Towards self-organizing Kalman filters

Joris Sijs
TNO Technical Sciences
Den Haag, The Netherlands
Email: joris.sijs@tno.nl

Zoltan Papp
TNO Technical Sciences
Den Haag, The Netherlands
Email: zoltan.papp@tno.nl

***Abstract*—Distributed Kalman filtering is an important signal processing method for state estimation in large-scale sensor networks. However, existing solutions do not account for unforeseen events that are likely to occur and thus dramatically changing the operational conditions (e.g. node failure, communication deterioration). This article presents an integration solution for distributed Kalman filtering with distributed self-organization to cope with these events. An overview of existing methods on both topics is presented, followed by an empirical case study of a self-organizing sensor network for observing the contaminant distribution process across a large area in time.**

I. Introduction

A current trend in estimation is to connect different sensor nodes via a scalable network topology and thereby, create "on-the-fly" (ad-hoc) networks for monitoring large-area processes with a high spatial accuracy. This trend is mainly a result from the widely available sensor nodes for setting-up such a (wireless) sensor network. However, sensor networks exhibit special features, which makes their application challenging. For one, sensor nodes are typically battery powered and/or have an energy scavenging device to acquire the consumed energy. In addition, due to power and space considerations, the onboard computational and communication capabilities of sensor nodes are seriously limited. Yet, they are frequently deployed in harsh, even hostile environment, where node failures are not exceptions but part of the normal operation. For example, the changing environment in which nodes exchange data induces dynamics in the network capacity and call for adaptive communication strategies to assure the availability of communication resources. These aspects make the application development extremely challenging, as a stable and predictable system performance should be delivered on a dynamically changing configuration with computational, communication and energy constraints. Distributed data interpretation algorithms with reconfiguration capabilities constitute an important and promising way to address these challenges.

In the following, the state estimation problem (based on Kalman filtering) is investigated in this context. Several distributed solutions addressing the Kalman filtering problem have been explored and aim to make use of the local processing elements that are already present in each node. Such *distributed* Kalman filters typically process the sensor measurements locally at each node rather then communicating them to a single, central node. Characteristic approaches to distributed Kalman filtering are presented in [1]–[4] and the references therein. The proposed methods perform a modified Kalman filtering algorithm locally in each node for computing a local estimate of the global state (i.e. forming a network of Kalman filters). Nodes improve their estimation results by exchanging measurements and/or local estimates with other nodes in the network. However, the available distributed Kalman filters assume a fixed topology and communication strategy. Hence, they cannot cope with the unforeseen operational events that are typical for sensor networks, nor do they address deliberate reconfigurations of the networked system during operation.

To solve this issue, signal processing algorithms running in large-scale sensor networks should exhibit robustness with respect to changing operational conditions and anticipated failure modes. Building in redundancy may result in complex, power hungry and prohibitively expensive solutions. A feasible alternative is to deploy the network with self-organizing capabilities: runtime reconfiguration enables to follow changes in the operational conditions of the sensor network, while assuring optimal use of available resources.

The contribution of this article is integrating solutions for distributed Kalman filtering with a framework of distributed self organization. As such, nodes employs a modified Kalman filter locally and, in addition, perform a management procedure that supports the network of Kalman filters to establish self-organization. To that end, existing solutions on distributed Kalman filtering are summarized into a generalized local estimation approach employed by individual nodes for estimating the state. Further, a supportive management procedure is designed that - depending on the available resources - can choose the local estimation algorithms and its parametrization to assure robustness and performance under wide range of operational conditions. The proposed network of self-organizing Kalman filters is further analyzed in an illustrative case-study.

II. Notation and preliminaries

$\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{Z}$ and $\mathbb{Z}_+$ define the set of real numbers, non-negative real numbers, integer numbers and non-negative integer numbers, respectively. For any $\mathcal{C} \subset \mathbb{R}$, let $\mathbb{Z}_{\mathcal{C}} := \mathbb{Z} \cap \mathcal{C}$. The notation 0 is used to denote either zero, the null-vector or the null-matrix of appropriate dimensions, while I_n denotes the $n \times n$ identity matrix. The transpose, inverse and determinant of a matrix $A \in \mathbb{R}^{n \times n}$ are denoted as $A^\top$, A^{-1} and $|A|$, respectively. Further, $A^{\frac{1}{2}}$ denotes the Cholesky decomposition of a matrix $A^{n \times n}$ (if it exists). Given that a random vector $x \in \mathbb{R}^n$ is Gaussian distributed, denoted as $x \sim G(\mu, \Sigma)$, then $\mu \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n \times n}$ are the *mean* and *covariance* of x.

III. Problem formulation

Let us consider a linear process that is observed by a sensor network with the following description:

The networked system consists of N sensor nodes, in which a node $i \in \mathcal{N}$ is identified by a unique number within $\mathcal{N} := \mathbb{Z}_{[1,N]}$. The set $\mathcal{N}_i \subseteq \mathcal{N}$ is defined as the collection of *neighboring* nodes $j \in \mathcal{N}$ that exchange data with node i.

The dynamical process measured by each node $i \in \mathcal{N}$ is described with discrete-time process model, for some local sampling time $\tau_i \in \mathbb{R}_{>0}$ and some k_i-th sample instant, i.e.,

$$\begin{aligned} x[\mathsf{k}_i] &= A_{\tau_i} x[\mathsf{k}_i{-}1] + w[\mathsf{k}_i{-}1], \\ y_i[\mathsf{k}_i] &= C_i x[\mathsf{k}_i] + v_i[\mathsf{k}_i]. \end{aligned} \tag{1}$$

The state and local measurement are denoted as $x \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^{m_i}$, respectively, while process-noise $w \in \mathbb{R}^n$ and measurement-noise $v_i \in \mathbb{R}^{m_i}$ follow the Gaussian distributions $w[\mathsf{k}_i] \sim G(0, Q_{\tau_i})$ and $v_i[\mathsf{k}_i] \sim G(0, V_i)$, for some $Q_{\tau_i} \in \mathbb{R}^{n\times n}$ and $V_i \in \mathbb{R}^{m_i \times m_i}$. A method to compute the model parameters A_{τ_i} and Q_{τ_i} from a corresponding continuous-time process model $\dot{x} = Fx + w$, yields

$$\begin{aligned} &A_{\tau_i} := e^{F\tau_i} \quad \text{and} \quad Q_{\tau_i} := B_{\tau_i} cov\big(w(t{-}\tau_i)\big) B_{\tau_i}^\top, \\ &\text{with} \qquad B_{\tau_i} := \int_0^{\tau_i} e^{F\eta} d\eta. \end{aligned}$$

