I-SENSE: A Light-Weight Middleware for Embedded Multi-Sensor Data-Fusion

Allan Tengg[1], Andreas Klausner[1], and Bernhard Rinner[2]

[1]Institute for Technical Informatics,
Graz Univerity of Technology, A-8010 Graz, Austria
{tengg,klausner}@iti.tugraz.at

[2]Institute of Networked and Embedded Systems,
Klagenfurt University, A-9020 Klagenfurt, Austria
bernhard.rinner@uni-klu.ac.at

Abstract — *In our I-SENSE project we demonstrate the combination the scientific research areas multi-sensor data fusion and pervasive embedded computing. The main idea is to provide a generic architecture which supports a distributed data fusion on an embedded system. Due to the high onboard processing and communication power of the used hardware, our proposed architecture is designed to perform sophisticated data fusion tasks. Another goal of I-SENSE research project addresses the reconfiguration of a distributed system at runtime, thus, to be able to react to changes in the system's environment dynamically.*
This paper though gives an overlook of our developed middleware which eases the development of distributed fusion applications on embedded systems and which includes reconfiguration facilities. We further present some experimental results obtained using our middleware and give an outlook of our ongoing research.

1 Introduction

Multi-senor data fusion is a technique by which data from several sensors are combined through a data processor to provide comprehensive and accurate information. The powerful potential of this technology stems from its ability to track changing conditions and anticipate impacts more consistently than could traditionally be done with a single data source.

The major goal of our I-SENSE research project [1] is to investigate and develop a scalable and embedded architecture for various multi-sensor applications. The I-SENSE framework is based on embedded intelligent sensor nodes with sufficient computing and communication performance, which allows us to distribute software tasks among geographically distributed sensor nodes. By delegating the CPU-expensive data fusion tasks into the sensor nodes, the requirements concerning the communication bandwidth can be reduced compared to centralized data fusion architectures. This makes widespread data-fusion applications more feasible. To accomplish a high flexibility, we decided that the

functional description of a data fusion system - the so called *Fusion Model* - should be defined (almost) independently of the present hardware configuration. Together with the hardware model, which is derived from the actual hardware, the I-SENSE framework tries to find a valid mapping from the fusion model on this specific hardware automatically.

The focus in this article is set on the fusion middleware which eases the development of a distributed fusion application. The user of this system has to define basically some software components the so-called *fusion tasks* and specify their interconnection among each other. All other steps necessary, like finding an optimal mapping of the components on the hardware platform, exchanging data between the components, detecting and handling errors in the running system as well as reconfiguring the system during runtime is handled by the I-SENSE runtime environment.

The remainder of the paper is organized as follows: Section 2 gives a review about related activities. Section 3 describes our current I-SENSE hardware in more detail and explains the so called *hardware model* well as the *fusion model* – the software description of a fusion system. Section 4 presents the I-SENSE middleware and the services it provides for *fusion tasks*. The configuration method for the I-SENSE network is the focus of section 5. A short overview of our case study *traffic surveillance* is given in section 6. In section 7 we present some results obtained from our system so far before section 8 concludes the paper with a short summary and gives an outlook of our further work.

2 Related work

Our idea of developing a high-performance data fusion architecture originates from the SmartCam research project which is conducted at the Institute for Technical Informatics at the Graz University of Technology [2, 3]. Our smart cameras combine video sensing, video processing and communication on a single embedded device, consisting of a network processor and various digital signal processors (DSPs). In the I-SENSE research project this SmartCam is extended to distributed embedded sensor nodes capable of fusing data from various heterogeneous sensors, ranging from simple sensors such as light barriers and induction loops over audio sensors to several different image sensors.

A project that seems to have quite many similarities with our project is called *DFuse* [4]. This research focuses on challenges of data fusion applications in wireless ad hoc sensor networks. However, their system is designed to be used on 'motes'. Since motes are usually battery powered, the main concern in *DFuse* is power consumption. Besides that, both the communication range and the communication bandwidth, is very limited between fusion-nodes in *DFuse*.

