Rapid Prototyping of Activity Recognition Applications

David Bannach[1], Oliver Amft[2], and Paul Lukowicz[1]
[1] Embedded Systems Lab, University of Passau
[2] Wearable Computing Lab, ETH Zurich

January 13, 2008

Abstract

The Context Recognition Network (CRN) Toolbox permits fast implementation of activity and context recognition systems. It utilizes parameterizable and reusable software components and provides a broad set of online algorithms for multi-modal sensor input, signal processing, and pattern recognition. It features mechanisms for distributed processing and support for mobile and wearable devices.

We present different case studies indicating its merit in industrial projects, as educational tool for students, and processing engine in activity recognition demonstrators. Moreover, we summarize user evaluation results.

1 Introduction

Today, the development of activity recognition systems is mostly done in two phases. First the recognition method (sensor setup, feature set, classifiers, classifier parameters, fusion methods etc.) is designed. In this first phase experimental data is mostly fed offline into conventional rapid prototyping tools such as Matlab. These tools provide a rich reservoir of 'off the shelve', parameterizable algorithms and visualization methods. Thus, different system variants can be tested quickly without the need for time consuming implementation work.

Unfortunately most such simulation environments are not suitable for actually running applications, especially in mobile and pervasive environments. In general, they depend on custom 'engines' or libraries requiring

large memory footprints and high computing power. Consequently, the implementation of activity recognition applications is mostly done in a separate, second phase. The selected algorithms are implemented in an appropriate programming language and then distributed to specific devices. Issues that need to be addressed include sensor interfaces, synchronization of the sensor signals, and optimization for specific devices (e.g. floating-point or fixed-point calculation).

The Context Recognition Network (CRN) Toolbox (available under LGPL from http://crnt.sf.net) presented in this paper has been developed to combine the two phases and permits quick construction of complex multi-modal context recognition systems, that can be immediately deployed in the targeted environment.

The CRN Toolbox is not a general purpose pervasive middleware such as RUNES [6] or sensor node operating system as TinyOS [9]. Neither it is a high-level framework for rule-based automation, such as VisualRDK [12]. Instead, it is a tool set specifically optimized for the implementation of multi-modal, distributed activity and context recognition systems running on POSIX operating systems. Like conventional rapid prototyping tools, it contains a collection of ready to use algorithms (e.g. signal processing, pattern classification). Unlike classic event detection in homogeneous sensor networks, as DSWare [10], the CRN Toolbox supports complex activity detection from heterogeneous sensors. Moreover, its implementation is particularly optimized for mobile devices. This includes the ability to execute algorithms either in floating-point or fixed-point arithmetic without recoding. With its mature functionality, we believe that the CRN Toolbox will not suffer from limited user acceptance as the Context toolkit framework [8].

The CRN Toolbox contains dedicated building blocks for interfacing a broad range of sensor nodes and support for synchronization, merging, and splitting of data streams. In contrast to the PCOM model [5], which focuses on contract-based spontaneous configuration, the Toolbox relies on a known network topology. Applications can be flexibly distributed among devices (including servers) just by starting the configured Toolbox runtime on the appropriate system. Another important feature is the ability to interface conventional simulation environments such as WEKA (http://www.cs.waikato.ac.nz/~ml). The functionality is accessible through a graphical configuration editor that allows the construction of complex applications by connecting and configuring a set of task icons corresponding to different processing steps.

The concepts utilized by the CRN Toolbox, including graphical programming, data driven computation, parameterizable libraries, and distribution are themselves not new. The contribution of the CRN Toolbox is having

adapted and integrated them in a way, optimal for rapid and efficient implementation of context recognition systems.

Toolbox Concept

The concept of the CRN Toolbox stems from the observation that most activity recognition systems are built from a relatively small set of algorithms. These include sliding-window signal partitioning, standard time and frequency domain features, classifiers, and time series or event-based modeling algorithms.

The key differences between systems are in sensor choice, parameterization of algorithms (e.g. sliding-window size) and data flow. The data flow can be as simple as feeding single-dimensional sensor data to a mean filter and a classifier. This could be a configuration for recognizing sitting and standing from an upper leg accelerometer, for example. It can be as complex as fusing data from tens of heterogeneous sensors, working with different sampling frequencies, different feature computations, and even different classifiers. In such complex systems, sensor subgroups are often handled by different platforms (e.g. different mobile devices and servers for stationary sensors). The implementation must take care of the distributed computation, collection of data and synchronization of the different data streams.

The CRN Toolbox simplifies the implementation of even such complex, distributed context recognition systems to the following three steps:

1. compiling the Toolbox for all platforms that it needs to run on,
2. selecting and configuring the algorithms and data flow for each platform,
3. starting the Toolbox on each platform with the dedicated configuration.

If algorithms should be analyzed that are not present in the current Toolbox implementation, rapid prototyping tools running on a remote server can easily be interfaced.

