A reconfiguration framework for self-organizing distributed state estimators

Coen van Leeuwen
TNO Technical Sciences
Den Haag, The Netherlands
Email: coen.vanleeuwen@tno.nl

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*—A sensor network operating under changing operational conditions will have to adapt to its environment, topology and system performance. In order to obtain this flexible behavior, a reconfiguration framework is proposed for distributed signal processing solutions. The considered example in this article is distributed Kalman filtering, whereas the reconfiguration framework is based on a first order logic reasoner to find a feasible configuration in a dynamic execution context. In a simulated scenario of a greenhouse temperature field estimation, the proposed system can minimize the state estimation error, while satisfying the systems constraints such as battery life, communication bandwidth or reliability and timeliness of response.**

I. Introduction

Self-organization and self-optimization are promising approaches to address the practical challenges of large-scale sensor nd actuator (control) networks, such us easy deployment, robustness, energy efficiency, etc. The terms self-organization and self-optimization are used to denote a desired system property (also collectively called as reconfiguration): the system is not entirely specified *a priori* to its deployment but some design decisions are postponed to the nominal operation phase of the system. The main purpose of such a property is to gain robustness of the system's performance with respect to operational changes (internal) as well as environmental changes (external). For example, to enable the system to cope with *changing system configurations* (i.e. adding and removing subsystem components) without re-programming the existing set-up (ease-of-deployment), to support a *mobile group* of subsystems observing particular areas (dynamic sensor management), to *adjust on environmental changes* affecting the communication resource (changing network capacities) and to *adapt to a variety of system goals* during operation depending on current needs and the monitored situation (multi-purpose).

The proposed framework is demonstrated by discussing a greenhouse scenario, where the temperature distribution within the greenhouse is to be estimated. The main reason for selecting the (distributed) state estimation is because state estimation is used in a wide variety of applications, such as object tracking, traffic management and indoor climate control to name a few. Yet, the proposed reconfiguration approach is applicable for any system in which state estimation is the key component for processing sensor measurements. Therefore, in what follows a generalized framework is presented for reconfigurable state estimating systems.

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. Distributed State Estimation

Since the article focuses on a reconfigurable state estimating system, distributed state estimation is an important aspect and will be addressed in this section by considering a linear process model describing the state dynamics. In this context, several distributed solutions of the Kalman filter have been explored. See, for example, solutions proposed in [1]–[4] and the references therein. This section aims to derive a general framework, so that most of the currently available distributed Kalman filtering solutions can be employed in the proposed reconfiguration scheme. To that extent, let us start with the state estimation problem, after which the generalized framework is introduced along with some illustrative estimation algorithms.

A. 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.,

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

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

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

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]. The goal is to reason about the design decisions made by these existing solutions and to select the most appropriate one for the current situation (depending on available communication and computational resources and on the estimation performance). This reasoning process will be addressed in a "management layer" encapsulating the variants for local Kalman filtering, which will be discussed in section IV-B.

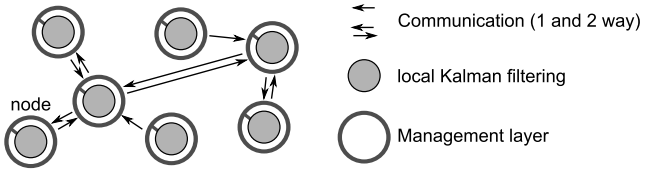

Figure 1. A network of Kalmen filters supported by a management layer.

The reasoner, which is employed by the management layer, should make an objective decision on which distributed Kalman filter (DKF) solution is currently most appropriate. This means that the alternatives for state estimation solutions should be described within a generalized framework, otherwise it is infeasible to compare the different alternatives in an objective manner. Next, let us introduce such a general framework for distributed state estimation.

B. A general framework for distributed Kalman filtering

The functional framework for computing the local estimate x_i is derived from existing DKF solutions. Typically, these solutions propose that each node i performs a Kalman filter (locally) based on its local measurement y_i and thereby, establishes an initial estimate $x_i \sim G(\hat{x}_i, P_i)$, e.g., in [1]–[4] and some overview articles in [5], [6]. After that, different DKF solutions propose different types of variables exchanged between two neighboring nodes i and j.

