Autonomous AIBO watchman

TDT4735 Software Engineering, Depth Study

Ingeborg Strand Friisk

November 28, 2003

Norwegian University of Technology and Science, NTNU
Department of Computer and Information Science, IDI

Supervisor: Tor Stålhane
Co-supervisor: Siv Hilde Houmb

Abstract

As society has gradually moved towards an extensive use of computers and automatized support, both in everyday life and in work environments, the role and feasibility of autonomous robots has grown in importance. The concept of smart homes, with several computer-based systems making everyday life easier is presently an active research and development subject, i.e. at Telenor [27]. Another aspect is the inclusion of electronics and robotics for fun and entertainment in homes such as PC and TV-games and the robot entertainment dog, AIBO. AIBO walks on four legs and commercial software makes it act like a small pet, that walks around, sings songs or chooses to do nothing. AIBO owners also has the opportunity to program it to do other and maybe more useful things.

This report presents the possibility of automating watchman activities using simple and small robots such as AIBOs. We demonstrate through design analysis and implementation how AIBO can be used as a watchman to keep areas under surveillance. The main focus in this study is safety, with risk analysis and some implementation as natural parts of the study to better comprehend the concept of software safety in critical systems. A simplified version of the safety lifecycle of IEC 61508 has been applied. For a structured Preliminary Hazard Analysis HazOp has been applied. UML use cases is used to visualize and specify the functional requirements, and sequence diagrams provide details on the interaction between the different classes in the implemented solution. The main result of this project is the implementation and analysis of AIBO to walk and look around, and detects pink balls (as potential intruders). Several more requirements have been identified, but due to time constraints they have not been implemented. As not all requirements and no safety requirements have been implemented and the testing has been applied in a very informal manner. It would have been interesting to apply several more increments with implementation, testing and risk analysis, to give the implemented system more useful and accurate behavior, unfortunately the time frame has been quite limited, and these issues is therefore a subject for future work.

Preface

> **Safety** is freedom from accidents or losses.
>
> Nancy Leveson [5]

> If the safety of humans is entrusted to a computer-based system it is essential that it operates correctly. Unfortunately, the task of ensuring safe operation is an intractable one.
>
> Neil Storey [24]

Safety of software is of great importance, but difficult to achieve. This report documents the work done in the course "Software Engineering, Depth Studies", taught at the Norwegian University of Technology and Science, NTNU. This course is part of the ninth semester in the Master of Technology degree at the Department of Computer and Information Science, IDI.

This is my first well-ordered experience with the aspect of software safety, and the main purpose for this project was to give me an introduction to the concept of software safety.

In this project I have received help from many people. I'd like to thank the supervisor of the project Tor Stålhane, who was the main contributor of the concept of an AIBO watchman. My co-supervisor Siv Hilde Houmb has been incredibly enthusiastic and have given a lot of valuable help and feedback. I would also like to thank Pavel Petrovic who helped me install cygwin and OPEN-R SDK the first time, and for allowing me to borrow the Memory Stick Reader/Writer and a PCI/PCMCIA adapter. I'd like to thank my co-student Kristian Abrahamsen for helping me carry out the HazOp analysis.

November 28, 2003

Ingeborg Strand Friisk

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Safety-critical systems are growing more and more complex. To keep these systems safe, methods and techniques are necessary to prevent accidents from happening. In the past, consequences of accidents caused by dangerous new technology were limited. We no longer have the luxury of learning from experience, but must attempt to anticipate and prevent accidents before they occur. [5]

Safety is something that is not always considered for software, since software in it self can't hurt people. However, when software is operating with real-world objects, for example by monitoring and controlling aircrafts, rockets or nuclear power plants, software safety is of great importance.

There are several examples of not-so-safe software in real-world applications. Between June 1985 and January 1987, a computer-controlled radiation therapy machine, called Therac-25, massively overdosed six people. On 4 June, 1996, the Ariane 5 launcher veered off its flight path, broke up and exploded only 40 second after initiation of the flight sequence.

As the society gradually moves toward an extensive use of computers, new areas of computer-based support are growing. Mobile watchman services is one of the fields that could be automatized. This project aims at presenting an initial prototype of a mobile robot watchman.

Mobile watchman services are contraceptive. A mobile watchman's main task is to look for unwanted incidents at irregular times. The watchmen inspect the buildings and secure it. A central part of a watchman's task is to control the persons present in the building, to check that water and lights are turned off, to monitor alarm systems and to check that machinery and devices are connected according to existing routines. [14]

This project is executed in an academic context, and large real-world robots are neither accessible nor desirable to use. The AIBO, a small robot dog, seems more appropriate, due to both it's size and the availability, as it is a commercial product. The AIBO is also complex enough to constitute a challenge when it comes to safety aspects and risk analysis.