Figure 1 shows an overview of the CRN Toolbox concept. The step-by-step configuration guide presents a simple example for recognizing kitchen activities from the user's on-body sensors.

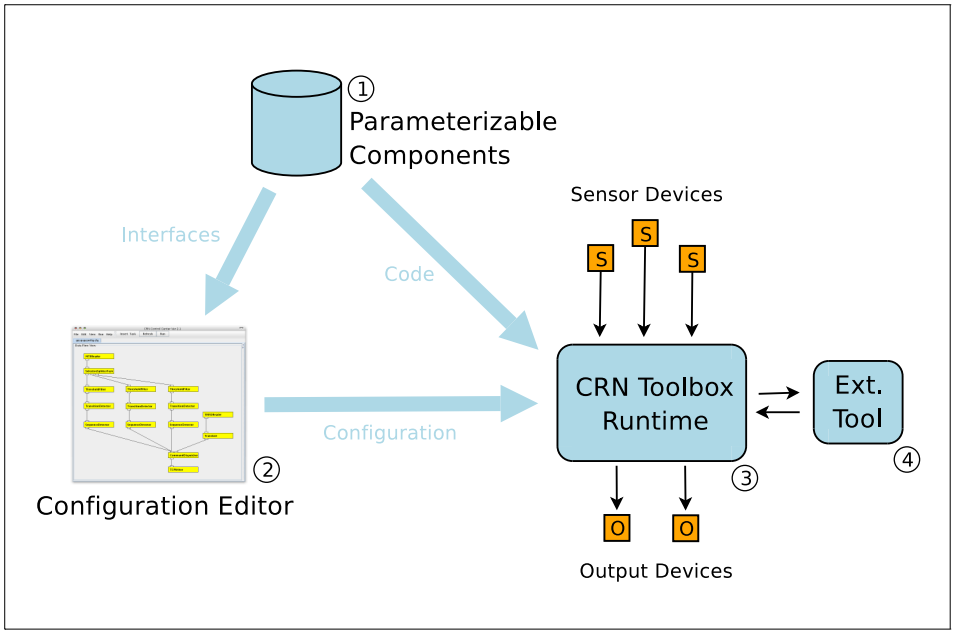


Figure 1: Concept of the CRN Toolbox: (1) a repository of parameterizable software components including I/O device readers and writers, filtering- and classification algorithms, components for splitting, merging, and synchronizing data streams, (2) a graphical editor for specifying data flow and configuring components, (3) the CRN Toolbox runtime environment for online executing the configured software components, (4) arbitrary external tools, communicating with the Toolbox runtime, e.g. live data stream plotting or another Toolbox (local or remote).

Components, but Reusable!

The basic building blocks provided by the CRN Toolbox are the reusable and parameterizable components. Conceptually, the components are active objects that operate on data streams. We refer to them as *tasks*. They encapsulate algorithms, data, and have an individual thread of execution. In essence, tasks run in parallel, waiting for *data packets* to arrive at their *in-port*, process the packet's payload according to their algorithm and parameter settings and provide the modified data packet at their *out-port*. Depending on the configured data flow, subsequent tasks will receive the packet for further processing.

The Toolbox provides *reader*- and *writer* tasks for interfacing with in- and output devices, processing algorithms for data filtering and classification as well as components for splitting, merging, and synchronizing of data streams. A summary of currently available tasks is listed in Table 1. Every task has an individual number of parameters that control its operation. For example, the K-Nearest Neighbor (KNN) classifier task takes the k, a file name with training data, and an optional step-size parameter.

The encapsulation in active objects and the parameterization proved essential for the reusability of the actual code. Hence, for most applications the fact that the Toolbox is implemented in C++ is insignificant, and yet they benefit from the efficient runtime.

The Motor: Runtime Environment and Flow Control

The Toolbox runtime provides the vital environment for tasks to operate. It handles dynamic creation and configuration of tasks as well as configuration of the data flow.

For parameter handling the Toolbox utilizes the JavaScript Object Notation format [7] with an object loader in the 'get instance by name' style. Thus, the Toolbox can be configured at runtime by text-based configuration files that define settings for tasks and data flow needed by the application.

The data flow between tasks is specified through directed *connections* from out-ports to in-ports. Each data packet transmitted along these connections contains data entities belonging to one time instant. The payload of a packet is organized as vector of values from an abstract data type. Moreover, the packets contain a timestamp and sequence number. For combining multiple streams the Toolbox provides *merger* tasks. Mergers combine the payload of packets from separate in-ports and synchronize data streams with different sampling rate.

Data packets are passed by pointer reference along the internal connections through the task network. Packets are cloned only if more than one receiver is connected to the same out-port. This implementation of the runtime core ensures high packet-processing performance. Moreover, we preserved processing performance by providing operations to the task-developer that inherently modify data objects, like the operator '+=' does, instead of allocating new objects.

