A Dynamic Environment Modelling Framework for Selective Attention

Thorsten Buchheim, Georg Kindermann, Reinhard Lafrenz, and Paul Levi
Institute of Parallel and Distributed Systems,
University of Stuttgart, Universitätsstr. 38, D–70569 Stuttgart
Phone: +49-711-7816-229, Fax: +49-711-7816-250
{buchheim,kindermann,lafrenz,levi}@informatik.uni-stuttgart.de

Keywords: Selective Attention, World Modeling, Communication, Multi-Agent-Systems

Abstract

Sensor data evaluation and environment modeling represent a central part in the design of robotic systems. The environment modeling process usually is done by several data processing modules by means of sensor data fusion, combination of a priori knowledge with currently sensed data, etc. Depending on the application scenario there is usually a trade-off between accuracy and time consumption of the environment modeling process. Some domains require a quite accurate environment model and processing time plays a secondary role. More reactive domains, however, demand for rather short cycle times, even if this results in a less accurate model. For some applications the required degree of accuracy may vary depending on the current situation or task to perform and some parts of the sensed information may be more important than others. Several architectures were proposed to address this problem by special mechanisms which schedule the available resources according to the needs of the current tasks which is usually denominated by the term "selective" or "focused" attention. In our work we will show how these concepts can be put into practice by a flexible and open software design which is applied for our *CoPs* RoboCup midsize team.

1 Introduction

Sensor data evaluation and environment modeling represent a central part in the design of robotic systems. To act and react properly on different situations an autonomous robot usually is equipped with various sensors to gather information about its surrounding which it uses to generate a model of its environment.

The environment modeling process usually is done by several data processing modules by means of sensor data fusion, combination of a priori knowledge with currently sensed data, etc. Most of the environment modeling approaches are structured in a hierarchical manner. While the lower evaluation levels usually extract simple features from the sensor raw data, like color blob detection in a 2D camera image or line extraction from a distance scan of a laser range finder, the higher levels use these data to construct a more or less detailed map of the environment.

Most times the environment modeling framework is designed once for the application domain and the underlying sensor and actuator setup of the robotic system. However, there are application scenarios that require a high degree of flexibility within the environment modeling process regarding sensor equipment as well as the evaluation algorithms. Prior experiences with multi agent based software architectures have shown the usefulness of a high flexibility of multi sensor systems like for a multi agent based system for optical inspection [Buchheim *et al.*, 2000]. Here, different sensors and evaluation mechanisms were designed as separate agents which cooperatively solved the task of quality inspection of workpieces. A similar approach was also used for our first RoboCup team software design [Lafrenz *et al.*, 2000].

Another important aspect for autonomous systems is the time efficiency of the environment modeling process. Depending on the application scenario, there is usually a trade off between accuracy and time consumption. While some domains require a quite accurate environment model and processing time plays a secondary role, e.g. in case of a robotic museum tour guide [Thrun, 1997], more reactive domains demand for rather time efficient algorithms.

According to the current situation or task to perform the required degree of accuracy may vary and some parts of the sensed information may be more important than others. Several architectures [Langley *et al.*, 1991; Tsotsos, 1997; Gat, 1991; Hayes-Roth, 1991] were already proposed to address this problem by special mechanisms which schedule the available resources according to the needs of the current tasks. These mechanisms are usually denominated by the term "selective" or "focused" attention.

Works like [Gat, 1991] and [Tsotsos, 1997] proposed extensions to Brook's subsumption architecture for attentive processing [Brooks, 1991]. In [Tsotsos, 1997] this is achieved by allowing behaviors not only to act in the physical world but also to manipulate its internal representation. A behavior is defined as a process which takes input in one or more representations and creates or manipulates other internal or external representations for subsequent processing. Gat's ATLANTIS architecture [Gat, 1991] defines three layers for control, sequencing, and deliberation where the sequencing layer selects the stimulus-response mapping by activating and deactivating specific modules in the control layer. The THEO architecture [Mitchell *et al.*, 1991] provides a means of selective attention by introducing the concept of eager and lazy sensing. While eagerly sensed features are updated constantly with each "sense-decide-execute" control loop, lazily sensed features are deleted after the evaluation step and only re-sensed when explicitly needed.

All of these works motivate clearly the necessity for such mechanisms and present concepts for them. In our work we will focus on how to put these concepts into practice by a flexible and open software design which allows for an easy integration of all these methodologies.

