Journal of Universal Computer Science, vol. 17, no. 5 (2011), 670-683
submitted: 8/2/10, accepted: 4/11/10, appeared: 1/1/11 © J.UCS

Bio-Inspired Mechanisms for Coordinating Multiple Instances of a Service Feature in Dynamic Software Product Lines

Jaejoon Lee
(Lancaster University, Lancaster, United Kingdom
j.lee@comp.lancs.ac.uk)

Jon Whittle
(Lancaster University, Lancaster, United Kingdom
whittle@comp.lancs.ac.uk)

Oliver Storz
(Lancaster University, Lancaster, United Kingdom
oliver@comp.lancs.ac.uk)

Abstract: One of the challenges in Dynamic Software Product Line (DSPL) is how to support the coordination of multiple instances of a service feature. In particular, there is a need for a decentralized decision-making capability that will be able to seamlessly integrate new instances of a service feature without an omniscient central controller. Because of the need for decentralization, we are investigating principles from self-organization in biological organisms. As an initial proof of concept, we have applied three bio-inspired techniques to a simple smart home scenario: quorum sensing based service activation, a firefly algorithm for synchronization, and a gossiping (epidemic) protocol for information dissemination. In this paper, we first explain why we selected those techniques using a set of motivating scenarios of a smart home and then describe our experiences in adopting them.

Keywords: Bio-inspired computing, dynamic software product line, variability mechanism
Categories: D.2.12

1 Introduction

Product line engineering is a paradigm of software reuse, which aims at developing a family of products with reduced time-to-market and improved quality [Clements, 02] [Weiss, 99] [Kang, 02]. Most approaches in product line engineering, however, have focused on the development of statically configured products using core assets with static configuration of variation points [Bosch, 02]. That is, all variations are instantiated before a product is delivered to customers and, once the decisions are made, it is difficult for users to alter them.

Recently, there have been increasing demands for dynamic product reconfiguration in various application areas. Dynamic product reconfiguration refers to making changes to a deployed product configuration at runtime. Dynamic addition, deletion, or modification of product features, or dynamic changes of architectural structures [Kramer, 90] are some of the examples. Dynamic product reconfiguration has been studied in various research areas such as self-healing systems [Garlan, 02] [Ganek, 03] [Oreizy, 99], context-aware computing [Yau, 02] [Schilit, 94], software component deployment [Mikic-Rakic, 02] [Hoek, 03] [Hall, 99], and ubiquitous computing [Sousa, 02] [Banavar, 02]. When a change in the operational context is detected, it may trigger product reconfiguration to provide context-relevant services or to meet quality requirements (e.g., performance).

Dynamic reconfiguration approaches in the literature, however, have focused on specific problems of each application area (e.g., behavior models for dynamic changes, context recognition from software or hardware environments, and autonomous management of software component versions), and development of reusable and dynamically reconfigurable core assets has not been fully investigated [Lee, 06] [Gomaa, 04]. As such, a research theme that addresses development issues for reusable and dynamically reconfigurable core assets has emerged and it is called dynamic software product lines (DSPLs) [Hallsteinsen, 08].

An appealing application area for DSPL approaches is that of next generation smart homes. Smart homes are equipped with many small embedded devices that interact with each other and respond to their environment in order to assist and/or improve the daily lives of the homeowner. Smart homes present a number of difficult software engineering challenges. Firstly, these types of system must be highly self-adaptive, able to dynamically reconfigure their behavior both to respond to changes in the environment and to coordinate with other devices that may be in the vicinity. Secondly, the longevity of smart homes implies that devices cannot simply be replaced when they become obsolete. Rather, they must evolve organically over time. Homeowners may purchase new devices to add to their home, remove or update existing devices, or modify the overall requirements that govern how the devices work together. And all of this must be done in a way that the smart home continues to function effectively to satisfy its overall goals. Yet, we still want to reuse core assets to deploy smart home products for various customers and each product should meet user specific requirements.