1.1 Problem description

In today's society we depend on humans performing surveillance of large and often dangerous areas. This is both time consuming and costly, and exposes humans to the potential of being physically injured by unauthorized personell, gas leak or other hazards.

By replacing humans by autonomous robots, the threat of human injury is removed and the cost of performing the surveillance is reduced. Combining the use of humans working from

a control room and robots as watchmen increases the possibility of discovering unauthorized personell and other potential hazardous events.

In this project, we will use Sony's AIBO as an autonomous watchman. The AIBO will monitor a safety area and the correct performance of its action is therefore critical. When developing AIBO watchman we will use the development process from IEC61508 and perform preliminary hazard analysis (PHA) using HazOp (Hazard and Operability analysis). We will derive safety requirements from the HazOp analysis, suggest possible safety requirements allocation and realization and explore the relationship between functional and safety (non-functional) requirements.

The watchman will be developed through a set of phases. In phase 1 the goal is to develop software that enables AIBO to autonomous walk down a corridor in an approximately straight line. If the watchman detects unauthorized personell it will make a sound. In phase 2 we will extend the watchman's abilities to move. AIBO will then also be able to detect a turn and follow the curve of the turn, and be able to avoid other types of obstacles. This phase does also include the ability of AIBO to separate authorized from unauthorized personell.

Sony AIBO will be used since this is a robot with large processing capacity (64-bit RISC processor, Clock speed 384 MHz), WLAN capabilities and several built in sensors: Temperature Sensor, IR Distance Sensor, Acceleration Sensor, Pressure Sensors (head, face, back, legs, tail) and Vibration Sensor.

IEC 61508 is used in this project since it "has been widely accepted as the basis for specification, design and operation of Safety Instrumented Systems". [15]

The selection of HazOp came naturally, since it is one of the few structured risk analysis methods that can be used without in-depth knowledge about the components to be analyzed.

1.2 Related Work

The autonomous watchdog problem strongly resembles the mobile robot problem [13] used in many classes to teach students about software architecture. Skånhaug [16] wrote his master thesis on software architecture for mobile robots. Safety is an important part of the mobile robot problem, but not the main focus. Lillevik [6] worked with how safety requirements influence software architecture in the prestudy for his master thesis.

Karine Sørby did some work on the AIBO in her master thesis [23]. Her work focused on the relationship between security and safety, and she used the AIBO as an stationary observer outside a safety-critical zone with a cutting robot. The cutting robot operates in an area closed from unauthorized personell with a gate that could be opened from the outside with a button. The AIBO watched the area with the button, and if the button was pushed when an unauthorized person was in the area, the gate would not open. The AIBO role in her work can be seen as a stationary and communicating version of the robot watchman in this project.

The four legged league in RoboCup [12] is the biggest event for AIBO programmers, and a lot of universities and other organizations participate every year. In RoboCup two teams with 3 AIBO's on each team play soccer. The programmers of the participating teams do not usually focus on safety "because the robot should run aggressively as much as possible to complete the soccer game" [1]. Figure 1.2 contains a picture from a RoboCup game.