We will introduce a flexible and modular framework for environment modeling which neither imposes restrictions on the type or amount of data to be stored nor the data processing or data generating elements (like sensors) and furthermore provides functionality to control the time cycles for data generation or processing depending on the current task or action of a robot. We then present a successful application of this framework for the RoboCup domain.

The paper is structured as follows: Section 2 presents the data management concept of the framework. Section 3 introduces the data acquisition and processing elements as well as the time cycle control mechanisms used for selective attention. The integration of this framework within the communication framework of the *CoPs* robotic soccer team [Lafrenz *et al.*, 2002] of the University of Stuttgart will be shown in Section 4. Section 5 then illustrates an application scenario of the complete framework within the *CoPs* -team. Section 6 concludes and gives an outlook to future applications and improvements of the framework.

2 Data Management

One of the central issues of the world modeling process is the representation of the data supplied by the various sensors and data processing modules of the system.

The focus of our world modeling design was to achieve a maximum degree of scalability and extendibility which allows for an easy integration of new sensory devices and evaluation algorithms. Secondly our work concentrated on performance issues like minimal memory consumption and least possible data volatility due to frequent memory allocation during run time. All of this is accomplished by a flexible data container concept which handles generic data objects of any arbitrary type via a common interface which will be described in the following.

2.1 Data Representation - Containers and Data Objects

In our concept *Data Containers* represent the key elements of data storage. They aggregate a number of so called *Data Objects* which hold generic (environment) data elements and provide administration functionality like adding, locating and accessing stored *Data Objects*.

The concept of data containers was first applied for a RoboCup simulation environment for the F2000 league [Kleiner and Buchheim, 2003] to facilitate the storage of generic data objects in a plug-in based architecture. Here it was also proposed as a uniform way of data representation for the development of generic sensor evaluation algorithms for RoboCup.

A *Data Container* stores all environment information of one robot which is represented by a set of different environment features. Each feature is stored within a separate *Data Object* and can vary in its degree of abstraction (e.g. sensory raw data like a simple camera image or top level information like an absolute localization in the environment).

The data objects are identified by the following two attributes

1. the data source, the data stems from, e.g. a specific sensor or a data processing module and
2. the data feature the object represents, e.g. a red color blob or a relative ball position.

The data representation itself is done by so called *Data Units (DU)* which form the core of the *Data Object*. A *Data Unit* stores the data itself in any arbitrary format along with its time stamp. Each *Data Object* maintains a history list of the n past *Data Units* over time (where n is a configurable parameter). This data history can be essential for data evaluation purposes when e.g. data observations are made over time.

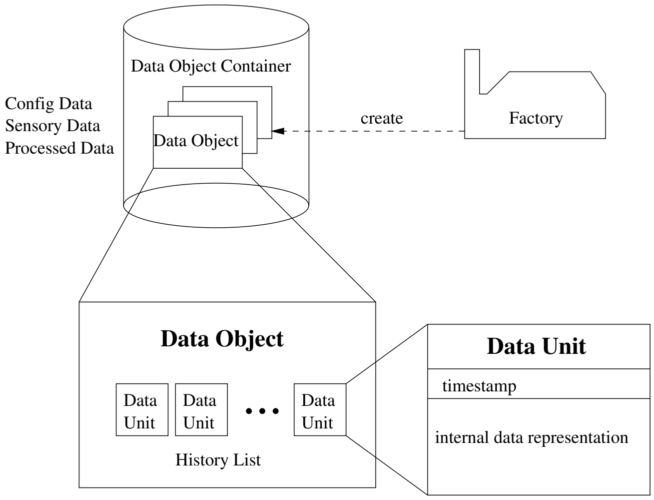

Figure 1: Data Container Concept

The creation of *Data Objects* is done according to the factory pattern [Gamma *et al.*, 1995]. Each new *Data Object* has to be registered with the factory by giving the two above mentioned data object identifiers along with a data descriptor defining the data representation used for the *Data Unit* part of the *Data Objects*. Within the data descriptor the underlying basic data type and further information like the number of entries, data fields per entry and history length are provided. After registration, the *Data Object* can easily be added to a *Data Container* just by providing its unique identifiers.

Since most software approaches for autonomous robots use multiple threads to simultaneously process incoming data, much attention must be paid to mutual exclusion for shared data objects. These necessary data locking mechanisms sometimes bear negative side effects on the overall performance of a system, when e.g. a data processing algorithm locks data for an extended period of time while modules of another thread try to access it for reading.