Like many other data fusion systems, we also describe the functionality in form of dataflow graphs [5]. Consequentially, the implementation of a generic data fusion architecture has to involve a dataflow graph synthesis tool. There have been presented plenty techniques for synthesizing dataflow graphs onto multiprocessor systems, a problem known to be NP complete [6, 7]. However, it is difficult to find reports of using genetic programming for solving task allocation problems, like we do in the I-SENSE project.

3 The I-SENSE architecture

With regard to the objectives of the I-SENSE project, a configuration strategy has been elaborated, which is described briefly in the upcoming section. This includes a detailed description of the hardware model as well as the fusion model.

3.1 Hardware model

The hardware model describes the distributed embedded system where the fusion application should run on. In our case it consists of a set of connected hardware nodes ($N1 \ldots N4$, cp. figure 1). Each hardware node has at least one general purpose CPU (parent) and optionally some digital signal processors (children) coupled via PCI, and various ports to interface sensors.

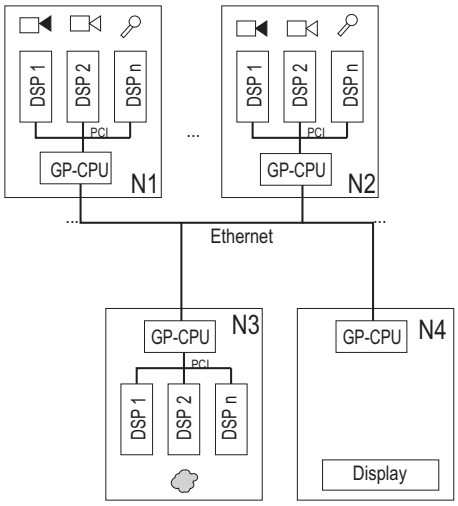


Figure 1: The hardware topology of an I-SENSE network

Our I-SENSE prototypes of sensor nodes are implemented with standard high-performance components:

Intel Pentium M board 'ePCI101' Embedded computer system from Kontron, 1.6 GHz (passive cooling), 512 MB memory, two 100 MBit/sec Ethernet ports, two serial ports, several USB ports VGA, 256 MB flash card on-board and 4 free PCI slots

Network Video Development Kits from ATEME, equipped with a Texas Instruments TMS320C6416 fixed point DSP running at 600 MHz and with a total of 264 MB of memory.

PCI Multifunction Encoder cards equipped with two Texas Instruments TMS320DM642 fixed point DSPs running at 600 MHz and 128 MB memory for each core. Each DSP has 12 multiplexed video ports that may be used to capture PAL or NTSC signals.

SI-C67DSP cards from Sheldon Instruments, based on a Texas Instruments TMS320C6713 floating point DSP running at 250 MHz and a total memory of 256 MB.

Currently the sensor boards are equipped with the following sensors.

- PAL color camera from *Ganz*

- PAL Infrared camera with night vision
- Professional Audiocard *Audiophile 2496*
- Light barrier

Every processing node allows to query and use its free resources (i. e. computing power, on/off chip memory, different sensors, ...) for *fusion tasks*. We provide a module which explores the embedded system automatically. This has two advantages: (i) faulty or missing hardware nodes can be found during start up and (ii) the hardware model can be built and parameterized during the initialization process.

A single sensor node in our I-SENSE framework has substantial processing power. However sophisticated video- and audio-based data fusion algorithms have high memory- and processing requirements.

3.2 Fusion model

The *Fusion Model* describes the functionality of the distributed fusion application and consists basically of a set of communicating tasks which may be represented as a task graph $G = (N, E)$. It is assumed to be a weighted directed acyclic graph, consisting of nodes $N = (n_1, n_2, ..., n_m)$ which represent the *fusion tasks* and the edges $E = (e_{12}, e_{13}, ..., e_{nm})$ which represent the data flow between those tasks.