Table 1: Summary of tasks currently provided by CRN Toolbox. Detailed task descriptions are available as Doxygen web pages from the code repository. The list is constantly growing as more and more users are contributing to the project.

Generic reader (4)	Reading from file, keyboard, TCP-socket, or serial device (including Bluetooth), using a decoder plug-in.
Specific reader (18)	ADS (heart rate), ARSB (walking sensing), BTnode, Hexamite, ID-10 RFID, Lukotronic, NMEA (GPS), MyHeart protocol, SkyeTek M1-mini RFID, Tmote force sensing resistors, Tmote RFID, Tmote magnetic distance, TMSI fiber protocol, Suunto ANT protocol, web interface input, Xsens MT9/MTi, Xsens Xbus, Wiimote
Channel reordering (4)	ChannelSelect, SelectiveSplitterTask, SimpleMerger, SyncMerger,
Filtering (4)	FilterTask, TransitionDetector, VecLen, Einsnorm
Filter plug-ins (15)	ASE (average signal energy), BER (band energy ratio), BW (bandwidth), CG (center of gravity), entropy, FFT, fluctuation, peak, max, mean, median, slope, scale, SFR (spectral rolloff frequency), threshold, variance
Classification tasks (9)	Distance2Position, Hexamite2D, HMM (hidden Markov models), KNN (K-Nearest Neighbor), PCFG (Probabilistic Context-Free Grammars) parser, RangeChecker, SequenceDetector, SimpleHexSensClassification,
Miscellaneous (4)	Synchronizer, Heartbeat, Valve, Nothing
Writer (9)	TCP server, TCP client, serial port, file, console, MyHeart protocol, graph display, image display, nirvana
Encoder plug-ins (9)	ARFFEncoder (WEKA), BinaryEncoder, CmdEncoder, IntLinesEncoder, JSONEncoder, PlottingEncoder, TextLabelEncoder, TimestampedLinesEncoder, SuperPacketEncoder
Decoder plug-ins (4)	ASCIIDecoder, FloatLinesDecoder, IntLinesDecoder, StringLinesDecoder

Synchronizing Independent Data Streams: A Solution

Synchronization of the data streams coming from different sensors is a major issue in multimodal activity recognition. When several independent sensors are used, their data streams should be synchronized to a common starting point.

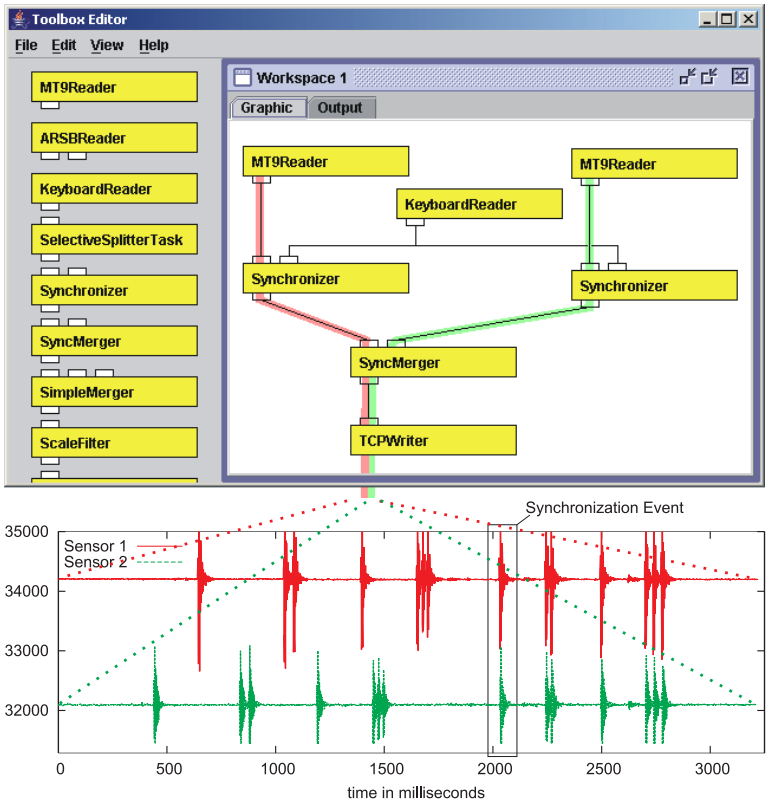


Figure 2: CRN Toolbox graphical configuration editor with a synchronization setup for two Xsens MT9 sensors. The waveforms demonstrates the data alignment achieved at an event detected by the `Synchronizer`s.

A feasible concept for this type of synchronization is aligning streams on events occurring simultaneously in the data from all involved sensors, e.g. a user's jumping up with a set of on-body acceleration sensors. We implemented this concept in the `Synchronizer` and `SyncMerger` tasks. Figure 2 depicts the solution for the example of two Xsens MT9 acceleration sensors. A characteristic high acceleration amplitude was inserted by the jump. The `Synchronizer` tasks detect the peaks caused by these events and adjust data packet timestamps accordingly. The `SyncMerger` combines the data streams by aligning the timestamps. The `Synchronizer` tasks are manually activated, e.g. by a `KeyboardReader`, to limit the alignment phases to controlled time frames. Our initial analysis of the method showed that an alignment of 0.5 seconds and better can be reached.