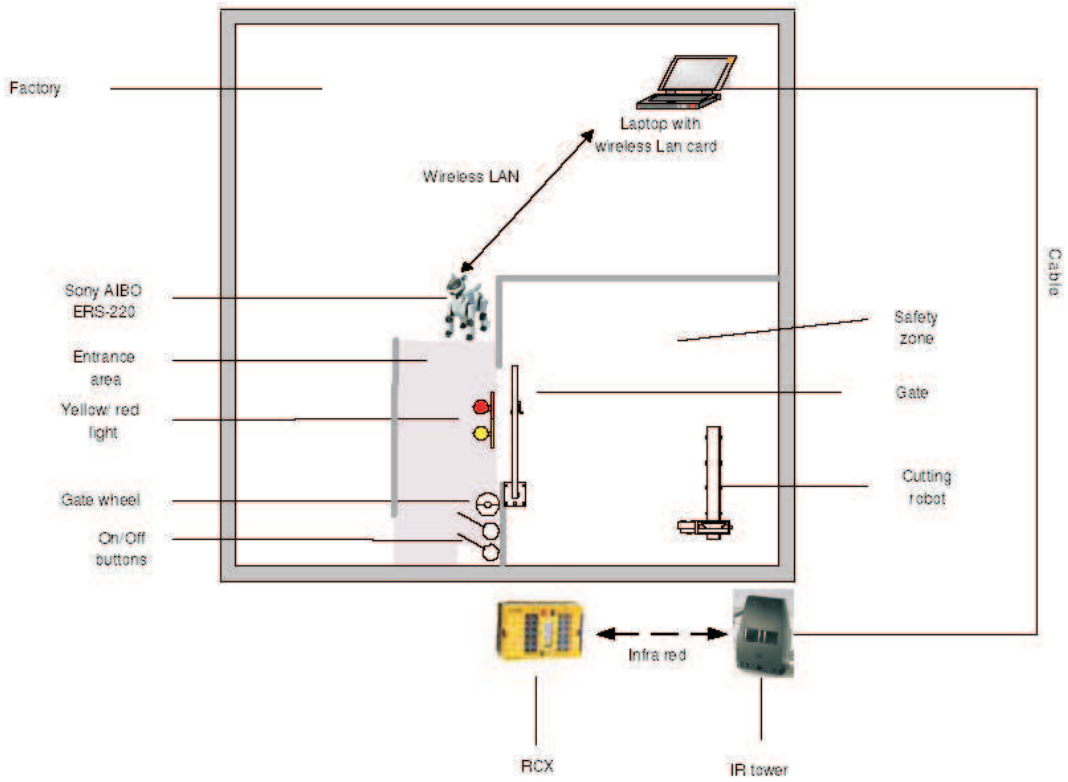

Figure 1.1: The cutting robot [23]

Figure 1.2: Picture from RoboCup

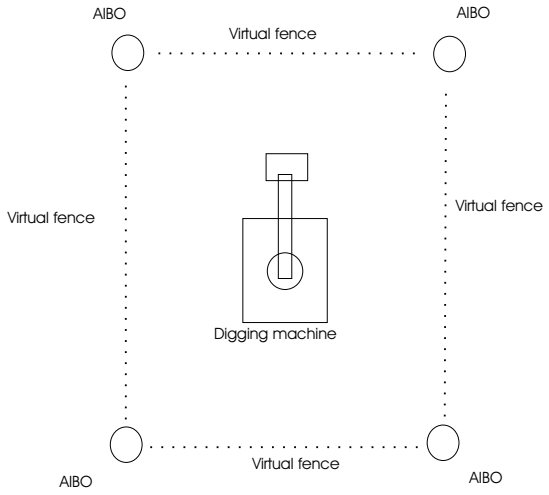

Figure 1.3: The digging machine at IDI

The Safety group at IDI wish to create a system with a safe mobile digging machine. The digging machine will be modeled with LEGO Mindstorms, with a virtual mobile fence surrounding it. The mobile fence consist of four AIBOs surrounding the digging machine and surveilling the boundaries of the area the digging machine is operating in. When the digging machine moves, the AIBO fence moves with it. If one of the AIBOs detect motion in the boundary it is watching it will send a message to the digging machine and the other AIBOs and the digging machine will stop operating. The system can be restarted with a mobile telephone. Figure 1.3 contains a picture of the digging machine environment. Hopefully, some of the results of this project can be used in the digging machine project.

1.3 Report structure

This section describes the structure of the report. The report consists of 10 chapters. The reader should read the chapters in chronological order, as some of the chapters has background material from previous chapters.

Chapter 1, Introduction: provides general information about the project, background and motivations.

Chapter 2, Safety and Risk Management: contains a brief introduction to the development of safety-critical systems and risk analysis methods.

Chapter 3, Sony AIBO: presents the technical equipment used in this project.

Chapter 4, Concept: gives a general introduction to the system developed during this project.

Chapter 5, Preliminary Hazard Analysis: contains the hazard analysis carried out in this project.

Chapter 6, Requirement Specification: describes the prototypes functional and non-functional requirements.

Chapter 7, Test Plan: presents the test plan for the prototype.

Chapter 8, Design and Implementation: describes the prototype's functionality, sequence diagrams and implementation status.

Chapter 9, Testing and safety validation: explains why the prototype have not been tested formally.

Chapter 10, Conclusion, Further work and Discussion: evaluates and summarizes the main results of this projects and give directions for further work

Chapter 2

Safety and Risk Management

Nancy Leveson [5] defines *safety* as "The freedom from accidents or losses". She defines *accident* as an undesired or unplanned event that results in a specified level of loss. Loss implies some type of damage to life, property or environment. Safety is an important quality. To achieve safety, careful planning is necessary, hazards must be predicted and steps must be taken against them. Alternatively, if the hazards are have low likelihood and negligible consequences, it is possible to accept them.

