Adaptive Agents for Cyber-Physical Systems

Ichiro Satoh
National Institute of Informatics, 2-1-2 Hitotsubashi Chiyoda-ku, Tokyo, 101-8430, Japan

Keywords: Self-adaptation, Bio-inspired Approach, Cyber-Physical Systems, Agent.

Abstract: This paper proposes a bio-inspired approach to adapting software components in CPSs. It introduces the notions of differentiation and dedifferentiation in cellular slime molds. When a software component delegates a function to another component coordinating with it, if the former has the function, this function becomes less-developed and the latter's function becomes well-developed like that in cellular differentiation. The approach enables software components on CPSs to be naturally adapted to changes in the cyber and physical world in a self-organizing manner. It is constructed as a middleware system to execute general purpose applications on a CPS. We present several evaluations of the approach in CPSs.

1 INTRODUCTION

Cyber-physical systems (CPSs) are complicated and dynamic by nature. They must adapt themselves to changes in the physical world. For example, the sensing systems for catastrophes, e.g., earthquakes, deluges, radiation, and forest fires, have been designed extensively against disasters. CPSs must be resilient. In fact, even when most sensor nodes are damaged, remaining nodes should organize themselves and achieve their goals as much as possible. CPSs should also adapt themselves to the requirements of multiple applications. Nodes in CPSs may be heterogenous and do not need to provide the same tasks unlike those in peer-to-peer computing systems and sensor networks. They should delegate some tasks to other nodes to save their resources.

The complexity and dynamism of CPSs are beyond our ability to build and manage systems through traditional approaches, such as those that are centralized and top-down. Instead, we propose an evolutionary approach to manage CPSs with small and simple rules for adaptation. It introduces the notion of *(de)differentiation* into CPSs. When similar functions are running on different nodes connected through a network, one of them should automatically be placed by the other. Remaining nodes should provide those functions that were provided by disconnected nodes as much as possible, when the network is partitioned.

Differentiation is the mechanism by which cells in a multicellular organism become specialized to perform specific functions in a variety of tissues and organs. Different kinds of cell behaviors can be observed during embryogenesis: cells double, change in shape, and attach at and migrate to various sites within the embryo without any obvious signs of differentiation. A CPS is a system featuring a tight combination of computational and physical components. The former consists of hardware and software components. Our approach offers a mechanism for adapting software components, which may be running on computers connected through a network. The approach involves treating the undertaking/delegation of functions in software components from/to other components, including physical and hardware components, as their differentiation factors. When a software component delegates a function to another component, if the former has the function, its function becomes less-developed. The latter's function becomes well-developed if the latter is a software component. The proposed system also should be reasonable and available in real systems instead of in simulation-based systems.

2 RELATED WORK

The notion of self-organization is rapidly gaining importance in the area of CPSs. We discuss several related studies on software adaptation in distributed and ubiquitous computing systems in addition to CPSs.

Several researchers have explored adaptive approaches to managing CPSs, but most of them have aimed at managing networks rather than functions in CPSs. In fact, they were constructed based on

Satoh I..
Adaptive Agents for Cyber-Physical Systems.
DOI: 10.5220/0004244402570262
In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence* (ICAART-2013), pages 257-262
ISBN: 978-989-8565-39-6

wireless sensor networks. One of the most typical self-organization approaches to distributed systems is swarm intelligence (Bonabeau et al., 1999; Dorigo and Stutzle, 2004). Although there is no centralized control structure dictating how individual agents should behave, interactions between simple agents with static rules often lead to the emergence of intelligent global behavior. Most existing approaches have only focused on their target problems or applications but they are not general purpose, whereas CPSs are general purpose. Our software adaptation approach should be independent of applications. Furthermore, most existing self-organization approaches explicitly or implicitly assume a large population of agents or boids. However, real CPSs have no room to execute such large numbers of agents. In fact, a CPS consists of embedded computers, whose computational resources are limited. Consequently, our software adaptation mechanism for CPSs must spend little computational resources of the systems for software adaptation as possible.