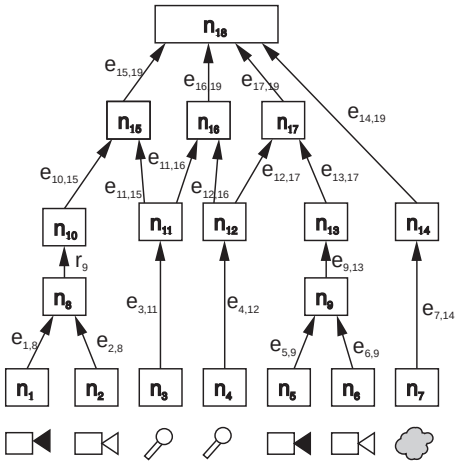


Figure 2: A simple *Fusion Model*

Each node has some properties, describing the (hardware/resource-) requirements of a task. Every edge from node u to node v (e_{uv}) indicates the required communication bandwidth between those two tasks. A quite simple example of a *fusion model* is shown in figure 2.

3.3 Scheduling of fusion tasks

It is important to note, that the *fusion tasks* are required to fulfill hard real-time requirements (i. e., a video frame is processed in n CPU cycles) while the entire system does not and can not (due to PCI and usage of standard Ethernet components) guarantee hard real-time behavior. Instead soft real-time conditions [8] are met. This means, that deadlines

may be missed occasionally. In our case such a miss is likely to occur during a reconfiguration of the system. During normal operation, our middleware is designed to keep the delay of data on the communication links as low as possible and constant. Assumed that there are no hardware faults and the fusion tasks are implemented correctly, the system guarantees that no data get lost inside the system. This is achieved by using well dimensioned buffers on all communication links. Together with the provided timestamping mechanism, it is ensured that the information from different sensors can be temporally aligned and the results of the fusion process can always be causally related to their real world origin. Furthermore, committing to soft-real time behavior and using a preemptive timeslice based scheduler simplifies the scheduling of the *fusion tasks*: Every task may begin as soon as the required data are available in its input buffers [9].

3.4 Description of fusion tasks

There are two parts needed to describe a task: First, a dynamic loadable library, written in C/C++, that does the data processing and which has access to the I-SENSE API. Second, meta-information about the task has to be provided in a separate XML file. This meta-information is required for two reasons: The automatic task placement module needs to know the resources and precisely predict the run time of each task to find a valid optimal mapping of tasks onto CPUs. When this configuration is loaded onto the distributed system, this meta-information is used to initialize the communication buffers and memory segments for every task. A very simple component description is demonstrated in the following XML listing:

```
<component>
    <platforms>
        <platform name="DSPC64">
            <property name="DLLFile" value="ColorCamera.bin"/>
            <property name="Stacksize" value="2048"/>
            <property name="IntMem" value="2848"/>
            <property name="ExtMem" value="0"/>
            <property name="NrDMAChannels" value="1"/>
            <property name="EnviromentSize" value="256"/>
            <property name="Cycles" value="6600"/>
        </platform>
    </platforms>

    <ports>
         <port name="0">
            <property name="InputMessageSize" value="256"/>
            <property name="OutputMessageSize" value="307200"/>
            <property name="InputBufferCount" value="3"/>
            <property name="OutputBufferCount" value="3"/>
            <property name="MessageRate" value="10"/>
        </port>
   </ports>
</component>
```

Most of the fields in this XML meta-information file are self explaining. The *EnvironmentSize* specifies the size of the task state storage which is initialized when a task is created. If a task is migrated from one processor to another, this memory block is always transferred along with the code. The field *Cycles* specifies how long it takes the task to process a message in the worst case. Finding the correct value is very simple in case of our used DSP cards. Here it is possible to use a profiler to get an accurate estimation

of the CPU cycles required to process a message. Due to the variety of Pentium based computer systems, different cache sizes, different memory modules used, and many other differences, it is quite difficult to say in advance how long it will take to process a message on a Pentium class general purpose CPU. We currently evaluate all algorithms on one specific platform and interpolate the timings linearly for other systems according to their benchmark results (processor performance, memory).