This chapter presents the suggested life cycle for safety-critical systems from IEC 61508. IEC 61508 "has been widely accepted as the basis for specification, design and operation of Safety Instrumented Systems" [15]. The process in IEC61508 relies on thorough risk analysis, and therefore several risk management methods will be discussed. The selected risk analyzes methods are Hazard and Operability Analysis(HazOp), Fault Tree Analysis (FTA) and FMEA/FMECA (Failure Modes, Effect(and Criticality) Analysis). HazOp uses guidewords and applies them to each subsystem to see what deviations from normal operation the combinations of guidewords and subsystems could cause. FTA can be used to visualize how the deviations found by HazOp arise. FMECA can be used to decide the criticality of the leaf nodes in FTA, and applying the results from FMECA into the FTA can help the developers of the system to decide the proper effort to reduce the likelihood of risks effectively.

2.1 IEC 61508

The purpose of IEC 61508 is to provide a framework for the lifecycle of safety-critical systems with electrical, electronical or programmable components. The standard covers all phases in the lifecycle of safety-critical systems, from initial planning and requirement specification through operation and to disposal or modification of the system. Figure 2.1 shows the IEC 61508 lifecycle model. The target of the phases are described in Table 2.1

2.2 Hazard and Operability Analysis (HazOp)

HazOp was developed by Imperial Chemical Industries in England in the early 1960s and later improved upon by the Chemical Industries Association in London [5]. It can also be applied to other domains. The purpose of HazOp is to identify potential hazards and operability problems caused by deviation from design intent of the system.

HazOp is usually conducted by a team of from four to eight engineers, including experts in the application area as well as those directly concerned with the design of the system [24]. The team starts from the basic specified operation of the system and investigates the effects of deviations