We focus, in this paper, on one specific software engineering challenge for next generation smart homes: coordination of multiple instances of a service feature in the context of DSPLs. In conventional software product lines, whenever multiple instances of a service feature need to be deployed for a product configuration, this information is gathered at the product analysis phase and taken into consideration for the product configuration. The critical assumption here is that the number of instances is determined and it will not be changed at runtime. For example, in a telephony domain, the maximum number of single line subscribers is determined for a product and their interactions are managed by a central coordinating component. In the smart home domain, this approach (i.e., centralized coordination of predetermined number of instances) is not feasible, because:

- the number of active service features depends on available mobile devices and these devices may join in and leave from a product configuration at runtime;
- a central coordinator cannot be deployed in a particular device due to the mobility and ad-hoc connectivity of devices; and
- the physical location of a device (e.g., two devices close to or far from each other) may also matter for deciding behavior changes (e.g., only one instance of a service feature should be active if multiple instances of them are close to each

other) and no variability management technique adequately supports such a situation.

To address such challenges, we studied various techniques that could support seamless addition/removal of service features and distributed coordination of them in the context of DSPLs. In particular, the issues identified above lead to the need for a decentralized decision-making capability that will, without an omniscient central controller, be able to seamlessly integrate new instances of a service feature. Because of the need for decentralization, we studied principles from self-organization in biological organisms. Organisms such as ants, bees, bacteria and birds are equipped with highly efficient, decentralized control mechanisms that permit rapid behavioral changes to respond to changes in the environment. Our ultimate goal is to investigate the applicability of such mechanisms in DSPLs for smart homes. As an initial proof of concept, we applied three bio-inspired techniques to a simple smart home scenario: quorum sensing based service activation, a firefly algorithm for synchronization, and a gossiping (epidemic) protocol for information dissemination. In this paper, we first explain why we selected those techniques through the motivating scenarios of a smart home and then describe our experiences in adopting them.

The rest of this paper is organized as follows. Section 2 introduces some background information of this paper and section 3 describes how we adopted and applied those techniques. In section 4, a simulation environment is introduced and is followed by related work and conclusions.

2 Background

2.1 Case study: a Smart Cup

A motivating scenario that we will use as a case study in this paper is a smart cup. An in-home networked smart home assists the elderly by monitoring their daily intake of crucial fluids. This is implemented by a sensor-enabled cup that beeps when fluid intake is necessary, has a level to monitor the fluid consumed, shares consumption-data with its neighboring smart cups, and also adjusts the required intake level when notified of consumption-data from other smart cups. This service feature is called a drinking reminder feature and each cup has an instance of this feature. (See Figure 1 for a feature model of a smart home, which includes this feature.)

The scenario currently assumes the presence of the following devices and functionalities:

- A smart cup is equipped with tilt sensors that provide each cup with information about the angle it is being held at. If provided with information about the initial fluid level, cups are able to use the information provided by the tilt sensor to infer how much fluid has been drained from the cup.
- A smart cup also has a beeper that enables it to produce an audible alarm if the targeted fluid intake has not been reached. The cup runs an *evaluation cycle* with a periodic frequency; at each period, the cup evaluates current fluid intake and beeps if necessary.
- A smart tap has a sensor that measures the amount of water that is emitted from the tap into cup devices. The tap uses its built-in radio to communicate

information about the amount of liquid that has been dispensed into the cup device that is currently positioned underneath the tap.