4 The I-SENSE Middleware

Basically, almost every embedded system with sufficient computation and communication power can be turned into an I-SENSE hardware node by simply running the I-SENSE middleware on it. Though for simplicity reasons, the current implementation is designed for Intel Pentium compatible processor running Windows XP Embedded operating system. When started, it first scans the system for supported DSP cards, installs the I-SENSE system on each detected DSP and establishes a connection via PCI to all DSP processors in the system. After the WIN32 based part has been initialized as well, the node is ready to accept commands and execute *fusion tasks*.

4.1 Software Architecture Overview

Figure 3 illustrates the internal structure of the I-SENSE middleware.

The *message router* is responsible for a correct and efficient data transfer from one *fusion task* to another, either on the same processor via *shared memory*, the same node via PCI or on a distant node via Ethernet. Furthermore, the message router supports message forwarding for tasks which have been migrated to another processor.

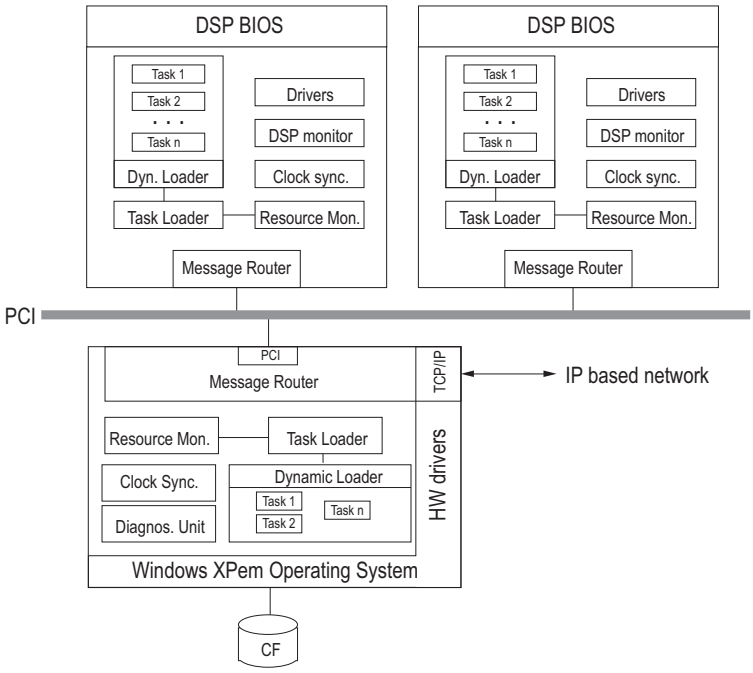


Figure 3: The main services of the I-SENSE middleware

Each processor in the I-SENSE network offers a service called *task loader*. It accepts

requests to load, start, stop, migrate and remove *fusion tasks*. Loading a task involves basically the following steps: First the fusion controller sends a request to load a specific task in form of a dynamic loadable library. If the code is not yet present at the system, the image is transferred. After that the task environment is transferred and installed. The next step involves the creation and registration of the communication links. If all previous steps have been completed successfully, the task main routine is called in an own thread.

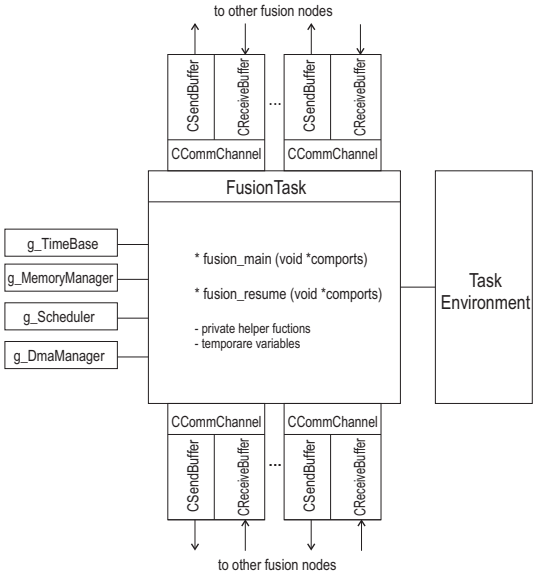
It is the *resource monitors* responsibility to keep a record of all consumed resources by a task (Memory blocks, DMA channels, ...). This assures a neat removal of *fusion tasks*.