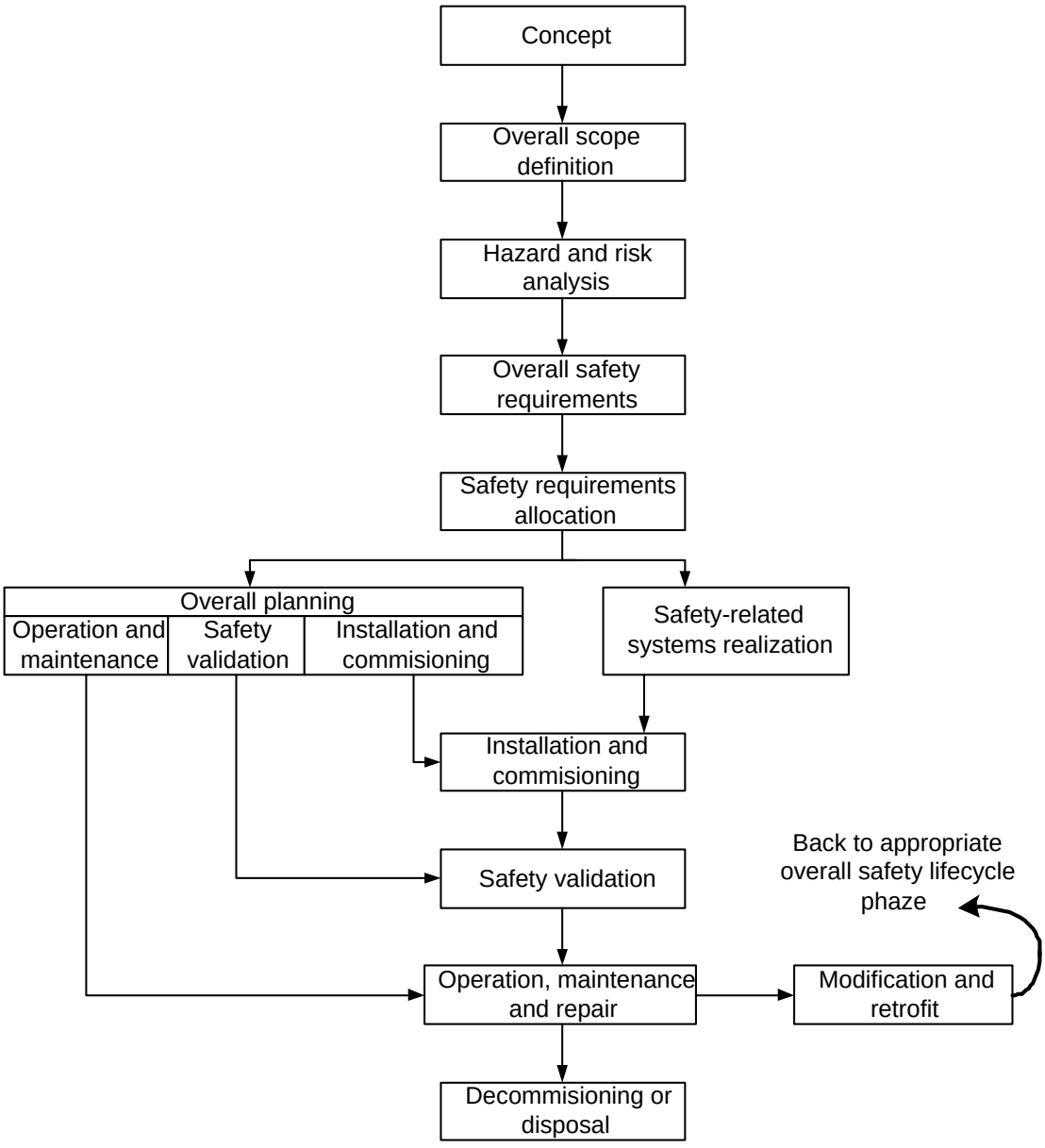

Figure 2.1: IEC61508 Lifecycle, extracted from[4]

Phase	Target
Concept	To develop a level of understanding of the system's functionality and it's environment.
Overall scope definition	To determine the boundary of the system and to specify a scope for the hazard and risk analysis.
Hazard and risk analysis	To determine the hazards and hazardous events of the system, for all reasonably foreseeable circumstances including fault conditions and misuse, and to determine the event sequences leading to the hazardous events found.
Overall safety requirements	To develop the specification for the overall safety requirements, in terms of the safety functions requirements and safety integrity requirements, for the safety-related system, other technology safety-related systems and external risk reduction facilities, in order to achieve the required functional safety.
Safety requirements allocation	To allocate the safety functions, contained in the specification for the overall safety requirements, to the designated safety-related system and external risk reduction facilities, and to allocate a safety integrity level to each safety function.
Overall operation and maintenance planning	To develop a plan for operating and maintaining the safety-related system and to ensure that the required functional safety is maintained during operation and maintenance.
Overall safety validation planning	To develop a plan to facilitate the overall safety validation of the safety-related systems.
Overall installation and commissioning planning	To develop a plan for installing and commissioning the safety-related systems in a controlled manner so the required functional safety is achieved.
Safety-related systems realization	To create safety related systems conforming to the specification for the safety requirements.
Overall installation and commissioning	To install and commission the safety-related system.
Overall safety validation	To validate that the safety-related system meet the specification for the overall safety functions requirements in terms of the overall safety integrity requirements, taking into account the safety requirements allocation for the safetyrelated systems.

Table 2.1: Lifecycle overview, extracted from [4]

from this normal operation. For each deviation the team sets out to answer a series of questions to decide whether the deviation could occur, and if so, whether it could result in a hazard. Safety features designed to control hazards are also considered [24].

Guide words are applied to any variable of interest such as flow, temperature, pressure, level of composition and time. Table 2.2 presents the guidewords traditionally used in HazOp. These are the guide words applied in this project.

Guide word	**Meaning**
NO, NOT, NONE	The intended result is not achieved, but nothing else happens (such as no forward flow when there should be).
MORE	More of any relevant physical property than there should be (such as higher pressure, higher temperature, higher flow, or higher viscosity).
LESS	Less of a relevant physical property than there should be.
AS WELL AS	An activity occurs in addition of to what was intended, or more components are present in the system than there should be (such as extra vapors, or solids or impurities, including air, water, acids, corrosive products).
PART OF	Only some of the design intentions are achieved (such as only one of two components in a mixture).
REVERSE	The logical opposite of what was intended occurs (such as back flow instead of forward flow).
OTHER THAN	No part of the intended result is achieved, and something completely different happens (such as the flow of the wrong material).

Table 2.2: Guide words for HazOp [5]

HazOp was originally designed for chemical plants. For other system, for example computer systems, it is necessary to cover the time aspect as well. The Ministry of Defence Standard 00-55 [9] suggests the additional guidewords in Table 2.3 for computer systems.

