GPS.

8 Future work

The work so far demonstrated prototypes of the several components in Figure 5, which, with small amounts of computer-scientist-in-the-loop, came together in systems such as illustrated in Figures 10 and 11.

Ongoing work concentrates on taking the integration of these components to a level of maturity that enables end users to discover players, and develop, deploy, change, and redeploy their applications assisted only by TeC's automated tools. Chief among those enhancements are

- a have players interpret any activity sheet that may be produced by a user interacting with the editor, as opposed to fairly application-specific sheets
- b fully integrate automated player discovery and the advanced features illustrated in Figure 2 and Figure 4.

Once this tool maturity is reached, evaluation of the overall usability of the TeC concepts will be carried out via hands-on user studies covering the entire end-user development lifecycle. Such maturity is necessary since the tools need to encourage users towards exploratory behaviours: flexible home automation of the kind envisioned here opens a new domain to a population of users unfamiliar with its possibilities.

Future enhancements to the TeC language include relating desired features to human-perceived time, such as 'at night' or 'in the summer', in addition to the relations already supported between features and space and human presence (e.g., Figures 2 and 4).

9 Conclusions

The expansion of computing into physical infrastructures, such as buildings and energy distribution, gives rise to technical and usability challenges. Chief among those challenges is the need to match user expectations, under penalty of rejection and circumvention of the very features designed to protect users and improve their quality of life.

Conventional application development by professional engineers collapses under the economics of the problem. No matter how sophisticated, no single solution will satisfy all users in all circumstances. Ideally, each end user would design a personalised solution, and evolve it as he understands and changes his mind about the required features and behaviours.

TeC offers an end-user design language for the automation of smart spaces. The four systems presented in this paper illustrate the features of TeC and its applicability to surveillance and energy management. This paper illustrated two concrete syntaxes for TeC, supported by editors for two different platforms: personal computers and smart phones.

Irrespective of the concrete syntax, the key traits of TeC include a declarative semantics akin to spreadsheets for describing the roles of distributed autonomous players. The activities of players coalesce into a team behaviour by means of exchanging asynchronous messages triggered by user-defined conditions. Typical features of devices such as sensors and actuators are captured in predefined activity types, and system-specific features are customised by end users and captured in activity sheets.

The relationship between an activity sheet and a player is akin to the relationship in SOA between a service description and a service provider: an activity sheet describes what needs to be done and a player provides the means to do it. Advantages of this separation include the ability to deploy the same team design at different locations, with different sets of players, and the ability to recover from player failures, without changing the design.

TeC aims for a sweet spot between expressiveness and usability, including innovative and powerful design constructs while keeping their use simple. These design constructs include the expression of spatial constraints, awareness of location and boundaries of semantically meaningful spaces, awareness of the presence and identity of users within those spaces, and constructs for dynamically activating and deactivating features in response to events recognised at the application level.

Acknowledgements

The work herein was funded in part by the National Science Foundation (NSF) under grant CCF-0820060. The authors wish to thank the following students for their prior contributions to several prototypes: A. El Masri, N. Mirzaei, A.P. Sabzevar and V. Tzeremes.

References

- Adjie-Winoto, W. et al. (1999) 'The design and implementation of an intentional naming system', in *7th Symposium on Operating Systems Principles*, ACM, pp.186–201.
- Asmidar, R. and Jais, J. (2009) 'A review on extended role based access control (E-RBAC) model in pervasive computing environment', in *1st Intl. Conf. Networked Digital Technologies*, IEEE CS, Ostrava, Czech Republic, pp.533–535.
- Barkhuus, L. and Dey, A. (2003) 'Is context-aware computing taking control away from the user? Three levels of interactivity examined', in *5th Intl. Conf. Ubiquitous Computing*, Springer LNCS, Seattle, WA, pp.159–166.
- Barkhuus, L. and Vallgård, A. (2003) 'Smart home in your pocket', in *Adjunct. Procs. of the 5th Intl Conf Ubiquitous Computing*, Ubicomp., Seattle, WA, pp.165–166.
- Blackwell, A. and Hague, R. (2001) 'AutoHAN: an architecture for programming the home', in *IEEE Symposia on Human Centric Computing Languages and Environments*, Arlington, VA, pp.150–157.