The goal of the sensor network is to compute a local estimate $x_i \in \mathbb{R}^n$ of the global state x in each node i. Since the process model is linear and both noises are Gaussian distributed, it is appropriate to assume that the random variable $x_i[\mathsf{k}]$ is Gaussian distributed as well, i.e., $x_i[\mathsf{k}_i] \sim G(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i])$ for some *mean* $\hat{x}_i[\mathsf{k}_i] \in \mathbb{R}^n$ and *error-covariance* $P_i[\mathsf{k}_i] \in \mathbb{R}^{n\times n}$. To that extent, each node i performs a local estimation algorithm for computing x_i based on its local measurement y_i and on the data shared by its neighboring nodes $j \in \mathcal{N}_i$. Existing methods on distributed Kalman filtering present an a priori solution for computing x_i and predefine what variables should be exchanged, at what time and with which nodes, e.g. [1]–[4]. Hence, for a given sensor network, a matched (static) estimation procedure is derived under predefined conditions. However, static approaches are not feasible for (large-scale) sensor network applications, due to operational events likely to occur in the system. Solutions should thus be in place that enables the state estimating network to cope with these dynamically raising challenges by a reconfiguration of the communication network during operation, accompanied by a re-evaluation on the available local estimation algorithms. These aspect can be addressed by self organizing methods, in which a feasible solution for (unforeseen) operational events is sought for during operation of a network rather than prior to its deployment.

The problem addressed is integrating state-of-art results in distributed Kalman filtering (DKF) with a management layer for creating a self-organizing networked system, see also Figure 1. The proposed solution derives a generalized set-up for a local estimation algorithm from existing DKF solutions, which is supported by the management layer. The management layer is responsible for choosing specific estimation and communication approaches, so to assure coherent operational conditions depending on the available resources of that particular node (communication, computational and battery-level). Let us start with a description of the local estimation algorithm.

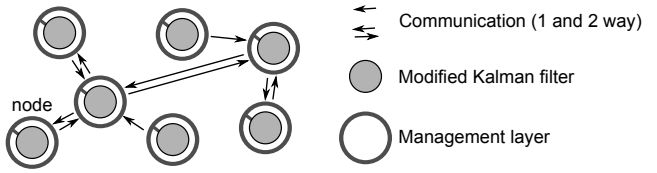


Fig. 1. A network of Kalman filters with supporting management layer to realize the self-organizing property for estimating the state.

IV. Distributed Kalman filtering

The algorithmic set-up for computing the local estimate x_i is a generalization of existing DKF solutions. Typically, these solutions propose that each node i performs a Kalman filter based on its local measurement y_i and thereby, establishes an initial estimate for $x_i \sim G(\hat{x}_i, P_i)$. See, for example, the methods proposed in [1]–[4] and some overview articles in [5], [6]. Additionally, various solutions were proposed to improve this initial local estimate x_i by exchanging data with the neighboring nodes $j \in \mathcal{N}_i$, i.e.,

- Share local measurements,, i.e., node i receives y_j for all $j \in \mathcal{N}_i$, which can be merged with a Kalman filter;
- Share local estimates, i.e., node i receives $x_j \sim G(\hat{x}_j, P_j)$ for all $j \in \mathcal{N}_i$, which can be merged with consensus or fusion methods.

Based on the currently available DKF methods a generalized local estimation algorithm can be designed relying on the node's local measurement and on the data received from neighboring nodes. Each functionality of this generalized set-up is characterized by a specific algorithm, though it is not necessary to specify them prior to deployment. Instead, nodes are deployed with a number of suitable algorithms for each functionality, from which a selection can be made during operation. The alternative algorithms related to DKF are presented next. It is assumed that nodes implements some form of synchronization among their local sampling instants τ_i according to a fundamental sampling time $\tau \in \mathbb{R}_+$ and a localized (time-dependent) scalar $a_i(t) \in \mathbb{Z}_{>1}$, i.e.,

$$\begin{aligned} t_{0_i} &= t_{0_j}, \qquad \forall\, i, j \in \mathcal{N} \\ \tau_i &= a_i(t) \cdot \tau, \quad \forall\, t \in \mathbb{R}_+,\ i \in \mathcal{N} \end{aligned}$$

The above characterization implies that any two nodes i and j in the network have the same initial sampling instant and that the local sampling time τ_i, for any node $i \in \mathcal{N}$, is equal to a multiple of τ. The corresponding multiplication factor $a_i(t)$ is determined by the management layer of node i.

A. Generalized local estimation algorithm

Figure 2 depicts a schematic set-up of the generalized local estimation algorithm, followed by a detailed description of the set-up. It shows that $x_i \sim G(\hat{x}_i, P_i)$ is computed by merging local measurements in a Kalman filtering function prior to merging local estimates in a synchronization/fusion function.

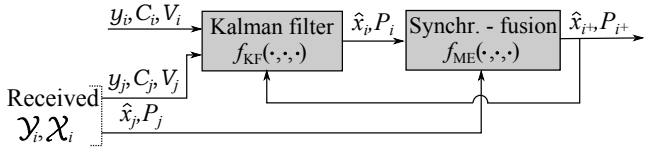


Fig. 2. Schematic set-up of the generalized local estimation algorithm performed by each node i in the network.

Note that the measurement model $y_j[\mathsf{k}_i] = C_j x[\mathsf{k}_i] + v_j[\mathsf{k}_i]$ should be available to node i before $y_j[\mathsf{k}_i]$ can be exploited. Therefore, if node j shares the local measurement, (y_j, C_j, V_j) should be exchanged, while if node j shares local estimates, it exchanges $(\hat{x}_j, P_j)$. Hence, the received data of Figure 2, yields

- $\mathbb{Y}_i \subset \mathbb{R}^{m_j} \times \mathbb{R}^{m_j \times n} \times \mathbb{R}^{m_j \times m_j}$ is the collection of (y_j, C_j, V_j) received by node i from neighboring nodes $j \in \mathcal{N}_i$. Note that $\mathbb{Y}_i$ could be empty, for example, when none of the nodes $j \in \mathcal{N}_i$ shares its local measurement;
- $\mathbb{X}_i \subset \mathbb{R}^n \times \mathbb{R}^{n \times n}$ is the collection of $(\hat{x}_j, P_j)$ received by node i from its neighboring nodes $j \in \mathcal{N}_i$. Similar as to $\mathbb{Y}_i$, also $\mathbb{X}_i$ can be an empty collection.