Guide word	**Meaning**
EARLY	Something happens earlier than expected relative to clock time.
LATE	Something happens later than expected relative to clock time.
BEFORE	Something happens before it is expected, relating to order or sequence.
AFTER	Something happens after it is expected, relating to order or sequence.

Table 2.3: Additional guide words for systems containing programmable electronics [9]

2.3 Fault Tree Analysis (FTA)

FTA is primarily a means for analyzing causes of hazards, not identifying hazards. The top event in the tree have been identified first by other techniques. FTA uses Boolean logic to describe the combinations of individual events, often faults, that can constitute a hazardous event. Each level in the tree lists the more basic events necessary and sufficient to cause the problem on the level above. The intermediate events (events between the top event and the leaf nodes in the tree) are pseudo events (abstractions of real events) - they are combinations or sets of the basic, or primary, events and are usually removed during the formal analysis of the tree. The relationships between events are specified by standard logical relations. The tree

visualizes the relationships between the events, and thus makes the system and its relations more understandable. A thorough understanding and definition of the system and its interrelations is essential when using the FTA process. Once the tree is constituted, it can be written as a Boolean expression and simplified to show the specific combinations of identified basic events sufficient to cause the undesired top event.

Leveson [5] identifies four basic steps in the Fault Tree Analysis:

1. system definition
2. fault tree construction
3. qualitative analysis
4. quantitative analysis

In the system definition part, the top events of the fault tree, initial conditions, existing and impermissible events are determined. The analysts may use system functional diagrams, flow diagrams, logic diagrams or other design representations, or may rely on their knowledge of the system. For each component that has more than one possible state, the analysts must decide upon the system state to be analyzed for the occurrence of the top event in the fault tree.

In the fault tree construction part, the analysts assumes a particular system state and a top event, and then write down the casual events related to the top event and their relations, using logical symbols to describe the relations. The process of identifying events is repeated until all basic events are found. Figure 2.2 shows the different fault tree symbols.

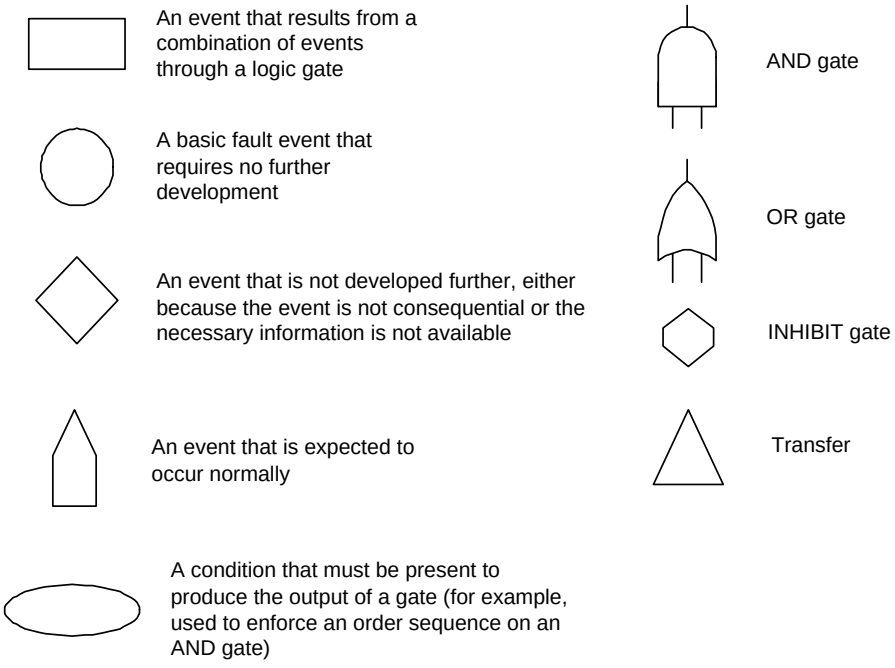

Figure 2.2: Fault tree symbols [5]

The purpose of the qualitative analysis is to reduce the tree to a logically equivalent form, constituting of only top event and basic events. The reduced trees are called *cut sets*. The goal of the qualitative analysis is to find the *minimal cut sets*, which represent the basic events that can cause the top event and which cannot be reduced in number. Cut sets are defined such that if a basic event in the cut set does not occur, the top event will not take place. Minimal cut sets provide information that helps identify weaknesses in the system. They can determine the importance of each event with respect to the top event.[5]

Quantitative analysis use the minimal cut sets to calculate probability of the occurrence of the top event from the probability of occurrence of the basic events. The probability of the top event will be the sum of the probabilities of all the minimal cut sets for the top event. The probabilities of the cut sets are determined by multiplying the probability of the basic events in each cut set [5].