- Campo, C. and Garcia-Rubio, C. (2006) 'DNS-based service discovery in ad hoc networks: evaluation and improvements', in *11th Intl. Conf. Personal Wireless Communications*, Springer LNCS, pp.111–122.
- Chakraborty, D. et al. (2006) 'Toward distributed service discovery in pervasive computing environments', *IEEE Transactions on Mobile Computing*, Vol. 5, No. 2, pp.97–112.
- Chin, J., Callaghan, V. and Clarke, G. (2006) 'An end-user programming paradigm for pervasive computing applications', in *Intl. Conf. on Pervasive Services*, IEEE, Lyon, France, pp.325–328.
- Covington, M. et al. (2002) 'A context-aware security architecture for emerging applications', in *18th Computer Security Applications Conf.*, IEEE CS, San Diego, CA, pp.249–260.
- Czerwinski, S.E. et al. (1999) 'An architecture for a secure service discovery service', in *5th Intl. Conf. on Mobile Computing and Networking*, ACM, Seattle, WA, pp.24–35.
- Drey, Z., Mercadal, J. and Cousel, C. (2009) 'A taxonomy-driven approach to visually prototyping pervasive computing applications', in *IFIP Working Conf. on Domain-Specific Languages*, Springer LNCS, Oxford, UK, pp.78–99.
- Hightower, J. and Borriello, G. (2001) 'Location systems for ubiquitous computing', *IEEE Computer*, Vol. 34, No. 8, pp.57–66.
- Humble, J. et al. (2003) 'Playing with the bits: user-configuration of ubiquitous domestic environments', in *5th Intl. Conf. Ubiquitous Computing*, Ubicomp., Springer LNCS, Seattle, WA, pp.256–263.
- IETF (1999) *SLP: Service Location Protocol*. Internet Engineering Task Force, available at <http://tools.ietf.org/html/rfc2608> (accessed on 30 June 2011).
- Kawsar, F., Nakajima, T. and Fujinami, K. (2008) 'Deploy spontaneously: supporting end-users in building and enhancing a smart home', in *10th Intl. Conf. Ubiquitous Computing*, Ubicomp, ACM, Seoul, Korea, pp.282–291.
- Ko, A. and Myers, B. (2004) 'Six learning barriers in end-user programming systems', in *IEEE Symp. on Visual Languages and Human Centric Computing*, IEEE CS, Rome, pp.199–206.
- Messer, A. et al. (2006) 'InterPlay: a middleware for seamless device integration and task orchestration in a networked home', in *4th Intl. Conf. on Pervasive Computing and Communications*, PerCom., Pisa, Italy, pp.298–307.
- Ponnekanti, S. et al. (2001) 'ICrafter: a service framework for ubiquitous computing environments', in *3rd Intl. Conf. Ubiquitous Computing*, UbiComp., Springer Verlag, LNCS, Atlanta, GA, pp.56–75.
- Raverdy, P.G. et al. (2006) 'Efficient context-aware service discovery in multi-protocol pervasive environments', in *7th Intl. Conf. Mobile Data Management*, IEEE CS.
- Román, M. et al. (2002) 'Gaia: a middleware infrastructure for active spaces', *IEEE Pervasive Computing*, Vol. 1, No. 4, pp.74–83.
- Sabzevar, A. and Sousa, J.P. (2011) 'Authentication, authorization, and auditing for ubiquitous computing: a survey and vision', *Intl. Journal of Space-Based and Situated Computing*, Vol. 1, No. 1, to appear.
- Sohn, T. and Dey, A. (2003) 'iCAP: an informal tool for interactive prototyping of context-aware applications', in *Conf. on Human Factors in Computing Systems*, extended abstracts, ACM, Ft. Lauderdale, FL, pp.974–975.
- Sousa, J.P. (2005) 'Scaling task management in space and time: reducing user overhead in ubiquitous-computing environments', PhD thesis, Carnegie Mellon University, CMU-CS-05-123.
- Sousa, J.P. (2010) 'Foundations of team computing: enabling end users to assemble software for ubiquitous computing', in *Intl. Conf. on Complex, Intelligent and Software Intensive Systems*, IEEE CS, Krakow, Poland, pp.9–16.
- Sousa, J.P. et al. (2006) 'Task-based adaptation for ubiquitous computing', *IEEE Trans on Systems, Man, and Cybernetics, Part C*, Special issue on Eng Autonomic Systems, Vol. 36, No. 3, pp.328–340.
- Sousa, J.P. et al. (2008) 'Activity-oriented computing', in *Advances in Ubiquitous Computing: Future Paradigms and Directions*, IGI Publishing, pp.280–315.
- Sousa, J.P. et al. (2009) 'A software infrastructure for user-guided quality-of-service tradeoffs', in *Software and Data Technologies*, Springer CCIS, Vol. 47, pp.48–61.
- Sousa, J.P., Poladian, V. and Schmerl, B. (2005) 'Project Aura demo video of the follow me scenario', available at <http://www.cs.cmu.edu/~jpsousa/research/aura/followme.wmv> (accessed on 30 June 2011).
- Truong, K.N., Huang, E.M. and Abowd, G.D. (2004) 'CAMP: a magnetic poetry interface for end-user programming of capture applications for the home', in *6th Intl. Conf. Ubiquitous Computing*, Ubicomp., Springer LNCS, Nottingham, England, pp.143–160.
- Waldo, J. (2000) *The Jini Specification*, 2nd ed., Addison-Wesley-Longman, Boston, MA, USA.
- Zhu, F., Mutka, M. and Ni, L. (2005) 'Service discovery in pervasive computing environments', *IEEE Pervasive Computing*, Vol. 4, No. 4, pp.81–90.