- Local measurements: node i receives y_j for all $j \in \mathcal{N}_i$, which can be exploited for updating x_i via a Kalman filter;
- Local estimates: node i receives $x_j \sim G(\hat{x}_j, P_j)$ for all $j \in \mathcal{N}_i$, which can be exploited for updating x_i via various merging solutions, e.g.. consensus or state fusion.

Based on the currently available DKF methods a generalized local estimation function is designed relying on the node's local measurement and on data received from neighboring nodes. The corresponding framework consisting of so called "functional primitives" is depicted in Figure 2. Each functional primitive of this framework, i.e., the Kalman filtering function f_{KF} and the merging function f_{ME}, 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 implementations for each functional primitive, from which a selection can be made during operation. Alternative implementations related to computing local estimation results $x_i \sim G(\hat{x}_i, P_i)$ and $x_{i^+} \sim G(\hat{x}_{i^+}, P_{i^+})$ are presented, next.

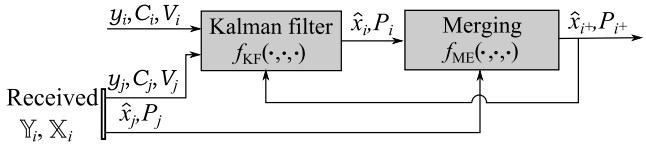

Figure 2. Framework of functional primitives compose the generalized local estimation function performed by each node i in the network.

Remark III.1 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, node j shares the local measurement (y_j, C_j, V_j), while if node j shares local estimates, it exchanges $(\hat{x}_j, P_j)$. This implies that 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 functional primitive combines the local $y_i[\mathsf{k}_i]$ 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 are combined 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, the information filter is more convenient when the amount of received measurements $(y_j, C_j, V_j) \in \mathbb{Y}_i$ varies at sample instants. Therefore, the Kalman filtering function in the framework of Figure 2 employs the Information filter. More precisely, $f_{\text{KF}}(\cdot,\cdot,\cdot)$ is a function with three inputs

and two outputs according to the following characterization:

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

The merging function of Figure 2, introduced as $f_{\mathrm{ME}}(\cdot,\cdot,\cdot)$, merges the local estimate $x_i[\mathsf{k}_i]$ with the received estimation variables $x_j \sim (\hat{x}_j, P_j) \in \mathbb{X}_i[\mathsf{k}_i]$ into a new estimate $x_{i^+}[\mathsf{k}_i] \sim G(\hat{x}_{i^+}[\mathsf{k}_i], P_{i^+}[\mathsf{k}_i])$. As an example, let us present three approaches for merging the local estimation results x_i with received estimation results x_j. Typically, the functional primitive $f_{\mathrm{ME}}(\cdot,\cdot,\cdot)$ is based on solutions for merging two state estimation results x_i and x_j, yielding a recursive behavior to merge all received estimation results. This means that the merging function $f_{\mathrm{ME}}(\cdot,\cdot,\cdot)$ performed by a node i has the following description:

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

Three alternatives for the inner-merging function $\Omega(\cdot,\cdot,\cdot,\cdot)$ are presented: one synchronization and two fusion approaches.

Recent DKF solutions, e.g., [2], [3], adopt a synchronization approach to characterize $f_{\mathrm{ME}}(\cdot,\cdot,\cdot)$. Such an approach stems from the idea of synchronizing different internal clocks in the network. Typically, synchronization is employed on the estimated means, for some scalar weight $\omega_{ij} \in \mathbb{R}_+$, yielding the following inner-merging function:

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

Solutions for establishing the weights ω_{ij} have been presented in literature extensively, for example in [7], [8].