Georgiadis et al. (Georgiadis et al., 2003) presented connection-based architecture for self-organizing software components on a distributed system. Like other software component architectures, they intended to customize their systems by changing the connections between components instead of the internal behaviors inside them. Like ours, Cheng at al. (Cheng et al., 2006) presented an adaptive selection mechanism for servers by enabling selection policies, but they did not customize the servers themselves. They also needed to execute different servers simultaneously. Herrman et al. proposed the bio-inspired deployment of services on sensor networks (Herrman, 2008). Unlike ours, their work focused on the deployment and coordination of services, instead of the adaptation of software itself to provide services.

We proposed a nature-inspired approach to dynamically deploying agents at computers in our previous paper (Satoh, 2007). The approach enabled each agent to describe its own deployment as a relationship between its location and another agent's location. However, the approach had no mechanism for differentiating or adapting agents themselves. We also presented an early prototype of this approach (Satoh, 2011) and its extension for sensor networks (Satoh, 2012). The former aimed at an adaptive mechanism in application-specific services and the latter was an early implementation of the approach.

3 APPROACH

Traditional management approaches may not be able to support complications or dynamism in CPSs. Instead, we introduce a bio-inspired approach to CPSs. The basic inspiration for our approach lies in cellular differentiation, which is the process by which a less specialized cell develops or matures to possess a more distinct form and function in developmental biology. When a parent cell divides into two or more daughter cells, the latter may be differentiated from the former. For example, cellular slime molds, e.g., dictyostelium discoideum, are eukaryotic microorganisms in the soil. Once food becomes scarce, dictyostelium discoideum cells start to aggregate and differentiate themselves. They can also be differentiated into two types: prespore cells and prestalk cells. Each cell tries to become a prespore cell and periodically secretes cyclic-adenosine-monophosphate (cAMP) to other cells. If a cell can receive more than a specified amount of cAMP from other cells, it can become a prespore cell. There are three rules. 1) cAMP chemotactically leads other cells to prestalk cells. 2) A cell that is becoming a prespore cell can secrete a large amount of cAMP to other cells. 3) When a cell receives more cAMP from other cells, it can secrete less cAMP to other cells.

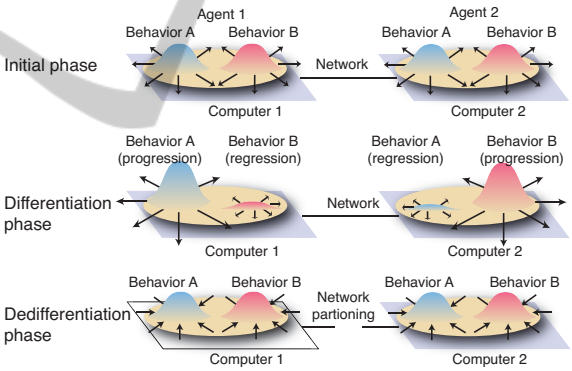

Figure 1: Differentiation mechanism for software adaptation.

Our approach introduces the notion of (de)differentiation into software components, called *agents* that is like a cell, where each agent is defined as autonomous software consisting of one or more functions. We provide hardware or physical components in CPSs with proxies that can interact with agents. The approach involves treating the undertaking/delegation of functions in agents from/to other agents as their differentiation factors. Each function is a programmable entity and has its own weight corresponding to the amount of cAMP and this indicates its superiority. When an agent delegates a function to another agent, if the former has the function, its function becomes less-developed and the latter's function becomes well-developed. Agents

may lose their functions due to differentiation as well as being busy, partitioned, or having failed. The approach also offers a mechanism for recovering from such problems based on dedifferentiation, which is a mechanism for regressing specialized cells to simpler, more embryonic, unspecialized forms. As happens in the dedifferentiation process, if there are no other agents that are sending restraining messages to an agent, the agent can perform its dedifferentiation process and strengthen its less-developed or inactive functions again.

4 SELF-ADAPTATION MECHANISM

Our approach is maintained through two parts: runtime systems and agents. The former is a middleware system for running on computers in CPSs and the latter is a self-contained and autonomous software entity. It has three protocols for (de)differentiation and delegation.

4.1 Software Component

Software components in CPSs are implemented as agents, where each agent consists of one or more functions, called the *behavior* parts, and its state, called the *body* part, with information for (de)differentiation, called the *attribute* part. The body part maintains program variables shared by its behaviors parts like instance variables in object orientation. The behavior part defines more than one application-specific behavior. It corresponds to a method in object orientation. The attribute part maintains descriptive information with regard to the agent, including its own identifier.