For this reason *Data Objects* provide read and write slots to update or access their data and automatically implement the necessary data locking functionality. To simultaneously read and write data objects each *Data Object* contains besides its history list of *Data Units* one additional *Data Unit* as "spare element". When requesting a write slot a pointer to this spare element is provided and this Data Unit is exclusively locked for the update process, while all other Data Units remain readable. When the update process is finished the data slot is released and the spare element is inserted at the front of the history list, while the last element is extracted and becomes the new spare element.

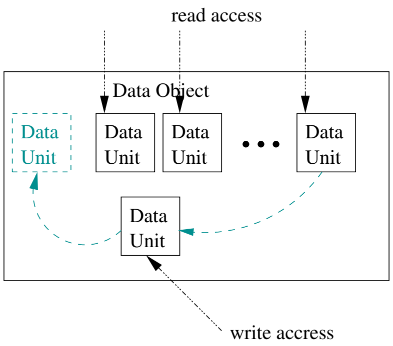

Figure 2: History Mechanism

In the CoPs team player architecture one *Data Container* is used to store the locally sensed and interpreted data of one robot which we call the robot's egocentric environment model. Currently data of the following categories are stored

- sensory (raw) data
- results of sensor data processing (sensor fusion and merging)
- control input, for RoboCup e.g. referee decisions
- control and configuration parameters affecting the behavior of some modules or the robot.

Usually, in a multi robot domain like RoboCup the robots partially share their data to either improve their view on the environment or to synchronize their actions. In our architecture we use a separate *Data Container* for each active robot transmitting data to store the communicated data. These communication *Data Containers* and the egocentric environment model form the complete world model of one robot of the *CoPs*-Team. Further details on the inter-robot-communication will be given in section 4.

3 Data Acquisition and Processing

The prerequisite for modeling the environment is the ability to sense it. Most robotic systems use multiple sensors to mutually compensate drawbacks of single sensors. The sensor data usually is processed in some way or fused with other data e.g. stemming from other sensors to obtain an accurate model of the environment. Both, sensors and processing algorithms should be easily integrateable into the system and should impose least possible restrictions for the developer. Within our framework *Sensors* and *Data Processors* are defined as so called *Information Sources* which manipulate *Data Objects* of the world model. *Data Processors* additionally require some input data to process which is retrieved from other *Data Objects*.

In the following, we will describe the sensor classes and the data processing units in detail and show the interplay with the world model.

3.1 Information Sources

Sensors and *Data Processors* represent the *Information Sources* of the framework which manipulate dedicated *Data Objects*.

Both, sensing and data processing are often parallelized by using multiple threads to avoid busy waiting for a sensor or an algorithm to provide new data. However, there may be situations where a fixed order of the incoming data is desired.

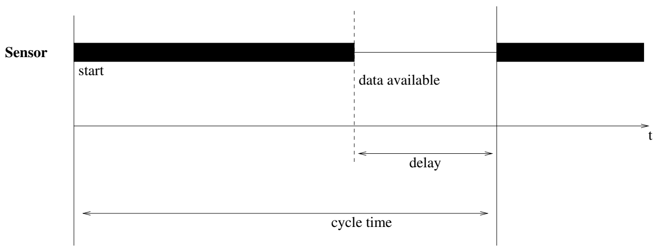

Figure 3: Sensor Cycle Time

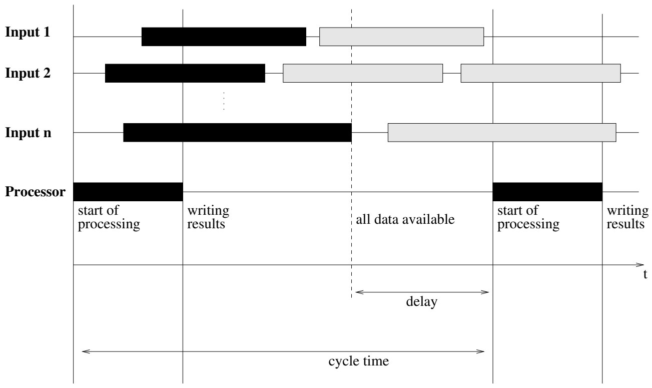

Figure 4: Data Processor Cycle Time

Our approach accounts for both requirements. For *Sensors* and *Data Processors* it is configurable whether they are run in an individual thread or in a so-called "common thread", where they are processed sequentially according to a fixed order.