Let us continue with a detailed description of the Kalman filtering function in Figure 2. This function merges the $y_i[\mathsf{k}_i]$ with the received measurements $y_j[\mathsf{k}_i]$ to update its local estimate $x_{i^+}[\mathsf{k}_i{-}1] \sim G(\hat{x}_{i^+}[\mathsf{k}_i{-}1], P_{i^+}[\mathsf{k}_i{-}1])$. Measurements can merged via the original Kalman filter or by the alternative Information filter, which was proposed in [1]. For this latter approach, measurements are rewritten into their *information form*, for some $z_j \in \mathbb{R}^n$ and $Z_j \in \mathbb{R}^{n \times n}$, i.e.,

$$z_j[\mathsf{k}_i] := C_j^\top V_j^{-1} y_j[\mathsf{k}_i] \quad \text{and} \quad Z_j[\mathsf{k}_i] := C_j^\top V_j^{-1} C_j.$$

The Information filter has similar results as the original Kalman filter but differs in computational demand. Furthermore, since its corresponding implementation is more convenient when the amount of received measurements $(y_j, C_j, V_j) \in \mathbb{Y}_i$ varies at sample instants, the Kalman filtering function in the set-up of Figure 2 employs the Information filter. Let us introduce this function as $f_{\text{KF}}(\cdot,\cdot,\cdot)$, i.e., having three inputs, along with the following characterization:

$$\begin{aligned}
&\underline{(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i]) := f_{\text{KF}}(\hat{x}_{i^+}[\mathsf{k}_i{-}1], P_{i^+}[\mathsf{k}_i{-}1], \mathbb{Y}_i[\mathsf{k}_i])} \\
&\quad M_i = A_{\tau_i} P_{i^+}[\mathsf{k}_i{-}1] A_{\tau_i}^\top + Q_{\tau_i}, \\
&P_i[\mathsf{k}_i] = \Big(M_i^{-1} + Z_i[\mathsf{k}_i] + \sum_{(y_j, C_j, V_j) \in \mathbb{Y}_i[\mathsf{k}_i]} Z_j[\mathsf{k}_i] \Big)^{-1}, \\
&\hat{x}_i[\mathsf{k}_i] = P_i[\mathsf{k}_i] \Big(M_i^{-1} A_{\tau_i} \hat{x}_{i^+}[\mathsf{k}_i{-}1] + z_i[\mathsf{k}_i] + \sum_{(y_j, C_j, V_j) \in \mathbb{Y}_i[\mathsf{k}_i]} z_j[\mathsf{k}_i] \Big).
\end{aligned}$$

The other function of Figure 2, introduced as $f_{\text{ME}}(\cdot,\cdot,\cdot)$, merges the local estimate $x_i[\mathsf{k}_i]$ with the received estimation variables $(\hat{x}_j, P_j) \in \mathbb{X}_i[\mathsf{k}_i]$ into a new estimate $x_{i^+}[\mathsf{k}_i] \sim G(\hat{x}_{i^+}[\mathsf{k}_i], P_{i^+}[\mathsf{k}_i])$. Please note that it is not obliged for a node to perform a fusion function, i.e., nodes can employ $x_{i^+} = x_i$ instead, for example, to save computational power and thus energy. Recent DKF solutions, such as the ones presented in [2], [3], adopt a synchronization approach to characterize $f_{\text{ME}}(\cdot,\cdot,\cdot)$. However, such approaches do not merge the error-covariances of the different estimates and are therefore not employed in this set-up. Instead, $f_{\text{ME}}(\cdot,\cdot,\cdot)$ follows a fusion approach. Fusion methods typically define a fusion function, denoted as $\Omega(\cdot,\cdot,\cdot,\cdot)$, to merge two *prior estimates* x_i and x_j. Fusion of *multiple* estimates (> 2) can be conducted recursively according to their order of arrival at the corresponding node. This means that the merging function $f_{\text{ME}}(\cdot,\cdot,\cdot)$ performed by a node i has the following description:

$$\begin{aligned}
&\underline{(\hat{x}_{i^+}[\mathsf{k}_i], P_{i^+}[\mathsf{k}_i]) := f_{\text{ME}}\left(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i], \mathbb{X}_i[\mathsf{k}_i]\right)} \\
&\quad \text{for each estimate } (\hat{x}_j[\mathsf{k}_i], P_j[\mathsf{k}_i]) \in \mathbb{X}[\mathsf{k}_i], \text{ do} \\
&\qquad (\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i]) = \Omega\left(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i], \hat{x}_j[\mathsf{k}_i], P_j[\mathsf{k}_i]\right), \\
&\quad \text{end for} \\
&\quad \hat{x}_{i^+}[\mathsf{k}_i] = \hat{x}_i[\mathsf{k}_i], \quad P_{i^+}[\mathsf{k}_i] = P_i[\mathsf{k}_i].
\end{aligned}$$

Note that the above merging function still needs suitable a fusion methods to characterize $\Omega(\cdot,\cdot,\cdot,\cdot)$. An optimal fusion methods was presented in [7], though it requires that correlation of the two prior estimates is available. In (self-organizing) sensor networks one cannot impose such a requirement, as it amounts to keeping track of shared data across the entire network. Alternative fusion methods that can cope with an unknown correlation are covariance intersection (CI) and ellipsoidal intersection (EI), as proposed in [8] and [9], respectively. In CI the fusion function is characterized as a convex combination of the two prior estimates x_i and x_j, for some scalar weight $\omega_{ij} \in \mathbb{R}_+$, i.e.,

$$\begin{aligned}
&\underline{\text{CI: } (\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i]) = \Omega\left(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i], \hat{x}_j[\mathsf{k}_i], P_j[\mathsf{k}_i]\right)} \\
&\quad \Sigma_i = \left((1-\omega_{ij}) P_i^{-1}[\mathsf{k}_i] + \omega_{ij} P_j^{-1}[\mathsf{k}_i]\right)^{-1}, \\
&\hat{x}_i[\mathsf{k}_i] = \Sigma_i \big((1-\omega_{ij}) P_i^{-1}[\mathsf{k}_i] \hat{x}_i[\mathsf{k}_i] + \omega_{ij} P_j^{-1}[\mathsf{k}_i] \hat{x}_j^{-1}[\mathsf{k}_i]\big), \\
&P_i[\mathsf{k}_i] = \Sigma_i.
\end{aligned}$$