Readers: Sensor Hardware Encapsulation

In the CRN Toolbox sensor interfaces are implemented as tasks without in-ports, called *reader* tasks. They instantiate new data packets for data samples acquired from sensors (or other sources) and provide them on their out-port. Our architecture supports various reader implementations that can capture different sensors or other sources, such as web pages, application outputs or data files.

For activity annotation, we implemented a keyboard-reader to perform online labeling of data. This reader proved very helpful, since the labeling can be stored aligned with the raw data for later evaluation.

Writers: Communication for Distributed Processing

The key to distributed execution and use of external tools, are *writer* tasks. They forward data received at their in-port to external interfaces, e.g. files, displays, or network connections. For the latter we use `TCPWriter` and `TCPReader` tasks to communicate via TCP/IP sockets. Data packets are transmitted on the channel in a serialized form. The serialization is obtained from an `Encoder` plug-in in the `TCPWriter` task. Similarly, the `TCPReader` uses a `Decoder` plug-in for de-serialization. Thus, two CRN Toolboxes running independently, e.g. on different hosts, can collaborate using the writer-reader communication.

Using this mechanism the Toolbox can link to arbitrary programs based on compatible interfaces. Currently, such interfaces exist for Matlab and WEKA. Both are used for data visualization and pattern recognition in experiments and demonstrators.

Configuration: Easy!

The rapid prototyping capabilities of the Toolbox raised our needs for an easy and quick configuration editor. Figure 2 shows the graphical configuration editor for the Toolbox. Tasks can be dragged from a library into the workspace and interconnected to other tasks with just a few mouse clicks. The Java-based editor produces configuration files for the Toolbox.

Step-By-Step Guide: How to Cook

With the CRN Toolbox building activity recognition applications becomes really easy. For example, it takes only five steps to implement your own kitchen activity recognition including the classifier training. You do not have to write additional code.
The ingredients are: a motion sensor mounted on a glove, a wearable computer or "kitchen PC", and, of course, the CRN Toolbox. In this guide we use the MT9 sensor from Xsens. Typical activities that can be recognized and discriminated by setup include stirring, whisking, cutting bread, slicing onions, and wiping with a cloth.

1. Using the graphical configuration editor, **create a configuration for recording training data** as shown in Fig. 3(a). Begin by adding the `MT9Reader`, to acquire data from the MT9 sensor at 100 Hz and provide all nine channels on its out-port. With the `SelectiveSplitterTask` pick out the channels of interest and send them through a set of filters: `MeanFilter` and `VarFilter` (variance), both operating on sliding windows. Set the window size to 100 (1 sec). With the `SimpleMerger` combine the two data streams again and add the output of a `KeyboardReader` task. This task is used to annotate the recording by keystrokes. Finally, add a `LoggerTask` for writing the resultant data streams into a file.

2. Select an annotation key for training each activity. Connect the sensor and start the Toolbox with the created configuration. Then, wearing the sensor glove, **perform each activity for about 30 seconds**. At the beginning of each activity press its selected annotation key.

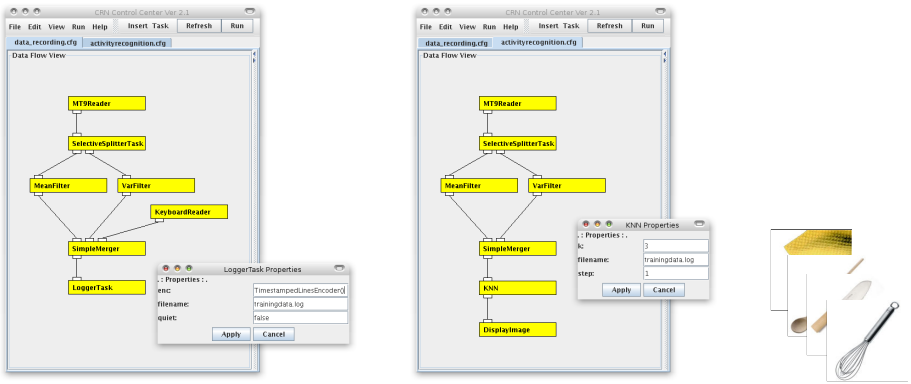
3. **Review the recorded training data** in the log file and reduce it to about 100 samples per activity class. The class label is indicated by the number in the last column.

continued

4. Now, modify the first configuration to **include the classifier and the output task** (see Fig. 3(b)). You may remove the `KeyboardReader`, because from now on the classifier will do the "annotation". Specify the filename of the training data in the properties of the `KNN` task. Attach the `DisplayImage` task to the `KNN` and specify a pictures that should be displayed on the screen for each recognized activity category.