Distributed sensor data fusion implies a uniform timebase for all nodes. Without a system wide synchronized clock, it would be impossible to combine results from different sensors. Therefore each processor has its own task which keeps the local *clock synchronized* with the system time.

To detect software- and hardware-failures, each node periodically checks its state, and the connection to its neighbor nodes. This functionality is summarized in the *DSP Monitor* and *Diagnosis Unit* block, respectively.

4.2 API for fusion tasks

The last chapter gave a rough overview of the I-SENSE software architecture. The user of the I-SENSE system hardly ever has to care about the components mentioned before. Each *fusion task* has a number of ports where it is connected to other *fusion tasks*, as defined in the *fusion model*. These communication links are bidirectional. The number of available ports and the number of available message slots as well as the size of the message slots for outgoing and incoming messages have to be declared in the task's metadata.




```
void fusion_main(void *commports[])
{
   CCommChannel port1(commports[0]);
   CCommChannel port2(commports[1]);
   MESSAGEHEADER *inmsg, *outmsg;

   // ToDo: Initialization, restore state from env.

   while (!g_Scheduler.CheckMigrate())
   {
      if (!port1.GetNextMessage(&inmsg,INFINITE))
      {
          // ToDo : process the input message

          if (port2.PrepareMessage(&outmsg,0)>=0)
          {
             // ToDo : write the result in the
             //        ouput message
             port2.SendMessage(outmsg);
          }
      }
      port1.DeleteMessage(inmsg);
   }

   // ToDo: Store task state in environment and
   //       do cleanup
}
```

Figure 4: The I-SENSE middleware from the viewpoint of a fusion task

In the listing of figure 4, the skeleton structure of all *fusion tasks* can be seen. After an initialization phase, a message is taken from *port 1*, its data is processed and the result is posted on *port 2*. This procedure is repeated potentially forever, as long as the system doesn't request the task to terminate/migrate. If the task is requested to prepare for a

migration, it must store its context in the task environment, so that it can continue its work on the new processor without information loss. The task does not have to care about unprocessed messages, the communication subsystem transfers them to the new location transparently.

In addition to a simple message passing system, the I-SENSE API provides other very useful functions to ease the development of distributed fusion applications. Via the *Timebase* module, tasks can query the system time – which is synchronized over the entire system – whenever they want. They can fork new threads by using the *Scheduler* module. The *Memory Management* module standardizes and encapsulates the hardware dependent memory management functions of the underlying operating system. A *DmaManager* provides a variety of functions to ease the programming of DMA transfers on TI DSPs – a powerful but complex issue for DSP programmers.

5 Configuration method

The desktop- and embedded-computers in the I-SENSE system do not require user interactions nor do they require a display or a permanent storage device. However, there has to be one extraordinary node, the so called *Master Node* where the user of the system can specify and change the functionality of the entire network. This particular node is furthermore in charge of finding and loading a configuration onto the network, doing reconfigurations during runtime and handling problems and exceptions in the system. On the *Master Node* a repository of all *fusion tasks* must be installed and this is where the *Hardware Model* is parameterized before a configuration is computed.