The agent has behaviors $b_1^k, \ldots, b_n^k$ and w_i^k is the weight of behavior b_i^k. Each agent (k-th) assigns its own maximum to the total of the weights of all its behaviors. The W_i^k is the maximum of the weight of behavior b_i^k. The maximum total of the weights of its behaviors in the k-th agent must be less than W^k. ($W^k \geq \sum_{i=1}^{n} w_i^k$), where $w_j^k - 1$ is 0 if w_j^k is 0. The W^k may depend on agents. In fact, W^k corresponds to the upper limit of the ability of each agent and may depend on the performance of the underlying system, including the processor. Note that we never expect that the latter will be complete, since agents periodically exchange their information with neighboring agents. Furthermore, when agents receive no retraining messages from others for longer than a certain duration, they remove information about them.

4.2 Protocol for Function Invocation

When an agent wants to execute a behavior, it needs to select one of the available behaviors ($b_i^j, \ldots, b_i^m$), even if it has the behavior, according to the values of their weights. This involves three steps.

i) When an agent (k-th agent) wants to execute behavior b_i, it looks up the weight (w_i^k) of the same or a compatible behavior from its database and the weights ($w_i^j, \ldots, w_i^m$) of such behaviors ($b_i^j, \ldots, b_i^m$) from the database (Figure 2 (a))

ii) If multiple agents, including itself, can provide the wanted behavior, the k-th agent selects one of the agents according to selection function ϕ^k, which maps from w_i^k and $w_i^j, \ldots, w_i^m$ to b_i^l, where l is k or $j, \ldots, m$.

iii) The k-th agent delegates the selected agent to execute the behavior b_i^l and waits for the result from the l-th agent (Figure 2 (b)).

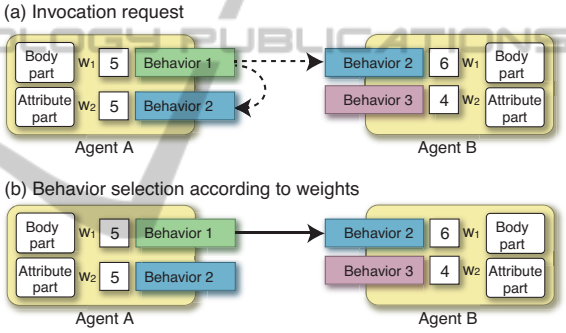

Figure 2: Selective invocation.

There is no universal selection function, ϕ, for mapping from the weights of behaviors to at most one appropriate behavior like that in a variety of creatures. Instead, the approach permits agents to use their own evaluation functions, because the selection of behaviors often depends on their applications. For example, one of the simplest evaluation functions makes the agent that wants to execute a behavior select one whose weight has the highest value and whose signature matches the wanted behavior if its database recognizes one or more agents that provide the same behavior, including itself.

4.3 Protocol for Adaptation

The approach introduces the undertaking/delegation of behaviors in agents from other agents as a differentiation factor. Behaviors in an agent, which are delegated from other agents more frequently, are well developed, whereas other behaviors, which are delegated from other agents less frequently, in the cell are

less developed. Finally, the agent only provides the former behaviors and delegates the latter behaviors to other agents. Our differentiation mechanism consists of two phases. The first involves the progression of behaviors in three steps.

i) When an agent (k-th agent) receives a request message from another agent, it selects the behavior (b_i^k) specified in the message from its behavior part and dispatches the message to the selected behavior (Figure 3 (a)). It executes the b_i^k behavior and returns the result.

ii) The k-th agent increases weight w_i^k of the b_i^k behavior (Figure 3 (b)).

iii) The k-th agent multicasts a restraining message with the signature of the behavior, its identifier (k), and the behavior's weight (w_i^k) to other agents (Figure 3 (c)).