5. Start the Toolbox with the new configuration. Now you can work in the kitchen as you wish and **let the Toolbox track your activities** or, even better, feed the results into a context-aware cookbook. Bon appétit!

To improve the system you may add more sensing modalities such as location, select useful features, and use more sophisticated recognition algorithms.



(a) (b) (c)

Figure 3: Configurations for the kitchen activity recognition. (a) recording of training data; (b) on-line classification and display of result; (c) example output of the classification using `DisplayImage`.

Case Studies

The vitality of a framework such as the CRN Toolbox stems from its continuous development and deployment in various projects. The showcase of applications in industrial projects, student classes, and demonstrators (see Table 2) highlights its maturity and widespread use. During these projects the Toolbox has been successfully deployed on different platforms, including

- Linux (arm32, i386, amd64),
- MacOSX (i386, iPhone), and
- Cygwin (i386).

Here, we depict three case studies from different areas, outlining the utilization of the CRN Toolbox.

Supporting Information Flow in Hospitals

Together with clinical partners in the EU-sponsored WearIT@Work project we developed a solution to improve information flow for the doctor's ward [1, 4]. At the ward round, doctors decide on further treatment of patient under tight time limitations. Access to patient documents right at the bedside would allow the doctor to make decisions based on all available patient information. Notebooks or PCs are impractical for this task, since their operation is time-consuming, distracting, and involves touching non-sterilized devices while in contact with patients.

Our wearable solution simplified the document access. When the doctor comes to the patient's bed, the bedside monitor automatically shows up a document list for that patient. The doctor could then browse these documents by pointing at the monitor and swivel the forearm. The system worn by the doctor consists of the Q-Belt Integrated Computer (QBIC, http://www.qbic.ethz.ch) running the CRN Toolbox, an Xsens motion sensor, and an RFID reader. The QBIC is worn as a belt, the latter two are attached to the lower arm. The patient wears an RFID tag. At each patient's bed there is a bedside monitor to display documents from the hospital's information system.

In the implementation we used a set of three tasks to process each gyroscope axis of the motion sensor. The set contained threshold detection and sequence matching tasks (`ThresholdFilter`, `TransitionDetector`, and `SequenceDetector`). This algorithm can detect forearm gesture sequences such as swivel left, then right (open document command). The `CommandDispatcher` acts as gateway,

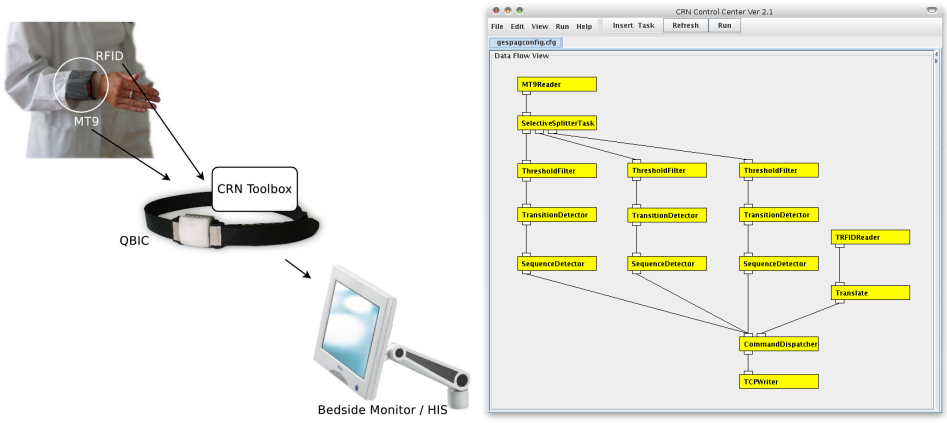


Figure 4: Hospital information support system setup and CRN Toolbox configuration.

forwarding the commands only in active state (controlled by an activation gesture). This task also consumes the patient identification from RFID. Finally, the commands are transmitted (`TCPClientWriter`) to the document browser of the hospital's information system. The Toolbox configuration is shown in Fig. 4.

The complete setup was tested in a two-weeks trial with doctors in an Austrian hospital. Currently we address the system's shortcomings, which were mainly the robustness of the gesture detection and the sensor wearability.

Monitoring Walking Habits

Together with industrial partners in the EU-sponsored MyHeart project we investigated new approaches for preventing cardiovascular diseases and maintaining low disease risks. Since many daily activities involve walking, we developed a walking habits monitor that supports active walking and could track activity intensity.

In our implementation, the QBIC served as a central data acquisition and processing hub, running the CRN Toolbox. The user's activity was monitored with a custom sensing unit containing acceleration and air pressure sensors attached to the belt. Additionally, a heart rate chest belt was connected via Bluetooth. Based on features from the belt sensor unit we classified walking straight, -up, -down, and idle as well as using the elevator up or down. The result along with the heart rate was forwarded to a mobile

phone.