Since some of the components are very fast and others require a longer time to produce new data, we can configure a cycle time for each thread. In opposite to cycle times in real-time operating systems, we don't have a *maximal* time consumption for each cycle but a *minimal* one, so we can avoid unnecessary checks, e.g. for a new video frame. If the frame rate is known, we are able to configure an image analysis component with a cycle time of the frame rate. The cycle time mechanism is shown in figures 3 and 4.

Attention Selection

By dynamically configuring the cycle times, it is possible to concentrate on specific perceptions. This means for example that a driving robot can be configured for short update cycles of the self-localization, whereas for a standing robot there is no need to constantly update its position once a reliable and accurate estimate has been found. The main advantage of this mechanism is that it can be used for attention selection. It is a very powerful tool for concentrating the available resources depending on the current situation.

Sensors

Sensors possess a pure data generation functionality. Sensor objects must automatically register their individual *Data Object Type* with the factory when they are created. During the registration process, additional parameters like the size of a single data element, the number of elements in a data set and the length of the history to store is given. With this information, the world model creates the required dedicated *Data Object* for the sensor which is then added to the *Data Container*. The *Data Object* can then be accessed via the *Data Container* by the name of the sensor and the feature type. The feature types themselves are independent of the underlying physical sensor, so that several sensors can provide similar features, e.g. lines relative to the robot in a Cartesian coordinate frame can either be provided by a camera or by a laser range finder.

Data Processors

Data Processors can be integrated into the world model in the same way as *Sensors*. Each processor is connected in two ways to the world model: for reading and writing. It can read *Data Objects* of *Sensors* or other *Data Processors*. Then it writes the result of its own calculations into its own dedicated *Data Object* of the world model like the *Sensors*. That means, that data from other sensors are fused or transformed into information on a higher level of abstraction.

One example of a *Data Processor* could be a virtual line detection sensor, which yields the geometrical description of boundary lines in the environment. These objects can be e.g. lines drawn on the floor, or walls. To detect these types of lines, one needs different physical principles of measurement. The lines on the floor are reasonably detected by a vision system, whereas the walls are detected by a laser range finder. Both information can be merged by a *Data Processor* implementing a so called "virtual line sensor".

Another area of application is e.g. plausibility checking. In this case, the information of several different sensors can be used to filter out wrong information. Often, complementary information is useful. A laser, for example, is not suitable to detect glass walls, whereas an ultrasonic sensor is. The combined information of both gives a more reliable model of the environment.

3.2 Data Driven Execution Cycles

The sensing mechanism follows a publisher-subscriber model [Gamma *et al.*, 1995]. In this context *Data Objects* act as publishers which *Data Processors* or in some cases *Sensors* can subscribe to. In case of an update of a *Data Object* all observing elements are automatically informed. Usually *Data Processors* subscribe to the *Data Objects* they need for their data processing. This is useful if data processing shall only be done when all input data is available or when the data flow of data processing mechanisms is to be monitored. *Sensors* usually only register for *Data Objects* containing configuration information to facilitate changes of the configuration at runtime.

Figure 5 shows an exemplary collaboration of *Sensors*, *Data Objects* and *Data Processors*. The publisher subscriber relation is marked by the dot ended lines. The filled and empty dots show the notification state of each *Data Processor* for the corresponding *Data Object*. In this example it is assumed that each *Data Processor* only initiates its processing step when all *Data Objects* are updated, thus yielding a data driven execution cycle.

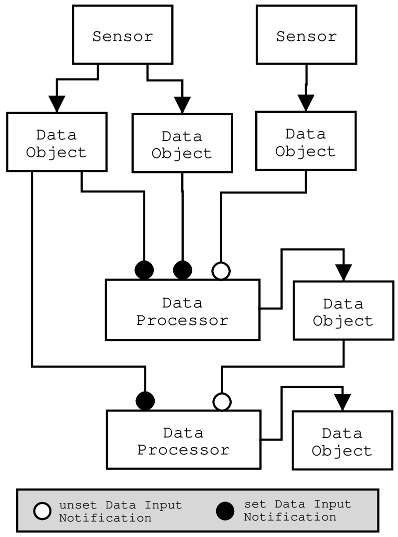

Figure 5: Data driven execution

One example application is the self-localization by Markov or Monte Carlo methods. Here, different types of sensor features are used and the likelihood of being at a specific place is calculated dependent on the sensory information. A Monte Carlo *Data Processor* thus registers with all relevant *Data Objects* and uses this information to update its dedicated *Data Object* containing the absolute position.