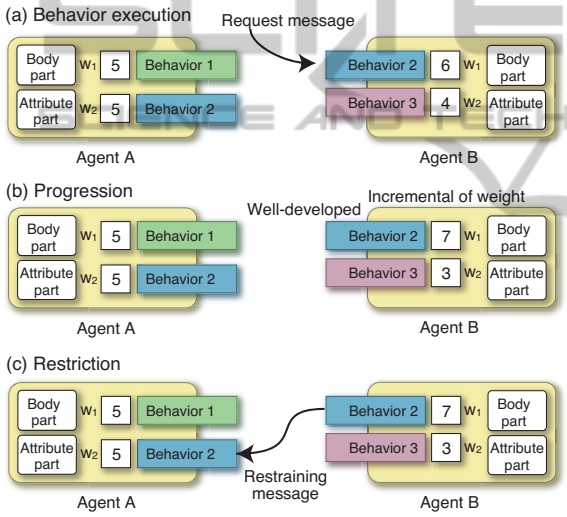

Figure 3: Differentiation-based adaptation on called side.

When behaviors are internally invoked by their agents, their weights are not increased. If the total weights of the agent's behaviors, $\sum w_i^k$, is equal to their maximal total weight W^k, it decreases one of the minimal (and positive) weights (w_j^k is replaced by $w_j^k - 1$ where $w_j^k = \mathbf{min}(w_1^k, \ldots, w_n^k)$ and $w_j^k \geq 0$).[1] Although restraining messages correspond to the diffusion of cAMP in differentiation, they can explicitly carry the weights of the agents that send them to reduce the number of restraining messages, because they can be substituted for more than one retaining message without weights. The second phase supports the retrogression of behaviors in three steps.

i) When an agent (k-th agent) receives a restraining message with regard to b_i^j from another agent (j-th), it looks for the behaviors ($b_m^k, \ldots b_l^k$) that can have the signature specified in the received message (Figure 3 (c)).

ii) If it has such behaviors, it decreases their weights ($w_m^k, \ldots w_l^k$) in its database and updates the weight (w_i^j) in its database (Figure 4 (a)).

iii) If the weights ($w_m^k, \ldots, w_l^k$) are under a specified value, e.g., 0, the behaviors ($b_m^k, \ldots b_l^k$) are inactivated (Figure 4 (b)).

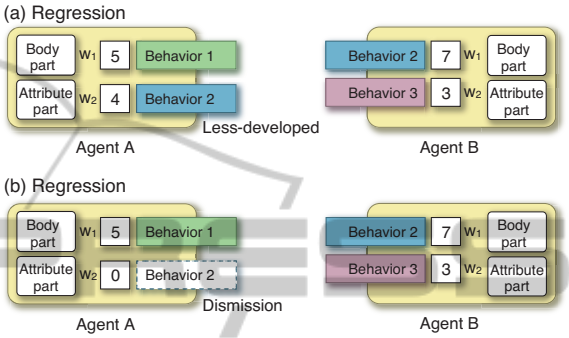

Figure 4: Differentiation-based adaptation on other sides.

CPSs may be damaged or stop due to disasters and problems. We need a mechanism for detecting and remedying failures in networking, computers, agents, remote computers, and other agents. To do this, each agent (j-th) periodically multicasts messages, called *heartbeat messages*, for behavior (b_i^j), which is still activated with its identifier (j). This involves two cases.

i) When an agent (k-th) receives a heartbeat message with regard to behavior (b_i^j) from another agent (j-th), it retains the weight (w_i^j) of the behavior (b_i^j) in its second database.

ii) When an agent (k-th) does not receive any heartbeat messages with regard to behavior (b_i^j) from another agent (j-th) for a specified time, it automatically decreases the weight (w_i^j) of the behavior (b_i^j) in its second database, and resets the weight (w_i^k) of the behavior (b_i^k) to the initial value or increases the weight (w_i^k) in its first database (Figure 5).

Note that behavior b_i^k is provided by the k-th agent and behavior b_i^j is provided by the j-th agent. The weights of behaviors provided by other agents automatically decrease without any heartbeat messages from the agents. Therefore, when an agent terminates or fails, other agents decrease the weights of the behaviors provided by the agent. If they have the same or compatible behaviors, they can then activate the behaviors, which may be inactivated. After a request

[1] Figures 2, 3, and 4 assume W^A and W^B to be ten.