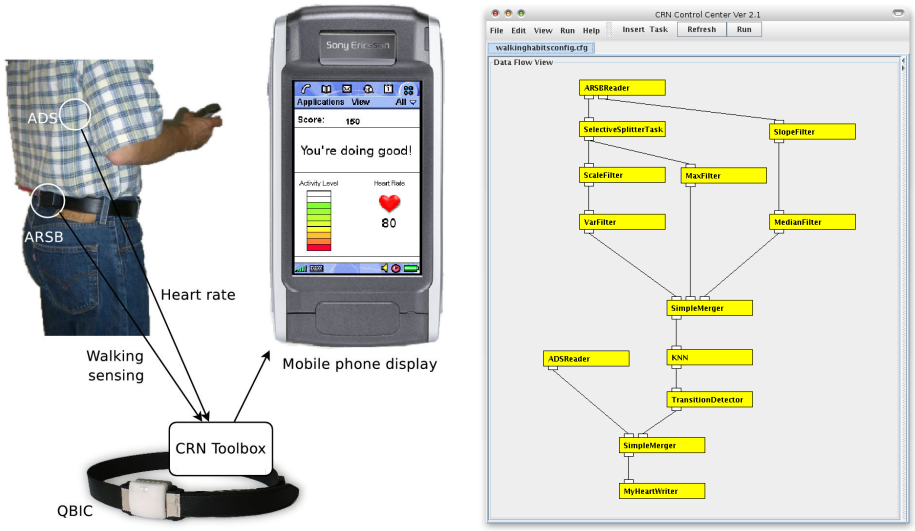


Figure 5: Monitoring walking habits phone visualization and CRN Toolbox configuration.

Figure 5 shows the final Toolbox configuration. This project required readers to capture data from the belt sensor unit (`ARSBReader`) and heart rate belt (`ADSReader`), several features filters, a classifier (`KNN`), and a writer to communicate to the mobile phone application (`MyHeartWriter`).

The visualizations on the phone showed activity level, heart rate and provided recommendations based on detected activities. In our ongoing work we use further sensors at the limbs to capture diverse activities.

A Mixed-Reality Car Parking Game

We designed a car parking game to explore the use of wearable systems in computer games [3].

The game plot features the player helping a virtual driver to fit the latter's virtual car into a parking lot. The player does so by weaving hand and arm gestures while facing a virtual scene at the roadside where a parking spot is available between other vehicles. Figure 6 shows a screenshot of the scene. The driver and car's behavior are simulated and follow the gesture commands. The goal is to perform this guiding task as fast and safely as possible, in particular avoiding collisions with other cars and obstacles.

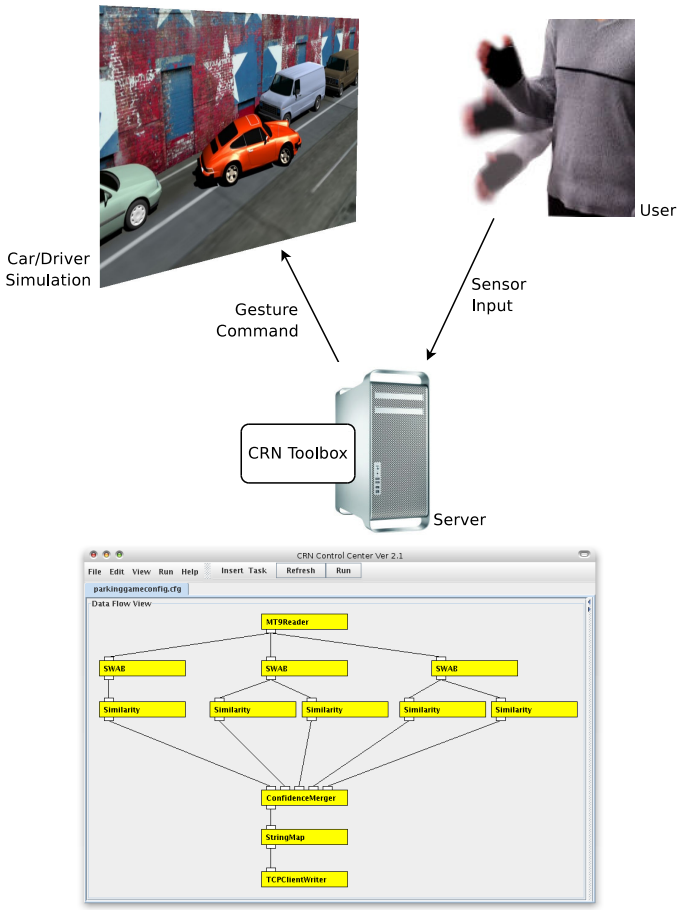


Figure 6: Scene of the parking game and CRN Toolbox configuration.