3.3 Configuration

Each *Data Processor* and *Data Container* possesses an individual Configuration-*Data Object* which it is automatically subscribed to. As one essential part this configuration contains the already mentioned configurable minimal cycle time.

With this mechanism it is possible to tune the system so that the sensors are evaluated at a time where the new data is available. In case of a camera sensor, the minimal cycle time should be at least the video frame rate. In an ideal case, each sensor runs in a separate thread at approximately its hardware cycle time. In this case, this does not only assure the most recent available information but also a constant update rate ("heart-beat") for each sensor. This is especially useful for time-dependent calculations, for instance Kalman filtering for optimal state estimation. For the attention selection we can configure this heart-beat dynamically depending of the current role that is currently assigned to the robot [Lafrenz *et al.*, 2000].

4 Communication

For team communication in our *CoPs* architecture we use a publish/subscribe event channel approach [Buschmann, 1996]. The advantage of this concept is that the supplier and the consumer of information do not need to know each other (decoupling) and only the used channels have to be known. The communication framework allows to connect to one or more event channel(s) as supplier (publisher) and/or consumer (subscriber). This allows to implement different levels of communication like differently prioritized or thematically separated channels.

4.1 Communication Levels

Our implementation uses a CORBA event channel [OMG, 1995] using the push model which means that a supplier pushes its information to a channel. The channel is realized by a server process which runs on one of the robots or on an external computer which broadcasts the provided information to the subscribers.

The modules pushing the data to the channels can be implemented within an own configurable thread or a common thread which is the same mechanism we already described in section 3.1. The common thread is normally used for triggering the transmission of *Data Objects* whenever they change, while individual threads are normally used to provide a constant "stream" of information with a configurable cycle time.

At the moment we use three communication levels, according to the known reactive, tactical, strategic level approach:

1. Information acquired from the local sensors
2. Derived data from local view and the received data from level one
3. Data processed by observing the local view and the information of level 1 and 2 over an extended period of time.

This enumeration also reflects the frequency of data exchange. In our application each robot provides a snapshot of its local view approximately every $250ms$ and is implemented by an own thread (level 1). Pushing the data of the other levels is implemented within the common thread and so dependent on how often the information is updated. Information of level 2 usually is exchanged about every $5 - 10s$ while information of level 3 is normally transmitted about every half a minute or more.

Level one data is intended to be used to extend the local view derived from the information of the own sensors. Each robot makes its own decision about what action he should execute next, based upon its knowledge of the *real world*. Restricted to the local view of a robot cooperative playing is a difficult task which can be considerably reduced in its complexity by the use of communication. For example a decision which robot of a team should approach the ball would require that each player performs a complete analysis of the current situation concerning the positions of all team mates as well as the ball to determine the player next to the ball. With each robot exchanging its own current distance to the ball with its team mates a solution can easily be found.

Level two data should be information which changes less frequently, e.g. the role of a robot (defender, attacker) or an estimation of the situation. Another point is providing additional information for other robots. For example a defender is often off duty while the ball is near the opponent goal. So our concept allows to change its attention by reducing the processing time for normal situation analysis and investing it for other purposes. So more sophisticated, time consuming analysis on its own sensor and/or the communicated information can be done. For example the robot can provide a (rough) estimation about the absolute position of the attacking robot, which is normally crowded by opponent robots and so not able to localize itself. This can be done for example by considering the communicated relative ball position of that robot and the own detected absolute ball position.

Level three could for example be used for negotiations and conflict resolutions. Considering the defender scenario mentioned above, this defender could detect some inconsistency, for example an incorrect localization of the attacker due to a sensor malfunction. At the moment this level is mostly used to signal possible problems to the human coach.

4.2 Communication Data Handling

Information received from other robots is stored in one *Data Container* for each robot. If a new robot enters the game, it registers itself at the different channels. So if a robot pushes its information to a channel for the first time the registered consumers receive this information. For storing the information of the new robot the others create a new instance of a *Data Container* and add the name of the new robot to their list of active robots which is stored in a *Data Object* within their local *Data Container*. This local *Data Container* builds the egocentric environment model of a robot (see section 2.1). It stores the information acquired from the local sensors and data processing results from communicated or sensory data. The information received from other robots is separately stored in so called *Communication Data Containers*. The *Data Objects* within this container are only updated by the communication module and read by *Data Processors* to compute information which is stored within the local, egocentric *Data Container* (see figure 6).