When FTA is applied to software systems, the probabilistic analysis is not applicable, assigning a probability to a software statement is difficult. If design errors are found in the tree through this process, they should be fixed rather than given a probability. [5]

2.4 FMEA & FMECA

In this section the risk analysis methods of FMEA and FMECA are described briefly. FMEA is short for Failure Modes and Effects Analysis, while FMECA is short for Failure Modes, Effects and Criticality Analysis. It is natural to mention the two methods at the same time, as FMECA is an extension of FMEA. In this project, FMECA could be used to find the criticality of the basic events in FTA, or the incidents discovered in HazOp.

2.4.1 FMEA

FMEA is a reliability analysis method that emphasizes successful functioning rather than hazards and risks. The goal is to establish the overall probability that the product will operate without a failure or that the product will operate with a certain length of time between failures. The steps in an FMEA is listed below:

1. Identify all components and their failure modes, considering all possible operating modes.
2. For each component failure mode, identify the impact on the other components and on the overall system.
3. Calculate the seriousness and probability of each failure mode.

In FMEA, the significant failure modes must be known in advance. These failure modes can be incidents identified by HazOp or basic events in FTA. The technique itself does not provide any systematic approach for identifying failure modes or for determining their effects. It does not consider effects of multiple failures; each failure is treated as an independent occurrence with no relation to other failures in the system except for the subsequent effects it might produce.

2.4.2 FMECA

FMECA is the same as FMEA, extended by an analysis of the criticality of the failure. Two steps are added to process of the FMEA:

4. Document means to reduce or control the failure rate and/or effects
5. Modify the columns for effects/failure rate when the control methods are applied to the unit.

The results from FMECA is documented in a table as shown in Table 2.4.

2.5 Relationship between different risk analysis methods

The risk analysis methods has different purposes. HazOp is developed to find deviations from normal operation and is designed to find errors previously unthought of. FTA gives a structure to analyze the relationship between the causes and the faulty behavior of the system, but will not help analyzers discover new faults. FMECA helps finding the criticality and failure probability, and can be applied to the basic events from FTA, or the incidents from HazOp to identify the most effective means to reduce the hazard likelihood or criticality. Table 2.5 describes how the risk analysis methods are related.

Item	Failure Mode	Cause of failure	Possible Effects	Prob.	Level	Possible Actions to Reduce Failure Rate or Effects
Motor Case	Rupture	a. Poor workmanship b. Defective materials c. Damage during transportation d. Damage during handeling e. Overpressurization	Destruction of missile	0.0006	Critical	Close control of manufacturing process to ensure that workmanship meets prescribed standards. Rigid quality control of basic materials to eliminate defectives inspection and pressure testing of completed cases. Provision of suitable packaging to protect motor during transportation.

Table 2.4: A sample FMECA [5]

2.6 Risk estimation and risk evaluation

When hazards are identified, risk is estimated as a combination of the frequency or probability of a specified hazardous event and the consequences when it occurs. When classifying the risks it is necessary to define or adopt a scale for consequence and frequency. Table 2.6 provides the MoD 00-56 [8] consequence classes , while Table 2.7 provides the MoD 00-56 [8] likelihood ranges.

When consequence and likelihood values of the hazard are estimated, the two values are combined into an estimate of the level of risk. Table 2.8 contains definitions of risk classes, and Table 2.9 shows how the combinations of likelihood and consequence are classified. When dealing with the classified risks, class A risks has highest priority, and class D risks has lowest priority.

$\downarrow From \setminus To \rightarrow$	HazOp	FTA	FMECA
HazOp	HazOp identifies incidents at different levels of abstraction.	The incidents identified by HazOp are inserted in fault trees based on abstraction level and the relationship between the incidents.	Incidents identified by HazOp may be understood as failure modes and thereby can be considered as starting points for FMECA.
FTA	A basic event (a leaf node in th fault tree representing an incident) may correspond to a subsystem/service on which HazOp may be applied.	A fault tree may be a part of another fault tree, i.e., the top incident of one fault tree may be a causing incident in another fault tree.	Basic events (leaf nodes in the fault tree representing incidents) may be understood as failure modes and thereby can be considered starting points for FMECA.
FMECA	From a basic incident (failure mode) one can associate a subsystem/service for applying HazOp on.	The analysis of a basic incident (failure mode) may identify a scenario leading to an unwanted incident. This may be represented as a path in the fault tree.	Basic incidents (or failure modes) may lead to incidents that are basic incidents (failure modes) in another FMECA.