In this application the CRN Toolbox performs the recognition of gestures from the player's glove. Its task is to the detect five gesture commands in the continuous datastream from the glove: forwards, backwards, turn left, turn right, and stop. We used acceleration and gyroscope sensors in three axes from an Xsens unit attached to the glove. The gesture spotting procedure utilizes an explicit time series segmentation algorithm (`SWAB`), followed by a class-specific feature similarity search (`Similarity`). Finally, the individual gestures are fused (`ConfidenceMerger`). The retrieved gestures are mapped (`StringMap`) to game commands and transmitted (`TCPWriter`) to the game simulation and graphics engine.

The game was used as demonstrator for tutorials and student courses.

We built recognition models for 16 different gestures. Hence, every player could customize the system, by selecting five gestures according to individual preferences. This was easily implemented by exchanging configuration files for the recognition tasks.

User Evaluation

Right from its very first days, the CRN Toolbox has been a community project. The very positive user feedback and the growing number of tasks indicate that our approach is well perceived. However, a thorough quantitative evaluation of middleware and programming tools such as the CRN Toolbox is hard [8].

While we have not yet performed a controlled assessment, we do have some empirical, in some cases even quantitative results that support our view on its usefulness.

Experience with Students

As presented in Table 2 the Toolbox has been widely used in student classes with over 60 students having worked with it.

A class of 19 fourth semester CS students implemented an application with the Toolbox to control the computer games Pong and Tetris by shaking a motion sensor in different directions. A typical solution for recognizing these gestures consists of five Toolbox components (approx. $\sim$1200 lines of code):

- acquire data from sensor
- apply filters (mean, variance)
- classify gestures, using KNN algorithm
- send result via TCP
- manage data flow

The exercise also included the implementation of a new Toolbox task for reading from TCP sockets. The students were just Java beginners and never programmed C++ before, yet with the Toolbox all students were able to solve the recognition problem within 20 hours. Four of them also completed the Java game in that time.

Evaluation of researchers

The CRN Toolbox was used in a activity recognition tutorial at ISWC 2006. The 12 participants were asked to rate their impressions after having worked with it for 4 hours. Ten completed the tutorial feedback form. On average they rated themselves as advanced software programmers with some knowledge of C++, but little experience in context recognition. They reported average durations of 10 minutes (max. 30 minutes) to understand the four tasks of the tutorial, 15 minutes (max. 30 minutes) to implement and run solutions with the Toolbox, and 20 minutes to debug their configuration if needed.

We received many positive comments, such as "good and fast platform for application development", "one can click filters together", "easy to understand, easy to use" and "you don't have to reinvent the wheel". Critics addressed missing documentation materials. Our current work addresses this issue by using automatic documentation tools (Doxygen) and web platforms more intensively.

The CRN Toolbox was developed to ease the process of building activity recognition systems. We believe that its quick adoption by researchers is due to intuitive design of reusable components and data flow mechanism. The spectrum of implemented solutions indicates that our approach is viable in the diverse environments of wearable and server-based applications. Even students managed to implement recognition solutions during class times.

As a framework, the CRN Toolbox introduces processing overhead. A prominent aspect in our design is the between-task communication, required in most useful configurations. It relies on a common packet format to exchange all media types. Besides the payload, a packet contains timestamp, sequence number, and a payload pointer, totally 16 Bytes, independent of the payload size. For a typical scenario, as the hospital support system, raw sensor-data packets have the highest transmission rate. In this example, an `MT9Reader` acquired a 9-channel Xsens MT9, requiring 36 Bytes for one sample. Each sample was sent separately, which amounts to 44% overhead. For packet rates above 100 Hz, such as audio, the effective overhead is reduced by transferring multiple samples in one packet.

Most current applications of the Toolbox do not exploit its distributed processing capabilities. We intend to use this feature in more complex applications in the near future. In our further work, we plan to investigate the

combination with existing pervasive middleware frameworks that often rely on activity and context recognition service, exactly what the CRN Toolbox provides.

Acknowledgments

The authors express their gratitude to all students and researchers that contributed to the development. The project is partly supported by the EU WearIT@Work and MyHeart projects.

References

[1] Kurt Adamer, David Bannach, Tobias Klug, Paul Lukowicz, Marco Luca Sbodio, Mimi Tresman, Andreas Zinnen, and Thomas Ziegert. Developing a wearable assistant for hospital ward rounds: An experience report. In *Proceedings of the International Conference for Industry and Academia on Internet of Things*, 2008. To appear.

[2] O. Amft, M. Kusserow, and G. Tröster. Probabilistic parsing of dietary activity events. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, pages 242–247, 2007.

[3] D. Bannach, O. Amft, K. Kunze, E. Heinz, G. Tröster, and P. Lukowicz. Waving real hand gestures recorded by wearable motion sensors to a virtual car and driver in a mixed-reality parking game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 32–39, 2007.