The fusion method EI results in a "smaller" error-covariance after fusion compared to CI, as the fusion result is not a convex combination of prior estimate. Instead, EI finds an explicit expression of the (unknown) correlation before the merging the independent parts of x_i and x_j via algebraic fusion formulas. To that extent, the (unknown) correlation is characterized by a *mutual covariance* $\Gamma_{ij} \in \mathbb{R}^{n \times n}$ and a *mutual mean* $\gamma_{ij} \in \mathbb{R}^n$, which then results in the following fusion function $\Omega(\cdot,\cdot,\cdot,\cdot)$:

EI: $(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i]) = \Omega\left(\hat{x}_i[\mathsf{k}_i], P_i[\mathsf{k}_i], \hat{x}_j[\mathsf{k}_i], P_j[\mathsf{k}_i]\right)$

$$\Sigma_i = (P_i^{-1}[\mathsf{k}_i] + P_j^{-1}[\mathsf{k}_i] - \Gamma_{ij}^{-1})^{-1},$$
$$\hat{x}_i[\mathsf{k}_i] = \Sigma_i\big(P_i^{-1}[\mathsf{k}_i]\hat{x}_i[\mathsf{k}_i] + P_j^{-1}[\mathsf{k}_i]\hat{x}_j^{-1}[\mathsf{k}_i] - \Gamma_{ij}^{-1}\gamma_{ij}\big),$$
$$P_i[\mathsf{k}_i] = \Sigma_i.$$

The mutual mean γ_{ij} and mutual covariance Γ_{ij} are found by a singular value decomposition, which is denoted as $[S, D, S^{-1}] = svd(\Sigma)$ for a positive definite $\Sigma \in \mathbb{R}^{n\times n}$, a diagonal $D \in \mathbb{R}^{n\times n}$ and a rotation matrix $S \in \mathbb{R}^{n\times n}$. As such, let us introduce the matrices $D_i, D_j, S_i, S_j \in \mathbb{R}^{n\times n}$ via the singular value decompositions $[S_i, D_i, S_i^{-1}] = svd(P_i[\mathsf{k}_i])$ and $[S_j, D_j, S_j^{-1}] = svd(D_i^{-\frac{1}{2}} S_i^{-1} P_j[\mathsf{k}_i] S_i D_i^{-\frac{1}{2}})$. Then, an algebraic expression of γ_{ij} and Γ_{ij}, for some $\varsigma \in \mathbb{R}_+$ while $\{A\}_{qr} \in \mathbb{R}$ denotes the element of a matrix A on the q-th row and r-th column, yields

$$D_{\Gamma_{ij}} = \text{diag}\big(\max[1, \{D_j\}_{11}], \cdots, \max[1, \{D_j\}_{nn}]\big),$$
$$\Gamma_{ij} = S_i D_i^{\frac{1}{2}} S_j D_{\Gamma_{ij}} S_j^{-1} D_i^{\frac{1}{2}} S_i^{-1},$$
$$\gamma_{ij} = \big(P_i^{-1} + P_j^{-1} - 2\Gamma^{-1} + 2\varsigma I_n\big)^{-1} \times \big((P_j^{-1} - \Gamma^{-1} + \varsigma I_n)\hat{x}_i + (P_i^{-1} - \Gamma^{-1} + \varsigma I_n)\hat{x}_j\big).$$

A suitable value of ς follows: $\varsigma = 0$ if $|1 - \{D_j\}_{qq}| > 10\epsilon$, for all $q \in \mathbb{Z}_{[1,n]}$ and some $\epsilon \in \mathbb{R}_{>0}$, while $\varsigma = \epsilon$ otherwise. The design parameter ϵ supports a numerically stable result.

This completes the description of the estimation algorithm depicted in Figure 2. A node i merges received measurement in $\mathbb{Y}_i$ with the Kalman filtering function f_{KF}, while received estimates in $\mathbb{X}_i$ are merged with f_{ME} via either CI or EI. Which functionalities are performed and what local variables are shared with other nodes is decided by the management layer. However, to make rational decisions the imposed communication and computational requirements of these different options should be available and are therefore presented, next.

V. Required resources

Important resources in sensor networks are communication and computation, as they both can be translated to the use of energy and time. The previously addressed estimation algorithm performed by each node is assessed on its requirement with respect to these two resources. To that extent, it is assumed that nodes could decide to share their local measurement in the original or in the information form. The resulting communication requirements of node i for the different data packages are then listed in Table I, while the order in computational demand for the different functionalities is presented in Table II, for some $M_i := m_i + \sum_{(y_j, C_j, V_j)\in\mathbb{Y}_i} m_j$.

VI. Self-organizing solutions

The design challenge of any embedded system is to realize the given functionalities (in this case state estimation) on a given hardware platform while satisfying a set of non-functional requirements, such as response times, dependability, power efficiency, etc. Model-based system design has been proven to be a successful methodology for supporting the system design process [10]. Model-based methodologies use multiple models to capture the relevant properties of the design. These models can then be used for various purposes, such as automatic code generation, design optimization, system evolution and so on. Crucial for the design process are the interactions between the different models, which can be expressed as constraints, dependencies, etc.

There are various models that formalize design consideration, such as requirements, components, constraints, to characterize the design space. Two fundamental models of the design are emphasized here: the task model (capturing the required functionalities of the employed signal processing method) and the physical model (capturing the hardware configuration of the implementation). Figure. 3 illustrates the use of the models in the design process.