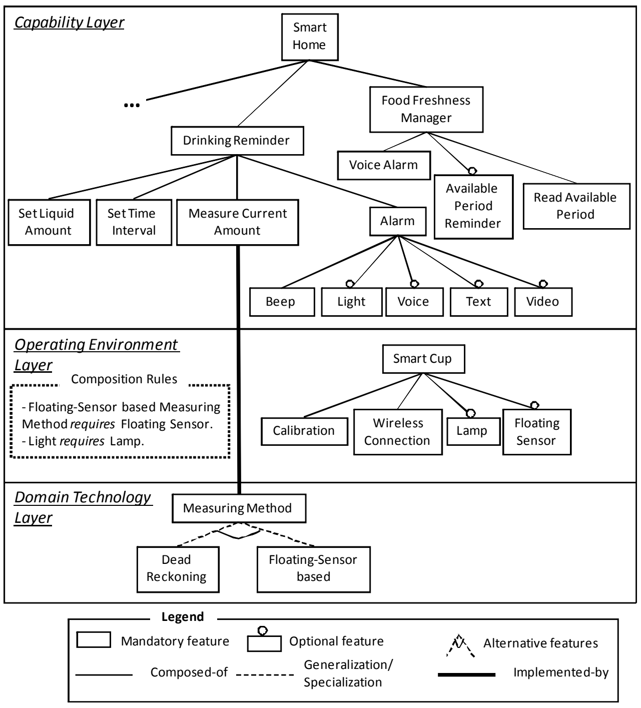

Figure 1: Feature Model of a Smart Home

The problems posed by this scenario include:

1. the dissemination of information about the fluid intake to other smart cups in the smart home to enable these devices to reach a decentralized decision as to whether enough fluid has been consumed or not;
2. the dissemination of the intended duration of the evaluation frequency, the targeted fluid intake, and the evaluation window during which actions of liquid intake should be considered as being relevant for the current evaluation,
3. the synchronisation of the evaluation cycles of the individual smart cup, with the aim of getting all cups to evaluate (and therefore sound the alarm) in a coordinated fashion, i.e., at the same time, and
4. the selection of smart cups within the home that should be responsible for sounding the audio alarm; the aim is to ensure that the alarm can ideally be perceived from every location within the home, whilst minimizing the number of smart cups that beep in order to conserve power and to avoid a cacophony of sound, which would annoy homeowners.

The following subsections provide background on the bio-inspired algorithms which we apply to this smart cup scenario. Note that the gossip and firefly algorithms already exist in the literature, but our contribution is a novel application domain; the quorum sensing algorithm is new.

2.2 Information Propagation: Gossip (Epidemic) Protocol

Recent years have seen the emergence of systems that disseminate information in computer networks using protocols that mimic how humans spread information through gossiping: individuals have knowledge of pieces of information and from time to time exchange a subset of their knowledge with other individuals, who may in turn pass this information on to further individuals, thereby gradually disseminating the information to all individuals in the peer group. (It is also called as an epidemic protocol, because the spread of information is similar to the spread of a virus in a biological environment.)

A node *N* that is participating in gossip-based information exchange periodically carries out the following steps [Bakhshi, 09]:

1. *N* randomly selects one other node (node *B*);
2. *N* randomly selects one or more information items from its local repository of information;
3. *N* sends the selected information items to *B*, causing *B* to add the received information to its local repository;
4. optionally:
    a. *N* requests *B* to provide randomly selected information from *B*'s repository;
    b. *N* adds the information received from *B* to its own repository.

Gossip protocols have been shown to be robust in the presence of failures of individual nodes [Gavidia, 06]. They have, for example, been used to disseminate news items in wireless mesh networks [Gavidia, 06], or to disseminate state information with the aim of constructing hierarchies of clusters in wireless sensor networks [Iwanicki, 09].

2.3 Synchronization: Firefly Algorithm

As part of their mating ritual, male fireflies gather in groups and flash in a synchronized fashion [Mirollo, 90]. The algorithm used to synchronize the flashing is completely decentralized and relies purely on each fly observing the light pulses emitted by its neighbors. The following is an overview of the algorithm [Azar, 07]:

1. flies flash with a certain frequency;
2. once a fly has flashed it has to gradually build up enough internal charge before it can flash again;
3. if a fly observes one of its neighbors flashing, it advances its internal clock to bring itself closer to flashing. This is achieved by applying a boost to the fly's internal charge;
4. the strength of the boost applied depends on the time that has passed since the firefly has last emitted a flash. The closer the flies are to flashing themselves, the larger the boost.

2.4 Feature Activation Control: Quorum Sensing

The term 'quorum-sensing' refers to the ability of organisms to activate certain behaviours based on the density of other organisms in their environment. Since all organisms within a certain region experience the same density, quorum-sensing can be used to reach consensus among organisms.

Quorum-sensing in the bacterial world is, for example, used for triggering the production of virulence factors or spores, for controlling bioluminescence or for controlling different phases in the development of bio-films [Bassler, 06].