[4] D. Bannach, K. Kunze, P. Lukowicz, and O. Amft. Distributed modular toolbox for multi-modal context recognition. In *Proceedings of the 19th International Conference on Architecture of Computing Systems*, volume 3894 of *LNCS*, pages 99–113. Springer, 2006.

[5] C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM - a component system for pervasive computing. In *Proceedings of the Second IEEE Conference on Pervasive Computing and Communications*, pages 67–76, 2004.

[6] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis. The

RUNES middleware for networked embedded systems and its application in a disaster management scenario. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 69–78, 2007.

[7] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.

[8] K. Edwards, V. Bellotti, A. K. Dey, and M. Newman. Stuck in the middle: The challenges of user-centered design and evaluation for middleware. In *Proceedings of the Conference on Human Factors in Computing Systems*, 2003.

[9] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Not*, 35(11):93–104, 2000.

[10] S. Li, Y. Lin, S. Son, J. Stankovic, and Y. Wei. Event detection using data service middleware in distributed sensor networks. *Telecommun Syst*, 26(2-4):351–368, 2004. Special issue on Wireless Sensor Networks.

[11] T. Stiefmeier, C. Lombriser, D. Roggen, H. Junker, G. Tröster, and G. Ogris. Event-based activity tracking in work environments. In *Proceedings of the 3rd International Forum on Applied Wearable Computing*, 2006.

[12] T. Weis, M. Knoll, A. Ulbrich, G. Muhl, and A. Brandle. Rapid prototyping for pervasive applications. *IEEE Perv Comput*, 6(2):76–84, 2007.

Table 2: Summary of major projects using the CRN Toolbox. For student class projects, approximate lines of code (LoC) and components count is shown.

Project description	**Utilization of the CRN Toolbox**
Industrial projects	
WearIT@Work supporting hospital information flow [1, 4]: gesture controlled access to patient's document using wrist worn motion sensor. Using RFID for patient identification.	Data capturing, gesture recognition, control of hospital's document browser, running on QBIC (Linux/arm32). Several demonstrators and test system built. A hospital trial was conducted.
WearIT@Work production support: activity recognition of car assembly and maintenance [11]. The project utilizes inertial motion and indoor location sensors.	Recording multimodal sensor data: Xsens, Hexamite, Ultrasound, muscle force; various demonstrators. Platform: Linux/i386.
MonAMI dynamic monitoring services	Dynamic re-configuration of the Toolbox depending on available sensors and registered services. Platforms: Linux/i386 and Linux/arm32.
MyHeart walking habits: online classification of walking activities and intensities to support active lifestyle and improve fitness.	Acquisition of heart rate, acceleration, and air pressure; classification; streaming results to a mobile phone and professional coaching center, running on QBIC (Linux/arm32).
NESD location tracking: GPS-based local map visualization	GPS position logging (NMEA protocol) and conversion for dynamic map display, forwarding to central mission server, running on QBIC (Linux/arm32).
Student classes and -projects	
ISWC 2006 tutorial "Hands on Activity Context Recognition": building a gesture recognition system for controlling a simulated car parking game with real waving gestures, 12 participants.	Testing of algorithms and gesture types using simulated data streams from a motion sensor glove. Components: 9, LoC: 7000. Platform: Linux/amd64.
Number entering game: entering binary digits using a motion sensor only, practical exercise and competition for ambient intelligence lecture, 15 students in 5th semester.	Understanding of algorithms, modularization and interfacing to sensor data. Components: 7, LoC: 2000. Platform: Linux/i386.
Location estimation and activity recognition: ultrasonic- and motion sensors practical exercise for ambient intelligence lecture, 12 students in 5th semester.	Understanding of algorithms and challenges of location tracking. Components: 7, LoC: 2000. Platform: Linux/i386.
Interactive World: software project to implement gesture control for games (Pong, Tetris), 3 days, 19 students in 4th semester.	Implementation of a TCP reader task, utilization of KNN classifier (gesture recognition). Components: 5, LoC: 1200. Platform: Linux/i386.
Activity monitoring: training a classifier to recognize human activities (sitting, standing, walking, running) and visualizing results, 10 students in 4th semester.	Learn the concepts and operation of a classifier, implement and testing. Components: 7, LoC: 2000. Platform: Linux/i386.
Demonstrators	
Parking Game: controlling a virtual driver and car with real hand gestures in a parking game [3].	Capturing glove-based inertial sensor data, gesture spotting using explicit segmentation, gesture event search, and fusion steps; controlling the game visualization engine. Platform: Linux/amd64.
Parsing of dietary activity events using probabilistic context-free grammars (PCFGs): inference of food intake cycles from activities [2].	Simulation of activity event input, PCFG parsing, reporting of results. Platform: Linux/amd64.
Hammering-screwdriving demo: recognizing assembly activities (hammering, screwdriving, sanding, vising) with a motion sensor in a glove.	Xsens motion sensor capturing, classification of activities, display of recognition result on screen and wireless connection, running on QBIC (Linux/arm32).