TABLE I
Required communication in the amount of floating points exchanged by node i depending on which data is shared.

exchanged data	communication demand
(y_i, C_i, V_i)	$m_i^2 + 2m_i + n$
(z_i, Z_i)	$n^2 + n$
$(\hat{x}_i, P_i)$	$n^2 + n$

TABLE II
Computational demand in the order of floating points operations per employed functionality.

functionality	computational demand
f_{KF}	$\approx O(3n^3 + M_i n^2 + nM_i^2)$
Ω according to CI	$\approx O(3n^3 + 9n^2)$
Ω according to EI	$\approx O(31n^3 + 7n^2)$

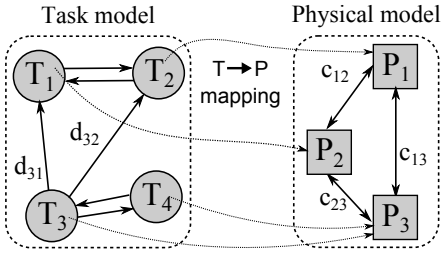


Fig. 3. Modeling of signal processing and implementation, in which each aspect is characterized with some properties, e.g., memory and execution speed for P_i (processing); bit-rate and latency for c_{ij} (communication), update rate and ♯instructions for T_q (task); message size and update rate for d_{qr} (interaction).

The task model in this figure is represented as directed a graph, the signal processing components (tasks) are represented by the vertices of the graph, while their data exchange (interactions) are represented by the edges. Both the tasks as well as the interactions are characterized by a set of properties, which typically reflect non-functional requirements/properties. The tasks run on a connected set of processors, represented

by the physical model of the system. The components in this simplified physical model are the computing nodes and the communication links.

The design process involves finding a particular mapping that defines the assignment of a task T_q to a processor P_i, i.e., it determines which task runs on which node. Obviously the memory and execution time requirements define constraints when assigning the tasks to nodes. Further, data exchange between tasks makes the assignment problem more challenging in distributed configurations, as a task assignment also defines the use of communication links c_{ij}- and the communication links have limited capabilities. The design process results in a sequence of decisions, which lead to a feasible system design. Traditionally the design process is "offline" (*design time & static*), i.e., it is completed before the implementation and deployment of the system itself. The task model, the hardware configuration and their characteristics are assumed to be known during this design time and the design uncertainties are assumed to be low.

Sensor deterioration, node failure, unreliable communication, depleted batteries, etc., are not exceptions but manifest themselves as common operational events. These events result in changes in the system configuration, as captured by the physical model, due to which implementations relying on static designs fail to deliver as specified.

A resource conscious way to cope with this problem is to assure that the sensor network can "follow" those system changes and "adjust" its internals to deliver (at least a predefined subset of) the assigned functionalities as far as it is feasible ("graceful degradation" property). This approach requires runtime reconfiguration capabilities and has significant impact both on system design and on the runtime operation of the system. Conceptually the system design process is not completely finished in design time but a set of design alternatives are provided for execution. During operation - depending on the health state of the configuration and its environmental conditions - a selection is made automatically to assure an optimal use of available resources.

Expressed in the concepts of Figure 3, the runtime reconfiguration is carried out via changing the task graph (i.e. selecting a different signal processing scheme, changing certain parameters of the processing blocks, etc.) or re-mapping the task graph to the physical model (i.e. changing the task assignment with the consequential change in the communication topology). As Figure 1 already indicates, our goal is to realize the reconfiguration functionality for distributed state estimation in a distributed manner to improve robustness and scalability. Figure 4 details the state estimation nodes of Figure 1. The *SE* component implements the node's state estimation algorithm, i.e. forming the core signal processing functionality. The *SE* components do not interact with each other directly but via the *SHELL*, which is responsible for assuring the optimal use of the system resources. The two main functions of this *SHELL* are the communication service (C) and management service (M). C acts as a switchboard, i.e. maintains connections among the SEs running on different nodes. M is the intelligent component, i.e., it controls C and SE according to the information about the system level goals and requirements the relevant estimation performance and current system health readings. The dashed lines represent the status/control interfaces. Note that M has access to the implementation platform for assessing its health state (e.g. remaining battery energy level) and to control platform parameters (e.g. clock speed). Phrasing differently: *SE* is the "number cruncher" without insight into the "big picture"; C is merely a configurable switch, while M has all the "knowledge" about achieving system level goals under optimal use of resources, i.e. it embodies the (certain subset of) design knowledge.

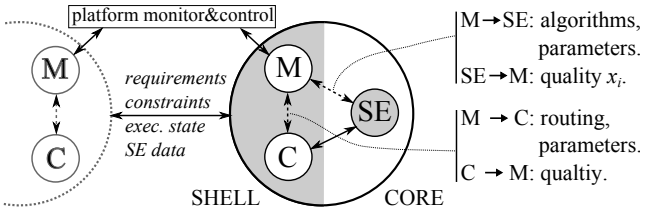


Fig. 4. State estimator node functionalities.

Consequently, the key element for runtime reconfiguration is the management services component M. The concept of a reconfiguration process is illustrated in Figure 5. The reconfiguration process is triggered by observed changes in the embedding environment of the system or in the system itself, e.g., realizing node failure or a low battery status. The "trajectory" for reconfiguration is not predefined but is a result of an optimization process attempting to maximize the "usefulness" of the system, as defined by performance criteria.

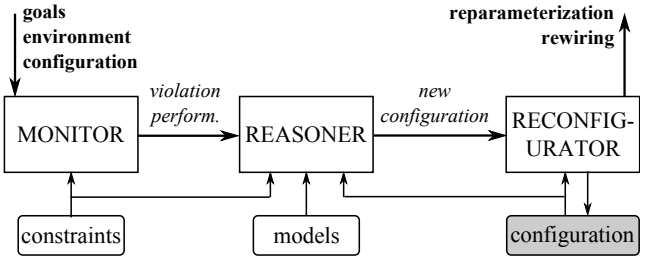


Fig. 5. The reconfiguration process.

The relevant models of the system, such as the task, physical and temporal model, are formalized and stored in a database represented by the $models$ block. The $constraints$ block represents the dependencies in the models and between models. The $MONITOR$ functionality checks if the observed changes result in a violation of certain constraints by the systems or a significant drop in its performance. If the $MONITOR$ concludes that current circumstances prevent the system to perform as requested, then the reconfiguration process is initiated. The $REASONER$ determines a new configuration that satisfies all constraints and provides an acceptable

performance. It should be emphasized the $REASONER$ may not carry out pure logical reasoning but also other types of search and optimization functions depending on the representation describing models, goals, etc. The new configuration is passed to the $RECONFIGURATOR$ functionality to plan and execute the sequence of operations for "transforming" the old into the new configuration in runtime.