The quorum-sensing circuit of bacteria generally works as follows:

1. bacteria release special molecules (called "auto-inducers") into the environment, either as part of normal growth or in response to changes in the environment. Autoinducers are typically molecules from the N-acyl homoserine lactone (AHL) family in the case of Gram-negative bacteria and short peptides in the case of Gram-positive bacteria [Bassler, 06];
2. these molecules accumulate in the environment;
3. bacteria of the same species that released the auto-inducers or in some cases different species of bacteria have special receptors for recognising the auto-inducer molecules;
4. once the auto-inducer concentration reaches a certain threshold, certain genetic sequences, and therefore certain behaviours, are activated.

The famous 'waggle dance' of bees is an example of quorum sensing. When searching for new nests, bees explore potential sites and, upon returning from a site, they recruit other bees to their site by 'dancing'. The number of dance repetitions is proportional to the quality of the site. Hence, bees finding good sites are able to recruit bees finding bad sites – bees finding poor sites stop dancing sooner. When the density of bees at a new site increases beyond a certain threshold, a decision is made for the whole swarm to move to the site.

3 Applying Bio-Inspired Algorithms to Smart Homes

We applied these three techniques to address the problems described in section 2.1. In the following, we explain how we adopted and applied the techniques in detail.

3.1 Sharing Consumed Liquid Amount Data

We propose to use a gossip protocol to propagate information within the smart home. While our design focuses on the dissemination of information about the fluid intake, similar gossip-based approaches can be used to disseminate other information, including the intended frequency that should be used for evaluating the liquid intake, the targeted intake, and evaluation window.

Each time fluid is consumed through one of the smart cups, it records information about this event, including:

- the amount of liquid that was consumed;
- the time that has elapsed since the event took place, thus enabling devices to discard events that occurred too far in the past and are therefore irrelevant for determining the overall fluid intake over the current evaluation period.

From time to time each smart cup exchanges a subset of these records (and any records that it has received from other smart cups) with the smart cups that are in communication range, i.e., they gossip. A smart cup receiving such a gossip message stores the received records, unless they are duplicates of records already present on the smart cup.

To enable the receiving smart cup to detect duplicates of records that it already possesses, each record additionally contains a globally unique identifier that is generated when the record is created. A received record is only stored by a smart cup if the unique id of that record does not match any records that are already present on it. The algorithm used by smart cups to record, manage and propagate records of fluid intake is outlined in Figure 2.

```
while true:
    if device clock has been advanced:
        increment elapsedTime on each stored record
        expire all records that are older than max
clock ticks
    if new fluid has been consumed:
        new_record = new ConsumptionRecord()
        new_record.elapsedTime = 0
        new_record.devicName = this.name
        new_record.consumption = newly consumed amount
        new_record.uid = create_new_uid()
        store new_record
    if time to communicate:
        if fluid has been consumed recently
           through this device:
            broadcast all records concerning this
                      device to devices in range
        randomly select a device we have records for
        broadcast that device's records to devices
        in range
    if we have received records:
        if we do not already possess these records:
            store these records
```

Figure 2: Algorithm used by devices to record, manage and propagate information about the amounts of fluid consumed

3.2 Synchronizing Checking Period

If a smart cup receives such a message (this act is similar to observing the flash of a firefly), it compares the timing information contained in that message with its own timing information. If the clocks of both smart cups are equal or are within a certain tolerance, the receiver records the sender and the smart cups are synchronised with the sender.

If the clock cycles differ, the receiver compares the number of smart cups that the sender considers itself synchronised with and the number of smart cups the receiver considers itself synchronised with. If the receiver is synchronised with a larger number of smart cups than the sender, the receiver discards the message. If the sender is synchronised with a larger number of smart cups than the receiver, the receiving device adjusts its clock cycle to the sender's clock cycle. This action is similar to fireflies boosting their internal charge and flashing sooner after having received a neighbour’s flash with the aim of eventually synchronising itself to the timing of their neighbouring fly.