The user triggers, either by loading a predefined or by creating a new *Fusion Model* or by changing the *Hardware Model*, the configuration process. Both models, the *Fusion Model* and the *Hardware Model* are the inputs of the so called *Optimizer* which tries to find a suitable mapping of the *fusion tasks* onto the processors which distributes the load balanced by invoking a genetic optimization algorithm [10]. Constraints help to enforce the mapping of an individual *fusion task* onto a dedicated processor. As soon as a valid configuration is found, the *Configuration Synthesizer* distributes and runs the *fusion tasks* on the network of distributed embedded platforms.

There are four possible situations that require the *Master Node* to trigger a reconfiguration:

- The user selects a new *Fusion Model* or modifies the existing *Fusion Model* or *hardware model*.
- A fusion node, usually located at a higher level in the fusion tree, detects a relevant event and decides to adapt the *Fusion Model* to better monitor or record this event.
- One of the *DSP Monitors* or *Diagnosis Units* reports the failure of a node. In such a case, the faulty node is removed from the *Hardware Model* and a reconfiguration is triggered to find, if somehow possible, an alternative mapping without the faulty node.
- A reconfiguration may be required if a software task allocates more resources than it has declared before. The I-SENSE runtime environment however tolerates this as long as the resources are available but it updates the *Fusion Model* to the real requirements to prevent potential conflicts in future.

6 Case study traffic surveillance

To demonstrate the feasibility of the I-SENSE approach in a real world scenario, we are developing a traffic surveillance system. This application requires (i) high data rates from the sensors to the processing units, (ii) high computation performance, and (iii) sophisticated algorithms for data abstraction and fusion. So in our opinion a promising solution for these requirements is to integrate sensing and computation into high-performance sensor platforms. To achieve this goal by using the I-SENSE middleware, *fusion tasks* have to be developed, beginning from the data acquisition, data preprocessing, object detection, object tracking, feature extraction, feature fusion – just to name a few.

6.1 Video analysis for traffic surveillance

Image processing tasks are usually very computation expensive and therefore a very good example to demonstrate the capabilities of the I-SENSE architecture. The following list gives an incomplete overview of the *fusion tasks* required to build a multi-sensor traffic surveillance system.

- *Image acquisition:* Acquire images from a camera and convert it into a standardized internal format (resolution, color space, ...) and detect faulty/missing cameras.
- *Image preprocessing:* Remove distortions caused by the camera lens, deal with interlaced images
- *Image filtering:* Apply different filtering methods to enhance or transform images
- *Image registration:* Transform two images from different cameras into a uniform coordinate system
- *Image fusion:* Combine images from two sources into a new image, i. e. create multi-spectral images
- *Motion detection:* Find moving objects in a sequence of images
- *Feature extraction:* Find characteristic features of detected objects to recognize the object in other frames/cameras again or to perform a classification of the object
- *Object tracking:* Follow a object in the scene and over more sensor nodes, detect collisions, lost cargo, etc.
- *Video recording:* Provide the system with an ability to record interesting scenes.

6.2 Acoustic traffic surveillance

Video based traffic surveillance is quite wide spread. However, there are various situations where video based surveillance fails or produces wrong inferences. Therefore collecting acoustic information is a quite natural way to extend the visual sensors. To provide information from the acoustic domain for our data fusion, the following software components are needed:

- *Data acquisition:* Acquire acoustic samples from microphones and provide it for further *fusion tasks*
- *Audio filtering:* Remove disturbing background noise and unwanted frequency bands

- *Signal transformation:* Perform a fourier analysis which is the basis for further signal analysis
- *Object detection:* Find objects passing by the microphones, estimate their speed by exploiting a stereo setup
- *Acoustic feature extraction:* Find characteristic features of detected objects to track the object from sensor node to sensor node other to perform an acoustic classification of the object

6.3 Other sensors for traffic surveillance

There are various other sensors available for traffic surveillance [11]. However, we decided to use a light barrier as additional sensor. The inferences of a light barrier alone are very limited. But as supplement of our audio visual traffic surveillance prototype its contribution is quite worthy.

6.4 Fusing data from different sensors

In our set of *fusion tasks* Support Vector Machines (SVM), proposed by Vapnik [12, 13] are used as classification method for decision modelling. Necessary time and memory usage are the main bottlenecks for training kernel methods, such as SVM.