The efficient implementation of runtime reconfiguration should address challenges both in representation, monitoring and reasoning. There are no ultimate answers to these questions, consequently the research area of runtime reconfigurable systems design is quickly evolving. Established domains as self-adaptive software systems [11] and dynamically reconfigurable hardware systems [12], [13] provide fundamental contributions. A few characteristic approaches to practical runtime reconfigurability includes model integrated computing (MIC) [10], formalization of the reconfiguration as constraint satisfaction problem [14], [15], multi-aspect modeling as representation combined with aspect oriented programming [16], discover - match - coordinate service oriented architecture scheme with a hierarchical service overlay [17] and object centric paradigm to compose the compound services [18]. [19] describes a model-based solution to validate at runtime that the sensor network functionalities are performed correctly.

Due to the typically large design space that should be explored, combined with the demanding reasoning algorithms, make the distributed real-time implementation difficult and many times unfeasible. To that extent, we adopted a rule-based formalism [20] to express the reconfiguration knowledge (e.g. directed search, optimization, etc.). Rule bases can be preprocessed offline (i.e. compiling the human readable format into executable "machine friendly" format) and small-footprint forward changing inference algorithms (reasoners) enable onboard implementations on relatively low performance platforms. The careful "tailoring" of the rule base, i.e. properly constraining the search space and dedicating the scope of the reconfigurability to the actual case in hand, makes the real-time implementation of runtime reconfiguration possible for a wide spectrum of distributed state estimation problems. A "flavor" of the rule formalism is given via application examples, next.

VII. Case study on a diffusion process

A spatio-temporal 2D diffusion process is considered in the case study for demonstrating and evaluating a self-organizing sensor network with DKF. The goal is to estimate the contaminant's distribution across an area of 1200×1200 meters that results from a contaminant source. As time passes, the corresponding concentration levels $\rho \in \mathbb{R}_+$ across this area change due to diffusion and wind from the North. As such, the contaminant distribution in the area is characterized on a grid with a grid-size of 100 meters, where $\rho^{(q)} \in \mathbb{R}^n$ denotes the concentration level at the q-th grid-point $q \in [1, 144]$ (a grid-point is defined as the center of a particular grid-box). A concentration level $\rho^{(q)}$ depends on similar levels at neighboring grid-points, which are denoted as q_n for north, q_s for south, q_e for east and q_w for west. See also Figure 6 for a graphical representation of these grid-points relative to the q-th grid-point. The *continuous-time* process model of $\rho^{(q)}$, for some $a, a_n, a_s, a_e, a_w \in \mathbb{R}$ and for all $q \in \mathbb{Z}_{[1,144]}$, yields

$$\dot{\rho}^{(q)} = a\rho^{(q)} + a_n\rho^{(q_n)} + a_s\rho^{(q_s)} + a_e\rho^{(q_e)} + a_w\rho^{(q_w)} + u^{(q)}.$$

The variable $u^{(q)} \in \mathbb{R}_+$ parameterizes the production of the contaminant source at grid-point q conform to $u^{(18)} = 75$, $u^{(29)} = 75$, $u^{(30)} = 100$, $u^{(31)} = 100$ and $u^{(42)} = 175$, for all time $t \in \mathbb{R}_+$, while $u^{(q)} = 0$ for all other q. Value of the remaining parameters are chosen to establish a northern wind direction, i.e., $a = \frac{-12}{800}$, $a_n = \frac{1}{800}$, $a_s = \frac{2}{800}$, $a_e = \frac{7}{800}$ and $a_w = \frac{2}{800}$. The resulting concentration levels of the simulated process are depicted in Figure 6.

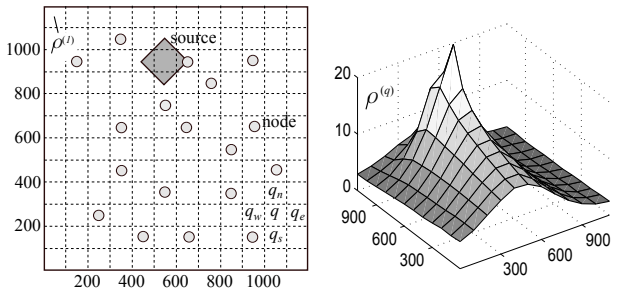


Fig. 6. The monitored area is divided into a grid. Each grid-point q has four neighbors q_n, q_s, q_e and q_w, i.e., one to the north, south, east and west of grid-point q, respectively. The contaminant matter produced by the source spreads through the area due to diffusion and wind.

The concentration levels are to be reconstructed by the deployed sensor network. The network consists of 18 sensor nodes that are randomly distributed across the area, see also Figure 6, for which the position of each node is assumed to be available. Further, the local measurement y_i is equivalent to the concentration level $\rho^{(q)}$ of node i its corresponding grid-point q, for some measurement-noise $v_i \sim G(0, 0.5)$. Each node i performs the local estimation algorithm illustrated in Figure 2 with a supportive management layer. Although wind measurements would improve the estimation results, such information cannot be assumed to be available for nodes in ad-hoc networks, nor knowledge on the chemical source. Therefore, the process-parameters employed by an individual node for state estimation, yield $a = \frac{-12}{800}$, $a_n = \frac{3}{800}$, $a_s = \frac{3}{800}$, $a_e = \frac{3}{800}$ and $a_w = \frac{3}{800}$, while the unknown source is represented by process-noise $w^{(q)} \sim G(0, 2 \cdot 10^3)$, for all $q \in \mathbb{Z}_{[1,144]}$. Further, the state is defined as the collection of all concentration levels, i.e., $x := \begin{pmatrix}\rho^{(1)} & \rho^{(2)} \cdots \rho^{(144)}\end{pmatrix}^\top$. The above description is used to define model parameters A_{τ_i}, Q_{τ_i} and V_i of the *discrete-time* process model in (1) for each node $i \in \mathcal{N}$ and some initial sampling time $\tau_i = 10$ seconds.

Two different types of sensor network configurations are assessed, a hierarchical and a flat set-up, in the presence of (unforseen) operational events. Let us first present the results with respect to self-organization, followed by the estimation performance in time. Note that it is not desired to assess the considered estimators with respect to a centralized solution, as it would result in system requirements that are infeasible.