Merging solutions that take the combination of error-covariances into account in addition to a combined estimated mean are known as fusion solutions. An optimal fusion method was presented in [9], though it requires that correlation of the two prior estimates is available. In the considered 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 [10] and [11], 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 $\omega_{ij} = \mathrm{tr}(P_i)\big(\mathrm{tr}(P_i) + \mathrm{tr}(P_j)\big)^{-1}$, i.e.,

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

The fusion method EI results in a "smaller" error-covariance 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 merging 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* $\gamma_{ij} \in \mathbb{R}^n$, yielding the following inner function $\Omega(\cdot,\cdot,\cdot,\cdot)$:

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

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 expression of γ_{ij} and Γ_{ij}, for some $\varsigma \in \mathbb{R}_+$ and $\{A\}_{qr} \in \mathbb{R}$ denoting the element of a matrix A on the q-th row and r-th column, yields

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

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

This completes the description of the estimation function depicted in Figure 2. Let us continue by explaining the reconfiguration parameters that can be tuned for the considered estimation function.

C. Tuning the functional primitives

The estimation function illustrated in Figure 2 consists of two functional primitives. A node i combines received measurements $\mathbb{Y}_i$ via the Kalman filtering functional primitive

f_{KF}, while received estimates $\mathbb{X}_i$ are merged in the functional primitive f_{ME} via either SY (synchronization), CI (covariance intersection) or EI (ellipsoidal intersection). There are several parameters and structural changes that the management layer can select, so that a suitable estimation function is constructed in line with the current situation and available resources. The different options for tuning a functional primitive are addressed in this section.

- Sampling time τ_i: The sampling time of both primitives f_{KF} and f_{ME} can be tuned accordingly. Lowering the sampling time τ_s implies that estimation accuracy, computational demand and data exchange will be decreased, i.e., saving communication and computational resources and thereby saving energy.
- Shared data $\mathbb{Y}_i$ and $\mathbb{X}_i$: The selection of which variables are shared, i.e., (y_i, C_i, V_i) and/or $(\hat{x}_i, P_i)$, can change in time. Exchanging both improves estimation results throughout the network but requires the most communication resources. Decreasing the usage of this resource can be done by exchanging either the local measurement or the local estimation result, which would yield a decrease in the estimation accuracy. Additionally the frequency of exchanging data can be influenced by changing the communication frequency parameter υ_i.
- Implemented algorithm: There is only one implementation of the functional primitive f_{KF}, which is the Information filter. Yet, for the merging primitive f_{ME} one has the option between four alternative implementations:
  1) Wire: The inner-function $\Omega(\cdot,\cdot,\cdot,\cdot)$ does not perform any merging and is thus characterized by $\hat{x}_i = \hat{x}_i$ and $P_i = P_i$. The required computational power for this alternative is minimal, though it would result in the lowest estimation accuracy as well;
  2) SY: The inner-function $\Omega(\cdot,\cdot,\cdot,\cdot)$ is characterized by the synchronization approach. The required computational power for this alternative is low, though it would result in a poor estimation accuracy as well;
  3) CI: The inner-function $\Omega(\cdot,\cdot,\cdot,\cdot)$ is characterized by the covariance intersection approach [10]. The required computational power for this alternative is moderate and would result in a moderate estimation accuracy;
  4) EI: The inner-function $\Omega(\cdot,\cdot,\cdot,\cdot)$ is characterized by the ellipsoidal intersection approach [11]. The required computational power for this alternative is high and would result in a high estimation accuracy;

The above options on changeable parameters and alternative functional primitive are exploited by the management layer for finding a match between the desired estimation quality (accuracy) and the required resources.

IV. Reconfiguration Framework

A. Architecture

The design challenge for any embedded system is to realize the required functionalities (in this case state estimation) on a given hardware platform while satisfying a set of nonfunctional 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 [12]. 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, etc. [13] Crucial for the design process are the interactions between the different models.

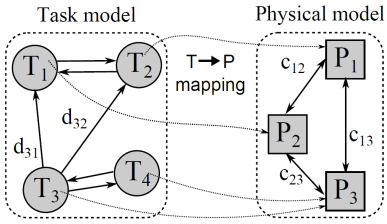

Figure 3. Task and physical model

Two fundamental models of the design are the task model (capturing the required functionalities of the employed signal processing method) and the physical model (capturing the hardware configuration of the implementation).