The software design allows for an easy implementation and integration of new *Data Processors* for the communicated information. The features of the implemented publisher/subscriber model we described for the *Data Objects* in section 3.2, are also provided for the *Communication Data Objects*. By using these mechanisms it is easily possible to implement *Data Processors* which are initiated each time a

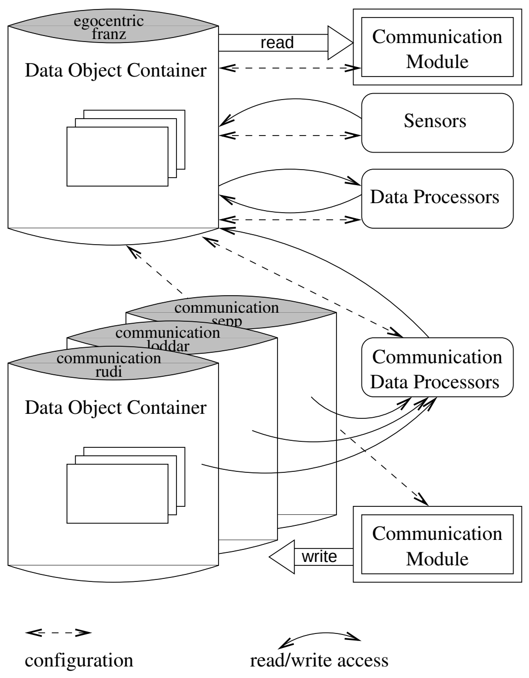

Figure 6: Communication Containers

robot sends a specific information or just when this information item is received from all active robots.

For administrative reasons, e.g. to detect that a robot dropped off, a *Data Processor*, the *active robot observer (ARO)* is started to observe the list of active robots and the containers holding the appropriate communication data. It is expected, that information of each robot is received regularly. So if no information is received from a robot for some time period, this robot is viewed as not active anymore and removed from the local list of active robots. Additionally another datum is updated, which is registered to be transmitted (normally level 3), to inform the other robots and the human coach about the possible drop off. Normally all *Data Processors* for communicated data are registered with the *Data Object* holding the list of active robots. So if the ARO removes the robot from the list, they are informed and can initiate an appropriate reaction.

4.3 Selective Attention for Communication

Selective attention can be achieved on the supplier as well as on the receiver side. For the supplier it is possible to configure the *Data Processors* producing the data to be transmitted and the communication modules (e.g. thread cycle times) responsible for sending it. This way it is possible to adjust the amount, quality and focus of information it provides depending on the current situation.

On the receiver side, it is possible to selectively choose which data from which robot or group of robots should be examined or ignored, by appropriately configuring the responsible *Data Processors*. So it is possible to control how much attention the receiver pays to which information provided by different robots.

For example the attacking robot can normally ignore information from other robots, which is of low plausibility and importance, while a defending robot, can do additional plausibility checks on the provided information, try to detect problems and initiate a conflict resolution procedure if necessary.

5 Selective Attention in the *CoPs* robotic soccer midsize team

Robotic soccer is a domain which usually requires both, fast algorithms for quick reactive behavior like dribbling or intercepting a ball and a high accuracy which is needed for strategic team behaviors, like e.g. defending the own goal area or for strategic positioning. Fortunately, most situations do not require both qualities at the same time which is the point where selective attention comes into play.

In our current approach of selective attention in the *CoPs-Team* we control the activity of the *Data Processors* of the environment model in two ways. Firstly by manipulating individual cycle times as described in section 3 and secondly by defining a special data input for explicitly triggering the execution of a particular *Data Processor*. This special "trigger input" is used like any other regular data input and has to be set to initiate an execution cycle. This allows to trigger special data processing routines "on demand".

Currently we use a role based control of selective attention for our team. In our team architecture there are three top level roles a player can be in:

- The *active player* is the one who usually goes for the ball or currently possesses it and tries to mark a goal.
- The *supporter* assists the active player by following him to catch the ball if the active player fails in executing his current action (e.g. rebound shot).
- The *defender* defends the goal the best way possible according to the current situation.

Here, the active player requires the highest reactivity to get or control the ball while the defender usually can use its resources to obtain a more accurate model of the environment. The roles are switched dynamically which means that when an attack of the opponent team occurs the defender automatically becomes the active player when the ball gets into the vicinity of the own goal.