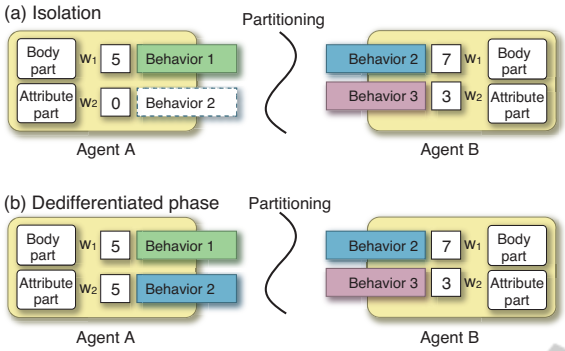

Figure 5: Differentiation-based adaptation on other sides.

message is sent to another agent, if the agent waits for the result to arrive for longer than a specified time, it selects one of the agents that can handle the message from its database and requests the selected agent. If there are no agents that can provide the behavior that can handle the behavior quickly, it promotes other agents that have the behavior in less-developed form.

5 EVALUATION

In the current implementation, each runtime system is constructed as a middleware system with Java. It is responsible for executing agents and for exchanging messages in runtime systems on other computers through a network. When a runtime system is (re)connected to a network, it multicasts heartbeat messages to other runtime systems to advertise itself, including its network address in a plug-and-play protocol manner. Adaptation messages, i.e., restraining and heartbeat messages, are transmitted as multicast UDP packets, which are unreliable. Application-specific messages, i.e., request and reply, are implemented through TCP sessions as reliable communications.

Let us suppose that a sensor-network system consisting of 15 $\times$ 15 nodes is connected through a grid network, as shown in Figure 6. Each node can communicate with its four neighboring nodes and the diameter of a circle in each node represents the weight of a behavior. The system consists of 225 runtime systems running on twenty-five PCs (Intel Core 2 Duo 1.8 GHz, MacOS X 10.6, and Java Runtime Environment ver.6) connected via a 1-GbE switch, where each PC executes nine runtime systems.[2] Each runtime provided nine runtime systems running on different Java VMs. These runtime systems, called *nodes* in this section, were connected according to the topology of the target grid network and could multicast to four neighboring runtime systems through the grid network (Figure 6 (i)). We also assumed that each node had two photosensors for two different wavelengths, called lights A and B. Each agent had two behaviors, called behaviors A and B, where the first responded to light in wavelength A and the second to light in wavelength B. Agents were distributed at all nodes. To emulate photosensors, we also deployed two (non-differentiated) agents as pseudo-sensors agents at each node, where such agents periodically notified their runtime systems of events every second. Some of the following experiments explicitly limited the reachability of messages for differentiation-based adaptation, e.g., restraining and heartbeat messages within UDP multicast domains.

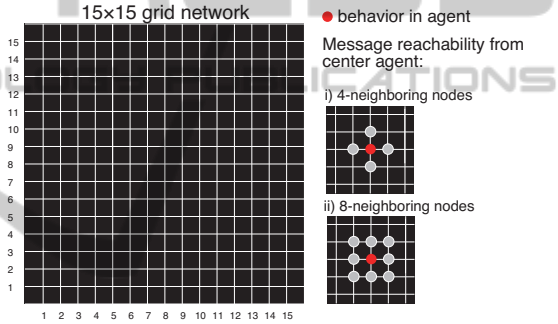

Figure 6: Visualization of Behavior weights in agents on grid network.

We assumed light was irradiated in one wavelength to all nodes in the first experiment and each node could only receive messages from at most four-neighboring nodes (Figure 6 (i)). Two behaviors of every agent were initially inactive because there was no light. When we started pseudo-photosensor agents for light A to emulate the irradiation of light A, all agents activated behavior A (Figure 7 (i)). After five seconds, some agents were more developed and the others were less developed (Figure 7 (ii)). As a result, the weights of behavior-A in agents were speckled and unstable at nodes. Ten seconds after this, the weights of the behaviors converged and stabilized in a grill shape (Figure 7 (iii)). When we stopped irradiating light A, agents diverged and then inactivated. This experiment proved that our approach enabled software-level adaptations to converge on their proper differentiated states.

We changed the reachability of messages to eight

[2]In this experiment, there were no serious differences between this system and 225 servers, because our approach was not processor- or memory- intensive. Since the loss rate of UDP packets was less than 5%, we could ignore collisions on the grid network in our performance evaluation in the experiment.