The oscillation frequency of fireflies is limited by the speed at which their internal potential can be charged up. Hence the speed at which fireflies are able to advance their internal clock is limited. However, similar limitations do not exist in the context of smart cups in our scenario: once a receiver has decided to synchronise its own clock to the clock of the sender, the receiver fully advances its internal clock cycle to match the sender’s clock cycle.

```
while true:
    if time to emit synchronisation message:
        message = new SynchronisationEvaluationMessage()
        message.sender = this.name
        message.timeToNextEvaluation =

                                   ourTimeToNextEvaluation
        message.synchronisedDevices = list of devices
               we are synchronised with broadcast message
    if synchronisation message received:
        if sender’s clock cycle and
           our clock cycle are within tolerance:
            store sender as being in sync
            store message.synchronisedDevices

                  as being in sync
        else:
            if length(message.synchronisedDevices) >
               length(ourSynchronisedDevices):
                synchronise our clock to match
                            message.timeToNextEvaluation
                store sender as being in sync
                store message.synchronisedDevices

                      as being in sync
            else if length(message.synchronisedDevices)
```

Figure 3: An outline of the synchronization algorithm

If both smart cups have an equal number of synchronised smart cups, the receiver synchronises its clock to the sender’s clock cycle if the sender is closer to evaluating the fluid intake.

By basing decisions on the number of other smart cups that a smart cup is synchronised with, we avoid situations where a smaller group of synchronised nodes can cause a much larger group of synchronised nodes to abandon their synchronisation and synchronise to the clocks of the smaller group. An outline of the algorithm is presented in Figure 3.

3.3 Selecting a Beeping Cup

Once a smart cup has come to the conclusion that the targeted fluid intake has not been reached, it uses a quorum-sensing inspired algorithm to decide whether to activate its built-in audible alarm or not. To achieve this, smart cups activate their built-in microphones and listen for audible alarm signals emitted by other smart cups for a randomly selected number of clock ticks.

If a smart cup receives an audio signal (in the frequency band that is used by the alarm) that is above a certain volume threshold during this period, they assume that a smart cup which is close by has already activated its alarm signal, and that activating another alarm signal in this area is unnecessary. As a result, the smart cup does not activate its own audio alarm. If the randomly selected number of clock ticks has passed and the smart cup has not received any signals above the given volume threshold, it infers that the area it is located in does not contain any others that have activated their alarm signal. It therefore activates its own alarm signal. An overview of the algorithm is presented in Figure 4.

```
if not enough fluid consumed:
    activate microphone
    time_to_listen = select random number
                     between 0 and n_listen
    listen for time_to_listen clock ticks
    deactivate microphone
    if not alarm signal above
       critical volume threshold received:
```

Figure 4: Algorithm used to decide whether to activate the audio alarm

In analogy to quorum-sensing, the algorithm uses the volume of the audio-alarm signals received from other devices to infer the density of the population of smart cups with activated alarms. The behaviour activated by the quorum-sensing circuit once the critical population density has been reached (i.e. the volume of the audio signal is above a certain threshold), is the abstinence from activating the smart cup's own audio circuitry. The quorum-sensing-based decision remains valid until the fluid intake is evaluated again.

4 A Simulation Environment

We have implemented a simulation environment for the scenario described in section 2.1 and have implemented the algorithms presented in sections 3.1, 3.2 and 3.3. The user interface of the simulation provides users with the functionality to add smart cups and other devices to the simulation, change their position, fill and drink from smart cups, and control the simulation time. (See Figure 5 for a screenshot of the simulator.)

Following the algorithm outlined in section 3.1, each smart cup maintains a cache of consumption records. Internally each smart cup maintains two different types of liquid consumption data:

- a list of consumption records relating to fluid that has been consumed through this device, and
- another consumption record related to fluid consumed through other devices.