Each robot player of the *CoPs* team is currently equipped with the following sensors:

- *SICK* laser range finder (LRF) providing distance data in a 180° angular field of view with a 1° resolution which is directly fed to a line and obstacle detection algorithm
- 2D camera detecting color blobs of the ball and goal colors
- a ring of 16 sonar sensor modules each providing one single distance measurement within their sensing range
- an odometry sensor of the Nomadic Scout robot platform, providing a position (x, y, θ) within an absolute odometry coordinate frame which, however, is biased by slippage of the wheels during the motion of the robot. Additionally the current wheel speeds are measured by taking the momentarily applied motor currents.

The LRF and the camera are designed each as *Sensor* threads with controllable cycle times. The odometry data and the ultrasonic sensors are updated each time the scout hardware platform is accessed when e.g. setting the wheel speeds of the robot.

Several data processing units process the incoming data and provide some meta level information on the environment or auxiliary information to support the decision making process for player actions which is done within a separate action execution layer which will not be described any further for the sake of simplicity. Some important sensor data processing elements are:

- Transformation of relative data into the odometry coordinate frame
- Kalman filtering of ball data
- Object tracking of laser detected objects
- Estimation of an outer bound of action
- Path Planning for obstacle avoidance
- Monte Carlo Localization.

A central problem within the midsize league is to determine the absolute position within the field. Besides the obvious advantage of absolute positioning for strategic team play, there is a further important aspect of a fixed coordinate frame which allows a motion-invariant representation of sensed data. Many algorithms like object tracking rely on the fact that the data is provided in a stable coordinate frame as given by a static observer. This data is then used to derive some motion information by matching successive measurements over time.

Since a reliable and fault tolerant localization mechanism cannot be assured at all times, we recurred to the absolute coordinate frame of the odometry sensor. Although this coordinate frame slightly shifts and rotates over time due to the slippage of the wheels, it turned out to be sufficiently stable for the limited time frame in which the object tracking in RoboCup usually is done.

One processing element is thus concerned with the transformation of incoming sensor data from the camera and the LRF to the odometry coordinate frame. One *Data Processor* applies Kalman filtering on these data to stabilize the ball position estimate and calculate its motion vector, while another processor does the same for obstacles detected by the laser.

Path Planning is used when a direct way to an intended target position is obstructed by obstacles. In the absence of an absolute localization, it is useful to define an outer bound for the path planning process, to avoid planning outside the field borders. To achieve this a *Data Processor* buffers detected lines from the laser within the odometry coordinate frame over a dedicated time frame which is then used for a guess of an outer bound where planning should take place.

We chose to implement the path planning as a separate *Data Processor* instead of integrating it in the action execution layer. The path planner is triggered on demand by the action execution layer, when needed to navigate around obstacles to reach a given target position. After having planned an appropriate path, the path planning *Data Processor* constantly checks whether the current path is still valid or if a replanning has to be done due to the motion of certain obstacles.

Although all these approaches work nicely, effective team play demands for an absolute localization. With an absolute position within the field the robots can share this information to organize their behavior by e.g. choosing appropriate positions within the field or assigning non-conflicting roles to each player. For the absolute localization we use a Monte Carlo approach which uses the detected lines of the laser and the color blob data of the camera to estimate an absolute position in the field. This algorithm, however, has the drawback to be rather time consuming depending on the number of features used for localization and the intended accuracy of the position estimate. Hence, localization is done as seldom as possible. For the rest of the time a reliable estimate is propagated forth by the odometry updates and Kalman filtering.

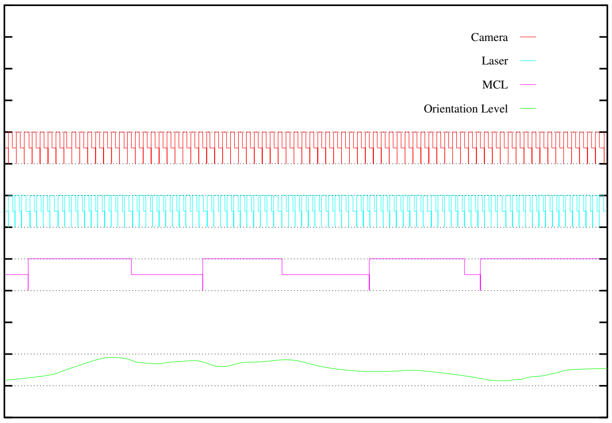

Figure 7: *Sensor and processing activity for active player role*

In our approach we partially renounce on a good absolute positioning for the active player in favor of a higher reactivity while the defender and supporter focus on determining an exact position within the field. However, there are situations when the degree of information for a current action falls below a critical limit and leads to a state of disorientation, like the occlusion of the opponent goal when dribbling the ball and a missing absolute localization within the field.