In Figure 3 the task model is represented as directed a graph: the signal processing components (tasks) are represented by the vertices of the graph, while their data exchange or precedence relations (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 the physical model are the computing nodes and the communication links.

It should be mentioned that in the signal processing (state estimation) context the task graph is designed in two phases (Figure 4): first the functional primitives are connected to form the signal flow graph satisfying the functional requirements and design constraints. Then the task network should be created via clustering the elements of the network of functional primitives to tasks considering computation, communication and temporal requirements. This is not a linear process and there is a strong dependency on the physical configuration. Moreover, the search and optimization in the design space make this an iterative process, which requires interactions between hardware, software and signal processing architecture designs.

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

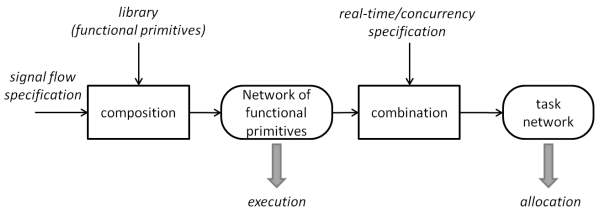

Figure 4. Function and task network dependency

links have limited capacities. The design process results in a sequence of decisions, which lead to a feasible system design. Traditionally this design process is "off-line", 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.

These are overly optimistic assumptions for large-scale sensor and actuator (control) networks: in many cases they are deployed in "hostile" environments, where component failures and dynamically changing configurations manifest themselves as common operational events.

Expressed in the concepts of Figure 3, conceptually the runtime reconfiguration is carried out via changing the task graph (i.e. selecting a different signal processing scheme, changing certain parameters of the functional primitives, 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. The functional scheme of the reconfiguration is shown in Figure 5. The primary functionality (in our case the state estimation) is realized by the task network (via the invocation of associated function primitives). The task network is built during initialization time according to (off-line) design specification. The reconfiguration runs parallel to the primary data stream: based on the execution status (e.g. quality of the results generated, the conditions of the hardware resources, the availability of the communication links, etc.) the reconfiguration functionality makes decisions about the configuration, its parameterization and resource usage in order to satisfy the given requirements and constraints. The reconfiguration is event driven, triggered by changes in the execution context or the changing (user) requirements and constraints. The reconfiguration may act on the software side (e.g. selecting a different algorithm to implement a particular functional primitive, changing task allocation, etc.) or on the hardware side (e.g. adjusting transmission power, suspending/awaking components, etc.).

The following characteristics of the proposed scheme should be emphasized:

- Every time instant the function/task network is a snapshot of the possible variants and mappings. The alternatives may not be explicitly enumerated but can be the result of a reasoning (problem solving) process.
- The scheme explicitly supports the separation of concerns principle. The reconfiguration mechanism can be designed and implemented relatively independently.
- The reconfiguration can be a resource demanding activity. The scheme allows for tuning the "intelligence level" of the reconfiguration depending on the performance of the hardware configuration - virtually leaving the signal processing aspect uninfluenced.
- There are low-overhead implementations available for dynamic data-flow graph based signal processing (e.g. [14]). Consequently the influence of the reconfiguration on the signal processing performance can be kept low. The interfacing between the data-flow graph and the "management" side is usually implemented by a simple API or message passing mechanism.
- The scheme is not specific to the distributed state estimation problem, but other signal processing tasks (incl. control) can be mapped into this architecture as well.
- The scheme is applicable to reconfigurations on "various levels of granularity": task, node and system levels, i.e. the reconfiguration scales from fine grade distributed to centralized. Needless to say distributed reconfiguration may need cooperation among the reconfiguration functionalities.
- From execution point of view the reconfiguration functionalities should be included in the task graph (as one or more cooperating tasks) and their resource demand should be accounted for.