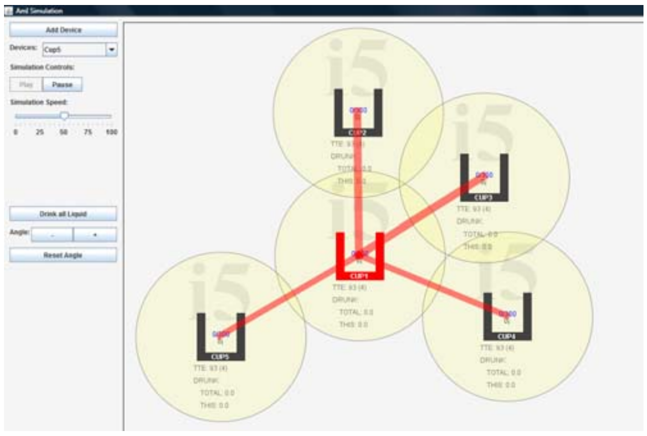

Figure 5: Screenshot of the simulator's user interface

Each time the simulation clock is incremented, both data sources are checked, and the data that are older than a predefined time window expire. Also, each smart cup is responsible for managing its local simulation time and synchronizes its clock with other smarts cups by using the algorithm explained in section 3.2.

Once a smart cup has decided to beep as the fluid consumption was insufficient, it first listens for other devices in its vicinity, as described in section 3.3. The model for simulating audio propagation is based on the distance between the individual devices. For simplicity, the simulation currently assumes that volume decreases linearly with increasing distance from the sound source. In reality, however, sound waves experience a quadratic decrease. To further illustrate the attenuation of sound by obstacles, the simulator offers the ability to introduce Wall objects into the simulation environment. In the simulation, Walls attenuate the amplitude of an audio signal by a factor of 2. The strength of the sound signal in the simulator is visualized by the thickness of the red line between devices as shown in Figure 5: the thicker the line is, the louder each cup can hear the beeping sound.

5 Related Work

The study of self-organization in computing is not new (see, for example, a survey paper from 1986 [Robertazzi, 86]). To date, however, this large body of work has mainly focused on network management issues in, for example, the Internet, wireless sensor networks and grid computing. The focus on these types of applications has meant that research has concentrated around infrastructure issues such as allocation of bandwidth and processing capacity, dissemination of information, power conservation, and fault tolerance.

We are more concerned with **how to design software** in a way that supports self-organization: for example, flexible design structuring techniques for software artefacts, software models that support self-organization, requirements for self-organizing software, and management of self-organizing software. Some notable works on self-organization are mentioned below.

Self-organization in wireless sensor networks has become a hot research topic recently, for example, to adapt to changing traffic patterns or to reconfigure topologies. There has been work in resource sharing (e.g., [Duque-Anton, 97]), forming and maintaining structures (e.g., [Cerpa, 02]), information dissemination (e.g., [Intanagonwiwat, 00]), and resilience (e.g., [Gupta, 03]). Such work, however, focuses on low levels of abstraction and does not address problems usually associated with software engineering such as how to design software to support self-organization.

There is also a distinguished history of bio-inspired computing. Some researchers (e.g.,[Abelson, 00] [Lodding, 04] [Nagpal, 03]) have studied embryogenesis as a means to design hardware-software architectures for persistent computing environments. Hofmeyr and Forrest [Hofmeyr, 00] describe an intrusion-detection system inspired by the human immune system. IBM's autonomic computing initiative uses the human nervous system to inspire self-managing systems. Evolutionary computation derives inspiration from natural selection. Recently, approaches based on studying swarms have been studied—for example, it has been shown that flock behavior of birds can be derived from three simple rules [Reynolds, 87] and these rules became the basis for computer graphics in the film *Batman Returns*. Despite all of this excellent research on (bio-inspired) self-organization, one area that seems to have been largely omitted to date is bio-inspired software engineering.

6 Conclusion

One of the challenges in DSPLs is the support for coordinating multiple instances of a service feature. In particular, there is the need for a decentralized decision-making capability that will be able to seamlessly integrate new instances of a service feature without an omniscient central controller. Because of the need for decentralization, we studied principles from self-organization in biological organisms. As an initial proof of concept, we applied three bio-inspired techniques to a simple smart home scenario: *quorum sensing* based service activation, a *firefly algorithm* for synchronization, and a *gossiping (epidemic) protocol* for information dissemination.