In these cases some resources have to be spent for the localization at the expense of a temporary lower reactivity. For the more passive roles, on the contrary, once a good position estimate has been found the activity of other *Data Processors* can be intensified like ball or opponent tracking.

To account for this we define a quality measure for the current state of the self localization (*orientation level*) which is used for determining the cycle times of the various *Data Processors*. This quality measure is derived from the Monte Carlo localization by the given probability of the best position estimate, its variance, and by a distance measure to other concurrent position hypotheses. Each role defines a minimum value for this localization measure which should be maintained during its execution time. The selective attention control mechanism then tries to appropriately control the cycle times of the various *Data Processors* to assure the needed level of orientation.

Figure 7 shows the activity of the camera and laser sensor together with the Monte Carlo Localization (MCL) *Data Processor* and the current level of orientation while the robot dribbles the ball. Each *Data Processor* in the diagram has three states, inactive (zero level), waiting (0.5 level) and busy (1 level). The diagram clearly shows a lower activity of the MCL *Data Processor* while the level of orientation remains above a certain level. This activity increases when the state of orientation gets critical by shortening the waiting period. On the other hand the laser and camera update cycles are relatively short to achieve a high reactivity.

6 Conclusion and Outlook

In this work we presented a flexible and generic software framework for world modeling which enables an easy realization of selective attention mechanisms. Until now we merely focused on the software realization to facilitate these mechanisms and applied them for our RoboCup midsize team architecture. Currently the selective attention modeling is done manually based on the current action and situation of a particular robot. For the future we intend to integrate learning mechanisms for attention control which automatically determine which features of the world model are most relevant in certain situation. In this context we will especially focus on cooperative sensing to cooperatively build a consistent and accurate world model by means of communication and load balancing among team mates.

References

[Brooks, 1991] R. A. Brooks. Intelligence without representation. *Artificial Intelligence Journal (47)*, pages 139–159, 1991.

[Buchheim *et al.*, 2000] T. Buchheim, G. Hetzel, G. Kindermann, and P. Levi. A multi-agent approach for optical inspection technology. In *The Sixth Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, 2000.

[Buschmann, 1996] Frank Buschmann. *Pattern-Oriented Software Architecture. A System of Patterns*. John Wiley & Sons, 1996.

[Gamma *et al.*, 1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, 1995.

[Gat, 1991] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. In *SIGART Bulletin 2*, pages 70–71, 1991.

[Hayes-Roth, 1991] B. Hayes-Roth. Architectures for intelligence. In *SIGART Bulletin 2*, pages 301–321, 1991.

[Kleiner and Buchheim, 2003] A. Kleiner and T. Buchheim. A plugin based architecture for simulation in the f2000 league. submitted to RoboCup Intl. Symposium, Padua, 2003.

[Lafrenz *et al.*, 2000] R. Lafrenz, N. Oswald, M. Schule, and P. Levi. A cooperative architecture to control multi-agent based robots. In *The Sixth Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, 2000.

[Lafrenz *et al.*, 2002] R. Lafrenz, M. Becht, T. Buchheim, P. Burger, G. Hetzel, G. Kindermann, M. Schanz, M. Schulé, and P. Levi. Cops-team description. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-01: Robot Soccer World Cup V*, pages S. 616 – 619. Springer Verlag, 2002.

[Langley *et al.*, 1991] P. Langley, K.B. McKusick, J.A. Allen, W.F. Iba, and K. Thompson. A design for the icarus architecture. In *SIGART Bulletin 2*, pages 104–109, 1991.

[Mitchell *et al.*, 1991] T.M. Mitchell, J. Allen, Chalasani P., J. Cheng, O. Etzioni, M. Ringuette, and J.C. Schlimmer. Theo: A framework for self-improving systems. In *Architectures for Intelligence*, pages 325–355, 1991.

[OMG, 1995] Object Management Group OMG. CORBA services: Common Object Services Specificataion. OMG Document Number 95-3-31, 1995.

[Thrun, 1997] S. Thrun. The museum tourguide project: Experiences with a deployed service robot. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1997.

[Tsotsos, 1997] J.K. Tsotsos. Intelligent control for perceptually attentive agents: The s* proposal. In *Robotics and Autonomous Systems*, pages 5–21, 1997.