For training data sets larger than 3000 elements, common SVM learning strategies are not feasible, especially on embedded platforms. Therefore, a modified version of the original SVM, the so called Least Squares Support Vector Machine (LS-SVM) [14, 15] is used for decision modelling in our system. The main characteristic of LS-SVMs is the lower computational complexity compared with original SVMs. LS-SVM and the original SVM are based on the same principals. The main difference is, that LS-SVM formulation uses equality constraints instead of inequality constraints for the cost function, which have to be minimized which is much easier to compute. The extraction of support vectors from a given training dataset is comparable with the problem formulation of finding the most significant vectors in a given data set. The optimal solution for solving this task should combine the following features. It should (i) be fast, (ii) lead to a sparse solution (i.e. low number of support vectors) and (iii) produce good classification results.

In [16] we present a method for an intelligent pre-selection of learning data in order to reduce the training set and therefore reduce the number of support vectors which are then used by the LS-SVM classifier. Using our approach leads to a sparse LS-SVM classifier with good classification results (approx. 2% higher error rate compared to standard SVM, which is negligible for our case study) and lower computational (70% faster than Standard SVM) and lower memory costs (about 55% less data for storage compared to LS-SVM) – especially for embedded systems with limited resources a preferred approach.

Due to the information sources (in more detail the sensors output or SVM output), contributed information can not associate a 100 percent probability of certainty to their individual output decisions. Dempster's rule of combination [17] is used to combine the knowledge from multiple sensors about events – the so called propositions. Furthermore, Dempster's rule lets us find the intersection of the propositions and their associated probabilities. The output of the fusion process at decision level is given to a decision logic. This task selects the hypothesis favored by the largest amount of evidence from the Dempster-

Shafer data *fusion task*.

7 Results

To test the performance of the middleware and estimate the time that it takes to configure and re-configure the system, we constructed some simple *fusion models* consisting of 4 tasks maximum. Table 1 presents the time span to load specific tasks. To obtain the presented results, caching of *fusion tasks* was turned of. It is self-evident that loading a task takes the longer, the larger the code- and environment-size is.

Task Name	CPU Type	Code Size	Environment	Time
Image viewer	Pentium	154 kB	256 Byte	139.7 ms
Motion detector	Pentium	162 kB	512 kB	273.6 ms
Camera driver	DSP	5 kB	256 Byte	23.3 ms
Motion detector	DSP	7 kB	512 kB	143.6 ms

Table 1: Time required to load a task onto the system

In a second test run, we measured the time it takes the middleware to move a task from one processor to another. The results of this experiment can be seen in table 2. In principal two different scenarios have to be considered. Either the task is moved from one node to another via Ethernet (*remote destination*) or the task is just moved between processors on the same node via PCI (*local destination*). The time spans collected in table 2 have been measured from issuing the first migration request to the confirmation that the task is running at its new location.

Task Name	Source	Code Size	Environment	Destination CPU	Time
Image viewer	Pentium	154 kB	256 Byte	remote Pentium	508 ms
Camera driver	DSP	5 kB	256 Byte	local DSP	436 ms
Camera driver	DSP	5 kB	256 Byte	remote DSP	475 ms
Motion detector	Pentium	7 kB	512 kB	local DSP	520 ms
Motion detector	Pentium	7 kB	512 kB	remote DSP	552 ms
Motion detector	DSP	162 kB	512 kB	local Pentium	613 ms
Motion detector	DSP	7 kB	512 kB	local DSP	395 ms
Motion detector	DSP	162 kB	512 kB	remote Pentium	623 ms

Table 2: Time required to migrate a task between processors

Regarding a reconfiguration during runtime, it is furthermore interesting how long it takes the system to update a communication link. Independent from the platform used, it took approximately 15 ms to alter a communication link.