The simulation results show that it is feasible to apply such mechanisms in DSPLs for smart homes. The current simulator, however, also reveals its limitations:

- the support for heterogeneous devices: we assume that only the smart cups are involved in the *drinking reminder* service feature. But, other devices such as a smart TV for a visual notification to a user can also be involved and the coordination among them needs to be further investigated;
- the enhancement of service quality: it is possible that a user uses the smart cup for watering a flower pot instead of drinking, but the question is how we can recognize such a situation to provide a high quality (accuracy for the drinking reminder) service. Therefore we should be able to recognize and integrate various optional sensors (e.g., humidity sensor installed at the flower pot) at runtime and provide more dependable services;
- the deployment of smart cups in a real environment: in a real smart home environment, these algorithms may not work as in the simulation environment due to noise, the short range of wireless communications, etc. By deploying

the software to a real hardware device, we might be able to explore more challenging issues.

We'll continue to explore the bio-inspired ideas to address these issues. We believe that this will lead us to deliver some fundamentally new insights on variability mechanisms inspired by the self-organization, dynamism and adaptation capabilities of biological organisms. Also, we're currently developing a smart cup on a hardware device and will perform an empirical study on pros/cons for adopting bio-inspired mechanisms to DSPLs.

Acknowledgements

This research is carried out in the Small Grants Scheme 'BIO-SCALE: Bio-inspired Software Composition for Assisted Living Environments' project, which is funded by Lancaster University, United Kingdom.

References

[Abelson, 00] Abelson, H., et al.: Amorphous computing, *Communications of the ACM*, vol. 43, 74-82, 2000.

[Azar, 07] Azar., P.: *Fireflies & Oscillators*. Harvard College Mathematics Review. Vol. 1 No. 2. 2007.

[Bakhshi, 09] Bakhshi, R., et al.: *An Analytical Model of Information Dissemination for a Gossip-based Wireless Protocol*. Proc. 10th Int'l Conf. Distributed Computing and Networking (ICDCN), Gachibowli Hyderabad, India, January 2009.

[Banavar, 02] Banavar, G., Bernstein, A.: Software infrastructure and design challenges for ubiquitous computing applications, *Communications of ACM*, Vol.45, Issue 12, 2002, 92-96.

[Bassler, 06] B. Bassler and R. Losick. *Bacterially Speaking*. Cell, Volume 125, Issue 2, 2006, 237-246.

[Bosch, 02] Bosch, J., et al., Variability Issues in Software Product Lines, In: van der Linden, F. (eds.): Software Product Family Engineering. *Lecture Notes in Computer Science*, Vol. 2290. Springer-Verlag, Berlin Heidelberg 2002, 13-21.

[Cerpa, 02] Cerpa, A., Estrin, D.: ASCENT: Adaptive self-configuring sensor networks topologies, in *IEEE INFOCOM*, 2002, 1278-1287.

[Clements, 02] Clements, P., Northrop, L.: *Software Product Lines: Practices and Pattern,* Addison Wesley, Upper Saddle River, NJ, 2002.

[Duque-Anton, 97] Duque-Anton, M., et al.: Extending Kohonen's self-organizing mapping for adaptive resource management in cellular radio networks, *IEEE Transactions on Vehicular Technology*, vol. 46, 1997, 560-568.

[Ganek, 03] Ganek, A.G., Corbi, T.A.: The drawing of the autonomic computing era, *IBM Systems Journal*, Vol. 42, No. 1 2003, 5-18.

[Garlan, 02] Garlan, D., Schmerl, B.: Model-based Adaptation for Self-Healing Systems, *Proceeding of the Workshop on Self-Healing Systems* (WOSS'02), Nov.18-19, 2002, 27-32.

[Gavidia, 06] Gavidia, D., Voulgaris, S., van Steen., M.: *A Gossip-based Distributed News Service for Wireless Mesh Networks*. Proc. 3rd IEEE Conference on Wireless On demand Network Systems and Services (WONS), Les Menuires, France, January 2006.

[Gomaa, 04] Gomaa, H., Hussein, M.: Dynamic Software Reconfiguration in Software Product Families, In: van der Linden, F. (eds.): Software Product Family Engineering, *Lecture Notes in Computer Science*, Vol. 3014. Springer-Verlag, Berlin Heidelberg 2004, 435-444.

[Gupta, 03] Gupta, G., Younis, M.: Fault-tolerant clustering of wireless sensor networks, in *IEEE Wireless Communications and Networking Conference*, 2003, 1579-1584.