A. Event driven self-organization

Before presenting the reconfiguration results of the network, let us first describe the two evaluated sensor networks.

- In a *hierarchical* sensor network nodes are given specific tasks so that the network consists of multiple subnetworks, as it is illustrated in Figure 7(a). In each subnetwork nodes exchange their local measurements with the center node of that particular subnetwork (denoted with dashed lines). The center node computes a local estimate based on these received measurements via f_{KF}, after which this estimate is shared with the center nodes of other subnetworks (denoted with the solid lines). The received estimates are then fused with the local estimate according to the merging function f_{ME}.
- The *flat* sensor networks reflects an ad-hoc networked system. Therein, nodes adopt a mesh-network-topology, as it is depicted in Figure 7(c). Since there is no hierarchy, each node computes a local estimate of the state by performing the estimation algorithm depicted in Figure 2, where local estimates are shared with neighboring nodes. Note that no local measurements are shared.

Two operational events will occur in this network, followed by the corresponding action as it is implemented in the reconfiguration process of each node. A rule-based representation formalism is used to define the "knowledge base" of the reconfiguration functionality. For the clarity of the illustrative example, we do not attempt a rigorously formal description of the knowledge base but only the "style" of the rule-based representation is shown. Similarly, instead of explicit optimization, $situation \rightarrow action$ type of reconfiguration activities are shown.

1. *Event*: at $t = 150$ seconds nodes 1, 3 and 8 will cross their critical energy level.

Action: If the critical energy level is crossed, then lower the node's local sampling time from 10 seconds to 20 seconds.

2. *Event*: At $t = 250$ seconds nodes 5 and 11 will break down, which is detected by other nodes in the networks.

Action hierarchical network: If a center node brakes down, then check the energy levels of other all nodes in the subnetwork. The node with the largest batter-level is assigned as the new center node with matching responsibilities.

Action flat network: If a nodes brakes down and the network has lost its connectivity, then establish a network connection with other nodes until this connectivity is re-established. In case this means increasing the communication range to larger distances, decrease the sampling time accordingly.

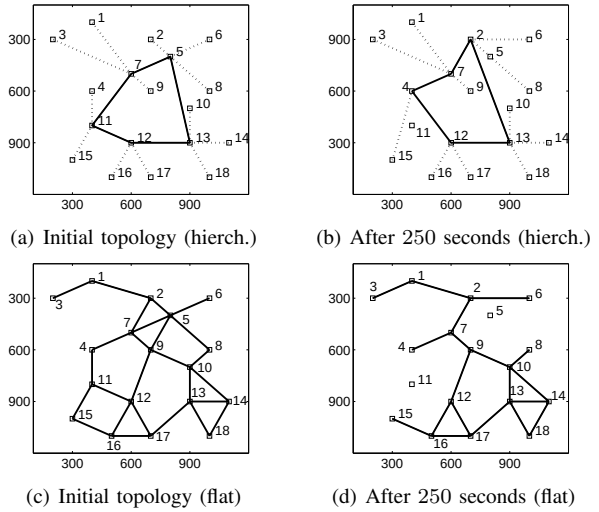
As an example, the rule set below shows the handling of the #2 event in the hierarchical configuration.

```
Rule_2a:
    IF   NEIGHBOR(?x) & TIMEDOUT(?x) &
         ?x.function = centerfun
    THEN set(go_for_newcenter,TRUE)

Rule_2b:
    IF   go_for_newcenter & NEIGHBOR(?x) &
         !TIMEDOUT(?x) &
         max(?x.power) = self.power
    THEN exec(assign,centerfun),
         exec(broadcast,centerfun_msg)
```

The handling of predicates, pattern matching variables, attributes and actions on the right-hand-side of the rules follows "standard" operation of rule-based systems, see e.g. [20].

The results of the above re-organizational rules for both types of network set-ups is depicted in Figure 7



(a) Initial topology (hierch.) (b) After 250 seconds (hierch.)

(c) Initial topology (flat) (d) After 250 seconds (flat)

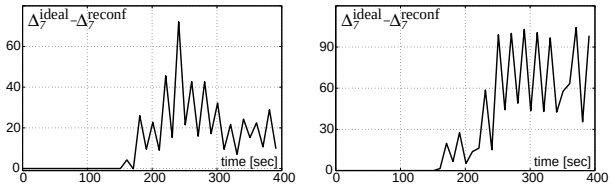
Fig. 7. Communication topology of the hierarchical network, i.e., (a) and (b), and of the flat network, i.e., (c) and (d). A dashed line implies the exchange measurements, wile a solid lines implies the exchange of estimates.

Let us start by analyzing the hierarchical network, for which Figure 7(a) depicts its topology prior to the event of nodes 5 and 11 braking down, while Figure 7(b) illustrates this topology after the event (assuming that the battery of nodes 2 and 4 have the highest battery-level). This figure indicates that nodes 2 and 4 have become responsible for estimating the state and thereby, replace nodes 5 and 11, respectively. In the case of the flat network, Figure 7(c) depicts the corresponding network topology prior to the operational events, while Figure 7(d) illustrates the topology and afterwards. This figure indicates that the sensor network re-establishes the connectivity of the network, also after the event of a node braking down. However, node 6 had to increase its radio power and thus will lower its local sampling time to $\tau_6 = 20$ seconds. Let us continue with an analysis of the estimation results, next.

B. Estimation results

The estimation performance is analyzed according to a particular estimation error, as it is depicted in Figure 8, for which the estimation error of single node i is defined as $\Delta_i := \big(x - \hat{x}_i\big)^\top \big(x - \hat{x}_i\big)$. Then, Figure 8 depicts the difference in the estimation error of a network *not* affected by operational event with the estimation error in a network that *is* affected by the previously presented operational event. The reason that the figure depicts the results of node 7, is because this node affected by both events.