8 Conclusion and Outlook

Although presented in connection with a traffic surveillance prototype, our I-SENSE design is not specialized for a specific scope of application. It can be used for any distributed application that processes data from geographically distributed sensors and is not required to fulfill hard real-time criteria.

The presented approach is capable of adapting its functionality if desired by the system or the user. Unfortunately, a reconfiguration causes the entire system to stop for a short time and, at least with the current I-SENSE implementation, the internal status of all *fusion tasks* is lost. Especially the loss of status information may be intolerable for many applications. A possible work around would be, that all internal data are gathered before deleting the actual configuration from the system and loading the new one. We prefer though a solution where the system continues to run and the internal status of the nodes remains unchanged.

Therefore our further work will include (i) the exploration of path search algorithms like A* to find a sequence of valid configurations to apply minimal to medium changes in the *fusion model* while the application is still running, (ii) the implementation, optimization and evaluation of advanced audio-visual feature extraction- and fusion-algorithms and (iii) the implementation of sophisticated error handling mechanisms for all kinds of faults imaginable.

Acknowledgments

This project has been partially supported by the Austrian Research Promotion Agency.

References

[1] A. Klausner, B. Rinner, and A. Tengg. I-SENSE: Intelligent embedded multi-sensor fusion. In *Proceedings of the 4th IEEE International Workshop on Intelligent Solutions in Embedded Systems (WISES)*, page 105116, Vienna, Austria, June 2006.

[2] Michael Bramberger, Roman Pflugfelder, Bernhard Rinner, Helmut Schwabach, and Bernhard Strobl. Intelligent Traffic Video Sensor: Architecture and Applications. In *Proceedings of the Workshop on Mobile Computing (TCMC 2003)*, Graz, Austria, March 2003.

[3] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Distributed Embedded Smart Cameras for Surveillance Applications. *Computer*, 39(2):68–75, February 2006.

[4] Rajnish Kumar, Matthew Wolentz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. DFuse: A Framework for Distributed Data Fusion. In *Proceedings 2003 ACM SensSys*, Los Angeles, Ca., November 2003.

[5] David L. Hall and James Llinas. *Handbook of Multisensor Data Fusion*. CRC Press, 2001.

[6] Ajith Tom P. and C. Siva Ram Murthy. Optimal task allocation in distributed systems by graph matching and state space search. 46(1):59–75, April 1999.

[7] Hluchy L., Dobrucky M., and Dobrovodsky D. A Task Allocation Tool for Multicomputers. In *Proc. of Scientific Conference with International Participation - Electronic Computers and Informatics*, pages 129–134. Kosice - Herlany, 1996.

[8] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kulwer Academic Publishers, Norwell, Massachusetts, USA, 1997.

[9] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.

[10] A. Tengg, A. Klausner, and B. Rinner. An Improved Genetic Algorithm for Task Allocation in Distributed Embedded Systems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007)*, London, UK, July 2007 (to appear).

[11] M.J. Dalgleish. Vehicle detection for advanced transport telematics. In *Proceedings of the 7th International Conference on 'Road Traffic Monitoring and Control'*, pages 164–167, London, UK, April 1994.

[12] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, USA, 1995.

[13] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, USA, 1998.

[14] J.A.K Suykens and J. Vandewalle. Least squares support vector machine classifier. *Neural Processing Letters*, 9(3):293–300, June 1999.

[15] J.A.K Suykens, P. Van Dooren, B. De Moor, and J. Vandewalle. Least squares support vector machine classifiers: a large scale algorithm. *European Conference on Circuit Theory and Design (ECCTD'99)*, pages 839–842, 1999.

[16] A. Klausner, A. Tengg, and B. Rinner. Enhanced Least Squares Support Vector Machines for Decision Modeling in a Multi-Sensor Fusion Framework. In *Proceedings of the International Conference on Artificial Intelligence and Pattern Recognition (AIPR-07)*, Orlando, US, July 2007.

[17] A. P. Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society*, 30:205–247, 1968.