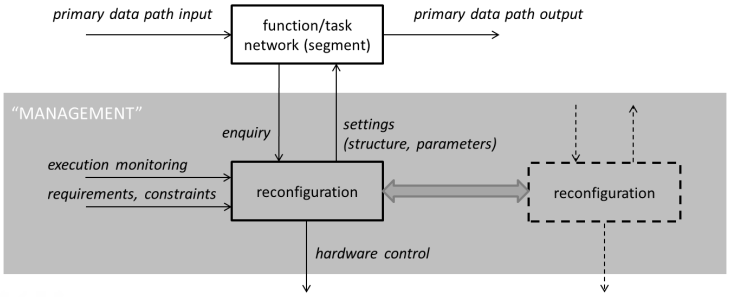

Figure 5. The reconfiguration of the primary data path

In the following the "management" side will be further detailed. The representation of the configuration and the design knowledge as well as the associated reasoning mechanisms are the key elements on the management layer, thus in the following these aspects will be emphasized.

B. Knowledge representation and reasoning

For the management layer as seen in Figure 1, which reconfigures the core tasks' functionality, a three-step strategy is implemented. The first step in reconfiguration is the *monitoring* step, in which the current status quo is observed, in order to reflect the system's own health/performance and the state of the embedding environment. The second step is

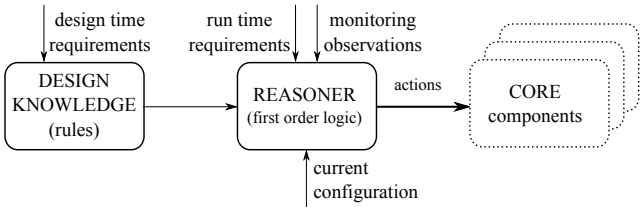

Figure 6. A schematic representation of the reasoning process

the *reasoning* step in which the observed characteristics are analyzed, decided whether reconfiguration is even required, and if it is required how to do so. The final step is the *actuation* step which performs the decisions made in step two, thereby completing the reconfiguration process.

The reasoning step is one of special interest, because it is mainly here where the intelligence about the configuration is represented in the system. Because the reasoning is only a single step within the reconfiguration process and it always operates with a predefined interface, it can be separated from the rest of the system which makes it relatively easy to implement any kind of reasoning module. A schematic representation of the reasoner is shown in Figure 6.

In order to reconfigure the tasks' core functionality, the reconfigurator requires knowledge and a method for reasoning about the various configurations. These two are intrinsically connected since the form of the knowledge representation depends on the method of reasoning. A case-based reasoner would require knowledge in terms of different complete configurations, whereas a utility reasoner would require a utility function.

From the many different forms of reasoning and knowledge representations, for this article a first order logic (FOL) reasoner will be used. The corresponding knowledge representation exists of a rule base existing of atoms representing the configurable parameters, and constraints and conditions in the form of logical compound statements to determine what parameter choices are valid and which are not. The problem statement of reconfiguration is now reduced to finding the values for the atoms that satisfy the statements (effectively a search). Now problem solving methods can be used for solving FOL problems such backtracking and the Selective Linear Definite (SLD) clause resolution which is proven to be sound and complete [15]–[17].

FOL has been used to describe reconfiguration knowledge in the past as well [18], [19]. An important reason for this is that FOL reasoners have proven to be very expressive in that they can describe both conditions and constraints, perform boolean, numerical and symbolical operations and therefore can reconfigure either by optimizing parameters or starting/stopping components [20].

Using a FOL reasoner however, also means that *a priori* to deploying the system, all constraints and conditions must be determined because a rule base is required by the reasoner. This requires some design time knowledge which can be expert or empirical knowledge from experiments. For scenarios in which multiple methods are applicable, a preferential order must be determined. A completely bias-free self-exploring reasoner is not implemented yet, but will be looked at in future.