Before Figure 8 is analyzed, let us denote the (hierarchical and flat) network in the *presence* of the above mentioned



(a) Hierarchical network (b) Flat network

Fig. 8. The difference in the estimation error of node 7 for a network *not* affected by operational event minus a network that *is* affected by the previously presented operational event.

operational events as the reconf-case and the same network in the *absence* of operational event as the ideal-case. Then, the figure indicates that the results of the reconf-case and the ideal-case are equivalent until the two operational events occur, which is expected as both network cases are similar until 150 seconds. After that time, the estimation error of node 7 in the reconf-case increases with respect to the ideal-case. This is due to the fact that nodes 1, 3 and 8 double their local sampling times into 20 seconds and thus, node 7 will receive twice as less measurement information from nodes 1 and 3. This leads to an increase in estimation error of node 7 compared to the ideal-case. Further, note that this error decreases when local measurement information from nodes 1 and 3 is received, i.e., at the time instants 170, 190, 210, ..., 370, 390. At these instants, node 7 receives two more local measurements, i.e., y_1 and y_3, which are not received at the other sample instants due to the fact that nodes 1 and 3 have an update rate that is twice as slow. After the second operational event, i.e., nodes 5 and 11 break down at $t = 250$, one can notice a difference between the results of the hierarchical network with respect to the flat one. Although in both cases the estimation error of the reconf-case is still higher than the ideal-case, the figures show that different optimal solutions exists for the two different type of network configurations. Nonetheless, from both cases one can conclude that the networked system is able to estimate the state in multiple nodes of the network, even in the presence of unforseen operational events.

VIII. Conclusions

Distributed Kalman filtering (DKF) is an important signal processing method for state estimation in networked systems. However, existing solutions do not account for operational events that are likely to occur in these systems, especially in (wireless) sensor networks. Therefore, this article presented a first step to combine DKF with existing approaches for establishing a self-organizing sensor network. Empirical studies on these self-organizing sensor networks for state estimation showed promising results that encourage a further investigation towards a network on self-organizing Kalman filters. Future research could thus involve other existing methods to determine the local estimation results at an individual node, or allowing the supportive management layer to determine which state elements should be estimated locally.

References

[1] H. Durant-Whyte, B. Rao, and H. Hu, "Towards a fully decentralized architecture for multi-sensor data fusion," in *1990 IEEE Int. Conf. on Robotics and Automation*, Cincinnati, USA, 1990, pp. 1331–1336.

[2] S. Kirti and A. Scaglione, "Scalable distributed Kalman filtering through consensus," in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Las Vegas, USA, 2008, pp. 2725 – 2728.

[3] A. Ribeiro, I. D. Schizas, S. I. Roumeliotis, and G. B. Giannakis, "Kalman filtering in wireless sensor networks: Reducing communication cost in state-estimation problems," *IEEE Control Systems Magazine*, vol. 4, pp. 66–86, 2010.

[4] J. Sijs and M. Lazar, "Distributed Kalman filtering with global covariance," in *Proc. of the American Control Conf.*, San Francisco, USA, 2011, pp. 4840 – 4845.

[5] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri, "Distributed kalman filtering based on consensus strategies," *IEEE Journal on Selected Areas in Communications*, vol. 26, pp. 622–633, 2008.

[6] J. Sijs, M. Lazar, P. Van de Bosch, and Z. Papp, "An overview of non-centralized Kalman filters," in *Proc. of the IEEE Int. Conf. on Control Applications*, San Antonio, USA, 2008, pp. 739–744.

[7] Y. Bar-Shalom and L. Campo, "The effect of the common process noise on the two-sensor fused-track covariance." *IEEE Trans. on Aerospace and Electronic Systems*, vol. AES-22, no. 6, pp. 803–805, 1986.

[8] S. J. Julier and J. K. Uhlmann, "A non-divergent estimation algorithm in the presence of uknown correlations," in *Proc. of the American Control Conf.*, Piscataway, USA, 1997, pp. 2369–2373.

[9] J. Sijs and M. Lazar, "State fusion with unknown correlation: Ellipsoidal intersection," *Automatica (*in press*)*, 2012.

[10] G. Karsai and J. Sztipanovits, "A model-based approach to self-adaptive software," *Intelligent Systems and their Applications, IEEE*, vol. 14, no. 3, pp. 46 –53, may/jun 1999.

[11] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. e. Magee, "Software engineering for self-adaptive systems: A research roadmap." in *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science, vol. 5525. Springer, 2009, pp. 1–26.

[12] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, 2001, pp. 642 –649.

[13] E. L. d. S. Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for mpsocs," *IEEE Des. Test*, vol. 27, pp. 26–35, September 2010. [Online]. Available: http://dx.doi.org/10.1109/MDT.2010.106

[14] T. Streichert, D. Koch, C. Haubelt, and J. Teich, "Modeling and design of fault-tolerant and self-adaptive reconfigurable networked embedded systems," *EURASIP JOURNAL ON EMBEDDED SYSTEMS*, p. 15, 2006.

[15] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti, "Constraint-guided dynamic reconfiguration in sensor networks," in *Proceedings of the 3rd international symposium on Information processing in sensor networks*, ser. IPSN '04. New York, NY, USA: ACM, 2004, pp. 379–387. [Online]. Available: http://doi.acm.org/10.1145/984622.984677

[16] B. Morin, O. Barais, J.-M. Jzquel, F. Fleurey, and A. Solberg, "Models at runtime to support dynamic adaptation," *IEEE Computer*, pp. 46–53, October 2009. [Online]. Available: http://www.irisa.fr/triskell/publis/2009/Morin09f.pdf

[17] S. Kalasapur, M. Kumar, and B. Shirazi, "Seamless service composition (sesco) in pervasive environments," in *Proceedings of the First ACM International Workshop on Multimedia Service Composition*, ser. MSC '05. New York, NY, USA: ACM, 2005, pp. 11–20. [Online]. Available: http://doi.acm.org/10.1145/1099423.1099428

[18] X. D. Koutsoukos, M. Kushwaha, I. Amundson, S. Neema, and J. Sztipanovits, *OASiS: A service-oriented architecture for ambient-aware sensor networks*, 2007, vol. 4888 LNCS, pp. 125–149.

[19] Y. Wu, K. Kapitanova, J. Li, J. A. Stankovic, S. H. Son, and K. Whitehouse, "Run time assurance of application-level requirements in wireless sensor networks," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '10. New York, NY, USA: ACM, 2010, pp. 197–208. [Online]. Available: http://doi.acm.org/10.1145/1791212.1791236

[20] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice-Hall, Englewood Cliffs, NJ, 2003.