[Hall, 99] Hall, R.S., Heimbigner, D.M., Wolf, A.L.: A cooperative approach to support software deployment using the software dock, *Proceedings of the 1999 International Conference on Software Engineering,* ACM Press: New York, NY 1999, 174-183.

[Hallsteinsen, 08] Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic Software Product Lines, IEEE Computer, Vol. 41 (4), 2008, 93-95.

[Hoek, 03] Hoek, A. van der, Wolf, A.L.: Software release management for component-based software, *Software-Practice and Experience*, Vol.33, 2003, 77-98.

[Hofmeyr, 00] Hofmeyr, S., Forrest, S.: Architecture for an artificial immune system, *Evolutionary Computation,* vol. 8, 2000, 443-473.

[Intanagonwiwat, 00] Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *6th Annual Conference on Mobile Computing and Networking*, 2000, 56-67.

[Iwanicki, 09] Iwanicki, K., van Steen., M.: *Multi-hop Cluster Hierarchy Maintenance in Wireless Sensor Networks: A Case for Gossip-Based Protocols*. Proc. 6th European Conference on Wireless Sensor Networks (EWSN), Cork, Ireland, February 2009.

[Kang, 02] Kang, K., Lee, J., Donohoe, P.: Feature-Oriented Product Line Engineering, *IEEE Software*, 19(4), July/August 2002, 58-65.

[Kramer, 90] Kramer, J., Magee, J.: The Evolving Philosophers Problem: Dynamic Change Management, *Transaction on Software Engineering*, Vol. 16, No. 11, November 1990, 1293-1306.

[Lee, 06] Lee, J., Kang, K.: A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering, In the Proceeding of the 10th International Software Product Line Conference (SPLC), August 21-24, 2006, 131-140.

[Lodding, 04] Lodding, K.N.: Hitchhikers guide to biomorphic software, *ACM Queue,* vol. 2, 66-75, 2004.

[Mikic-Rakic, 02] Mikic-Rakic, M., Medvidovic, N.: Architecture-Level Support for Software Component Deployment in Resource Constrained Environments, *Proceedings of First International IFIP/ACM Working Conference on Component Deployment*, Berlin, Germany, 2002, 31-50.

[Mirollo, 90] Mirollo, R.E., Strogatz., S.H.: *Synchronization of pulse-coupled biological oscillators*. SIAM J. Appl. Math. 50, 6 (Nov. 1990), 1645-1662. DOI= http://dx.doi.org/10.1137/0150098

[Nagpal, 03] Nagpal, R.: A catalog of biologically-inspired primitives for engineering self-organization, *Engineering Self-Organising Systems,* vol. 2977, 2003, 53-62.

[Oreizy, 99] P. Oreizy, et al., An Architecture-Based Approach to Self-Adaptive Software, *IEEE Intelligent Systems*, May/June 1999, 54-62.

[Robertazzi, 86] Robertazzi, T.G., Sarachik, P.E.: Self-organizing communication networks, *IEEE Communications,* vol. 24, 1986, 28-33.

[Reynolds, 87] Reynolds, C.W.: Flocks, herds and schools: a distributed behavioral model, *Computer Graphics,* vol. 21, 1987, 25-34.

[Schilit, 94] Schilit, B., Adams, N., Want, R.: Context-Aware Computing Applications, *Proceedings of IEEE Workshop Mobile Computing Systems and Applications*, IEEE CS Press, Los Alamitos, Calf. 1994, 85-90.

[Sousa, 02] Sousa, J.P., Garlan, D.: Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments, *Proceeding of the 3rd Working IEEE/IFIP Conference on Software Architecture*, Kluwer Academic Publishers 2002, 29-43.

[Weiss, 99] Weiss, D.M, Lai, C.T.R.: *Software Product-Line Engineering: A Family-Based Software Development Process*, Reading, MA: Addison Wesley Longman, Inc., 1999.

[Yau, 02] Yau, S.S., et al.: Reconfigurable Context-Sensitive Middleware for Pervasive Computing, *Pervasive Computing,* July/September 2002, 33-40.