V. Case Study

In the case study, the proposed framework is applied to a scenario in which the temperature distribution of a greenhouse has to be monitored. In the scenario there are N nodes all equipped with a temperature sensor, and a communication interface for communicating wirelessly. Each node has to estimate the temperature distribution. The implementation of the experiment was in the form of an a discrete event simulation.

Using off-line measurements from a small scale greenhouse setup, the system had access to real data, but still multiple experiments were feasible, whilst still keeping the same input. In the experiment, the six nodes that were simulated all received measurements from their sensor, which in turn would read the off-line measurement from the database.

The nodes were constrained in the bandwidth of the communication with other nodes, the amount of integer and floating point operations per second (IOPS/FLOPS) and a limited energy supply. In the ideal situation these constraints should imply that the node should initially use the most "expensive" method that would be computationally feasible, and under decreasing battery capacity would decrease its effort, and in the end use the "cheapest" method. At the same time the system would always employ methods that satisfy the current bandwidth conditions.

In the scenario implemented for this experiment, there are six nodes and in one of the nodes the monitor finds the battery level to drop below a certain threshold, such that reconfiguration is desired. In the experimental setup the system can change the following variables:

1) The algorithm used for the functional primitive f_{ME} (EI, CI or SY)
2) The sampling rate at which to sample the sensor τ_i
3) The communication rate to send out information to neighboring nodes υ_i
4) The *sleep* time of the radio, which controls the rate of incoming messages from neighboring nodes υ_i'

In the initial configuration the system uses EI in the merging functional primitive, samples its sensor once every 60 seconds and broadcasts its state estimation results every 90 seconds. Furthermore it fires the monitoring cycle every 150 seconds and shuts down the radio for 5 seconds every other 30 seconds. (Thereby using a pattern of 5 seconds sleep - 25 seconds receiving.) Exchanged variables are automatically chosen, so that if a node uses EI or CI it broadcasts $(\hat{x}_i, P_i, \mathsf{k}_i, i)$ whereas if it uses SY it broadcasts $(\hat{x}_i, \mathsf{k}_i, i)$.

The reasoner implemented relies on a Prolog based FOL interpreter [1]. Choosing this implementation has a couple of additional advantages. First and foremost, it has been used for a long time by experts for encoding wide variety of knowledge [21], and is very expressive, but also flexible in

[1] http://www.gnu.org/software/gnuprologjava/

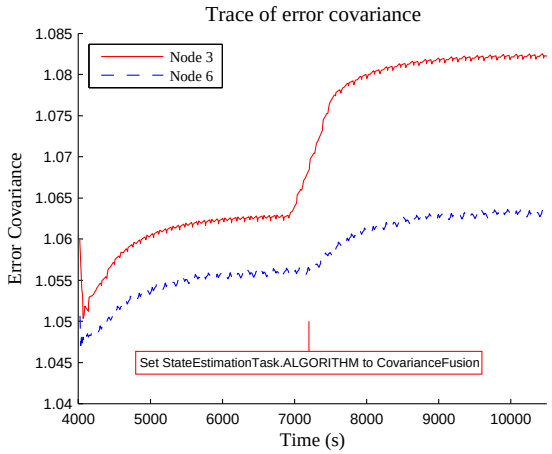

Figure 7. The error trace of two nodes, when changing the fusion algorithm in one of them

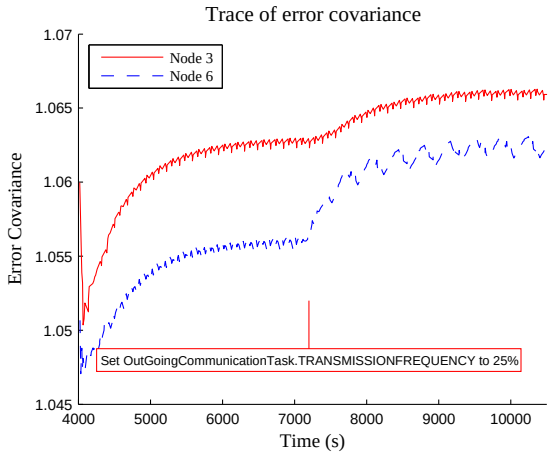

Figure 8. The error trace of two nodes, when changing the communicating frequency in one of them