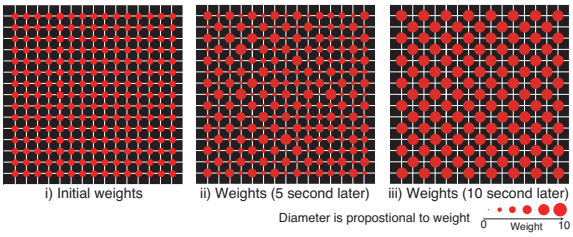

Figure 7: Behavior weights in agents with the whole stimulus on grid network.

neighboring nodes (Figure 6 (ii)). Like the first experiment, we irradiated light in one wavelength to all nodes. Figure 8 (i) and (ii) show the weights of behaviors in agents ten and twenty seconds later. Even though differentiated behaviors were uneven, they could be placed within certain intervals, i.s., two edges on the grid network. This proved that our approach was useful in developing particular functions of software components at nodes.

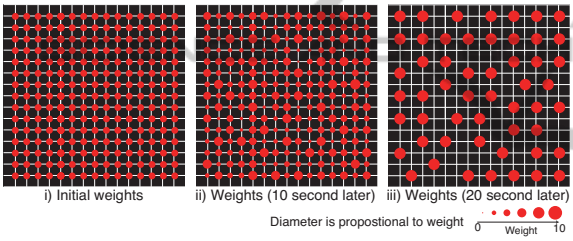

Figure 8: Behavior weights in agents with different message reachability.

We still assumed the reachability of messages eight neighboring nodes (Figure 6 (ii)). Figure 9 (i) was the initial weights of agents on the network. We explicitly made a flawed part in the network (Figure 9 (ii)). Some agents dedifferentiate themselves in nodes when a flawed part made in the network. In the experiment agents around the hole started to activate themselves through dedifferentiation. The weights of their behaviors converged according to the weights of their behaviors to the behaviors of other newly activated agents in addition to existing agents. Finally, some agents around the hole could support the behaviors on behalf of the dismissed agents with the flawed part. This result prove that our approach could remedy such a damage appropriatively in a self-organized manner. This is useful managing CPSs for sensing catastrophes, e.g., earthquakes and deluges.

6 CONCLUSIONS

This paper proposed an approach to adapting software components on CPSs. It is unique to other existing software adaptations in introducing the notions of (de)differentiation and cellular division in cellular slime molds, e.g., dictyostelium discoideum, into software components. When a software component delegates a function to another component, if the former has the function, its function becomes less-developed and the latter's function becomes well-developed. The approach was constructed as a middleware system on CPSs instead of any simulation-based systems. Components can be composed from Java objects written in JavaBean. We described several evaluations of the approach with a practical application.

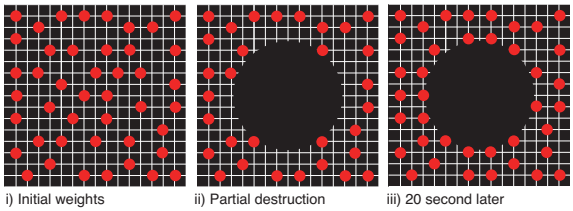

Figure 9: Adaption to flawed part in network.

REFERENCES

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

Cheng, S., Garlan, D., and Schmerl, B. (2006). Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of International Workshop on Self-adaptation and Self-managing Systems (SEAMS'2006)*, pages 2–8. ACM Press.

Dorigo, M. and Stutzle, T. (2004). *Ant Colony Optimization*. MIT Press.

Georgiadis, I., Magee, J., and Kramer, J. (2003). Self-organising software architectures for distributed systems. In *Proceedings of 1st Workshop on Self-healing systems (WOSS'2002)*, pages 33–38. ACM Press.

Herrman, K. (2008). *Self-organizing Ambient Intelligence*. VDM.

Satoh, I. (2007). Self-organizing software components in distributed systems. In *Proceedings of 20th International Conference on Architecture of Computing Systems (ARCS'2007)*, volume 4415 of *Lecture Notes in Computer Science (LNCS)*, pages 185–198. Springer.

Satoh, I. (2011). Cellular differentiation-based service adaptation. In *Proceedings of International Conference on Service Oriented Computing*. Springer.

Satoh, I. (2012). Resilient architecture for sensor networks. In *Proceedings of 9th International Conference on Networked Sensing Systems (INSS'2012)*. IEEE Computer Society.