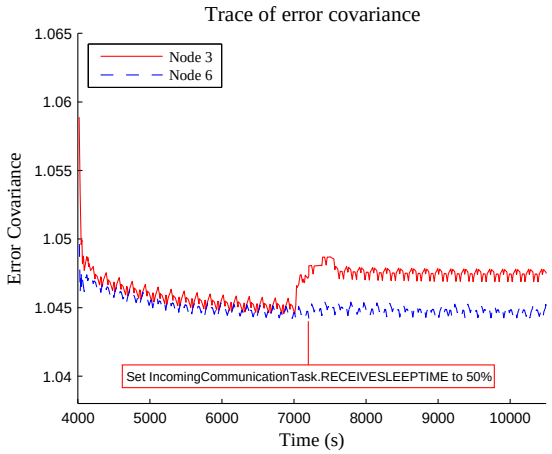

Figure 9. The error trace of two nodes, when changing the radio sleep time in one of them

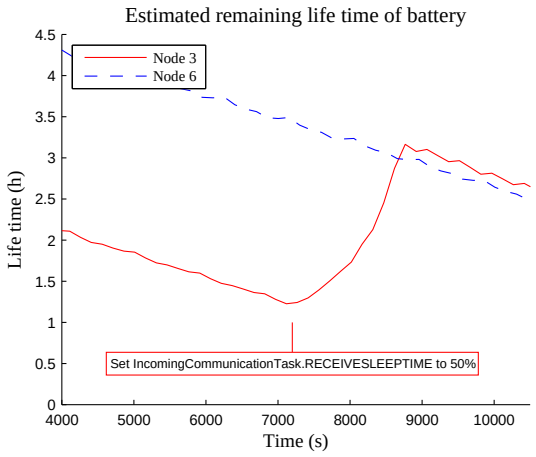

Figure 10. The estimated remaining battery life when changing the radio sleep time

the type of rules that could be used to describe the knowledge for the application. Secondly, Prolog is a well-known logical programming language and therefore the framework can easily be programmed using existing syntax and methods. Finally, the Prolog implementation can use an external rule base, separating the reasoning rule base from the implementation itself. This results in having the reconfiguration behavior defined separately and independently of the rest of the system.

The FOL rule base allows for a wide range of rule types. The rule base created for the case study contained rules of varying complexity, from very simple rules such as *"The ideal sampling frequency* 0.2 *hertz"*, *"The battery level is critical if it is below a certain threshold"* to more complex rules such as *"If the communication frequency is not low, and the battery is running low suggest lowering the communication frequency"*. These exemplary statements would formally be implemented in prolog as following:

Example prolog reconfiguration statements:

```
idealSamplingFrequency(0.2).
batteryCritical(State) :-
getBatteryLife(State, BatteryLife),
minBattery(Threshold), BatteryLife <
Threshold.
action(lowerCommunicationFrequency,
State, Reason) :- batteryLow(State),
\+ communicationFrequencyLow(State),
Reason = "The battery is low and the
communication frequency is high enough.".
```

The knowledge of the reconfiguration reasoner must be coded *a priori* to the deployment of the system. The constraints and conditions of the rules, and the order in which the different reconfiguration actions will be performed must be determined. This does however not mean that the system will reconfigure in any specific order, because the operating conditions are unknown beforehand, and therefore the constraints and conditions determine the configuration at runtime.

The order of the rules therefore simply implies a preference of certain configuration aspects, since in Prolog the first rule that satisfies all conditions will fire.

In order to determine the ordering of the different type of configurations and to create the related rule base, some expert knowledge was used and some experiments were ran. In these experiments the six nodes would do their work, and after two hours of simulation time, the configuration of Node 3 would change. For the results of these experiments, see Figures 7 through 9. In Figure 7 it can be seen that changing the fusion method has a significant impact on the error covariance of the state estimation in reconfigured node, but also in the node that stays the same. The same holds for a change in the outgoing communication frequency υ_i, of which the results can be seen in Figure 8. The sleep time of the radio has the relatively smallest impact, and seems the best option for the first reconfiguration action. This means the node will receive less messages from the other nodes and therefore the error covariance will go up, but much less than with any of the other options. Using this information, a rule base can be created.

When running the resulting system with the created rule base in a simulated experiment, the reconfiguration reasoner will deduce that a different sleep time is most desirable. In Figure 10 it can be seen that the corresponding remaining battery lifetime goes up significantly.

VI. Conclusions and Future Work

Using the framework introduced enables reasoning about the configuration of how to do state estimation. Using a library of different state estimation methods and a way of reasoning about when a method should be used, and with what kind of parameters, we can improve the quality under changing operating conditions.

The highly modular approach allows for easy implementation of a different reasoning engine, but also for portability for different state estimation tasks.

In the example experiment it is shown that the framework is capable of improving the battery lifetime of a sensor by run time reconfiguring the state estimation method. In the example this is done by modifying a parameter for the communication strategy, and thereby having minimal impact on the state estimation error.

In the current reasoner, an *a priori* determined set of ordered rules exists in the rule base. In future research it would be desirable to let go of these constraints and create a unbiased context-free configuration space exploring reasoner. An example of this could be a genetic algorithm adapted to the reconfiguration scenario. An alternative way to go could be implementing a learning algorithm.

Also this paper only looked at the state estimation problem, whereas the framework allows for reconfiguring all kinds of distributed reasoning tasks. In the direct future a reconfiguration method for a distributed control task will be studied.

Finally in this article only (high fidelity) simulated experiments are discussed. In the near future an implementation of the framework on various embedded system and mobile computing platforms will be carried out.

Acknowledgments

The project was a group effort as a part of the Adaptive Multi Sensor Networks (AMSN) TNO research program. The authors would like to thank Julio Oliveira and Mark Zijlstra for their help with implementing the experiments. Furthermore they would like to thank Leon Kester, Ad van Heijningen and Paul Booij for their contributions.

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, "State estimation in networked systems," Ph.D. dissertation, Eindhoven University of Technology, 2012.

[7] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems and Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[8] A. Tahbaz Salehi and A. Jadbabaie, "Consensus over ergodic stationary graph processes," *IEEE Trans. on Automatic Control*, vol. 55, pp. 225–230, 2010.

[9] 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.

[10] 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.

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

[12] 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.

[13] G. Karsai, F. Massacci, L. Osterweil, and I. Schieferdecker, "Evolving embedded systems," *Computer*, vol. 43, no. 5, pp. 34 –40, may 2010.

[14] G. Nordstrom, J. Sztipanovits, and G. Karsai, "Metalevel extension of the multigraph architecture," in *Proceedings of the IEEE ECBS'98 Conference*, Jerusalem, Israel, April 1998, pp. 61–68.

[15] R. Stärk, "A direct proof for the completeness of sld-resolution," *CSL'89*, pp. 382–383, 1990.

[16] K. R. Apt and R. N. Bol, "Logic programming and negation: A survey," *The Journal of Logic Programming*, vol. 19, pp. 9–71, 1994.

[17] M. Ben-Ari, "First-order logic: Logic programming," *Mathematical Logic for Computer Science*, pp. 205–222, 2012.

[18] M. Endler and J. Wei, "Programming generic dynamic reconfigurations for distributed applications," in *Configurable Distributed Systems, 1992., International Workshop on*. IET, 1992, pp. 68–79.

[19] M. Simonot and V. Aponte, "A declarative formal approach to dynamic reconfiguration," in *Proceedings of the 1st international workshop on Open component ecosystems*. ACM, 2009, pp. 1–10.

[20] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger, "A survey of self-management in dynamic software architecture specifications," in *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. ACM, 2004, pp. 28–33.

[21] G. Rossi, "Uses of prolog in implementation of expert systems," *New generation computing*, vol. 4, no. 3, pp. 321–329, 1986.