Table 2.5: The relationship between different risk analysis methods [3]

Category	**Definition**
Catastrophic	Multiple deaths
Critical	A single death, and/or multiple severe injuries or severe occupational illnesses
Marginal	A single severe injury or occupational illness and/or multiple minor injuries or minor occupational illnesses
Negligible	At most a single minor injury or minor occupational illness.

Table 2.6: Accident severity categories for military systems [8]

Likelihood	**Occurrence during operational life considering all instances of the system**
Frequent	Likely to be continually expected
Probable	Likely to occur often
Occasional	Likely to occur several times
Remote	Likely to occur some time
Improbable	Unlikely, but may exceptionally occur
Incredible	Extremely unlikely that the event will occur at all, given the assumptions recorded about the domain and the system

Table 2.7: Likelihood ranges [8]

Risk class	**Interpretation**
Class A	Intolerable risk
Class B	Undesirable risk, and tolerable only if risk reduction is impracticable or if the costs are grossly disproportionate to the improvement gained
Class C	Tolerable risk if the cost of risk reduction would exceed the improvement gained
Class D	Negligible risk

Table 2.8: Risk Class Definitions [8]

	Catastrophic	**Critical**	**Marginal**	**Negligible**
Frequent	A	A	A	B
Probable	A	A	B	C
Occasional	A	B	C	C
Remote	B	C	C	D
Improbable	C	C	D	D
Incredible	C	D	D	D

Table 2.9: Risk classification by likelihood and consequence [8]

Chapter 3

Sony AIBO

In this chapter Sony's entertainment robot AIBO is described. First, there is a section about AIBO, and it's technical specification. The commercial software from Sony and the programming environment will be described, and finally there is a section about AIBO WLAN configurations.

3.1 Introduction

AIBO is the nickname of the Sony entertainment robot. The name "AIBO" was coined from the words "A. I." (Artificial Intelligence), "eye", and "robot". In Japanese, the word "aibou" means "partner" or "pal". [20] Figure 3.1 provides a picture of AIBO ERS-220A, the AIBO model used in this project. A technical drawing is provided in Figure3.2, while the technical specification is presented in Table 3.1.

Figure 3.1: Sony ERS-220A Entertainment Robot AIBO [21]

The software of AIBO resides on a removable Memory Stick. There are several different programs for AIBO by Sony, all designed to utilize AIBO as a pal and companion. However, Sony does also provide the possibility to develop other programs for AIBO. For this, Sony recommend using a package called OPEN-R or AIBO Master Studio. The two options are quite different, Master Studio comes with GUI functionality and the Behaviors created can be added into the commercial AIBO software products. The drawback is that it isn't transparent, the programmer doesn't have total control over the AIBO. OPEN-R is quite the opposite. It doesn't have a graphical user interface, and can't be integrated with commercial AIBO software products. The good thing about OPEN-R is that the programmer is in total control of the AIBO, and it's also free.

CPU	64-bit RISC processor. Clock speed 384 MHz
Components	Body, Head, Leg x 4, Tail (removable)
Main Storage	32MB SDRAM
Program Storage Medium	Memory Stick Media for AIBO
Movable Parts	Head x 3 Retractable Headlight x 1 Legs x 3 x 4 legs
Input/Output	PC Card Slot Type 2 In/Out Memory Stick Media Slot In/Out AC in Power Supply Connector Input
Image Input	CMOS Image Sensor (100K pixel)
Audio Input	Miniature Stereo Microphone
Audio Output	Miniature Stereo Microphone
Built-in	Sensors Temp. Sensor IR Distance Sensor Acceleration Sensor Pressure Sensors (head, face, back, legs & tail) Vibration Sensor
Power Consumption	approx. 9W(standard operation in autonomous mode)
Operating Time	approx. 1.5 hrs. (standard operation in autonomous mode)
Battery Charging Time	approx. 2 hours
LCD Display	Time, Date, Volume, Battery Condition
Operation Temp.	5-35 degrees Celsius (41-95 F.)
Operation Humidity	10-80%
AIBO Entertainment Robot (ERS-220A) includes	AC adapter Lithium-ion Battery Transfer Plug Pink Ball Release Pin
Dimension	152x296x278mm (w x h x l)
Mass	approx. 1.5kg (incl. battery and MS media)

Table 3.1: AIBO ERS-220A Technical specification[21]

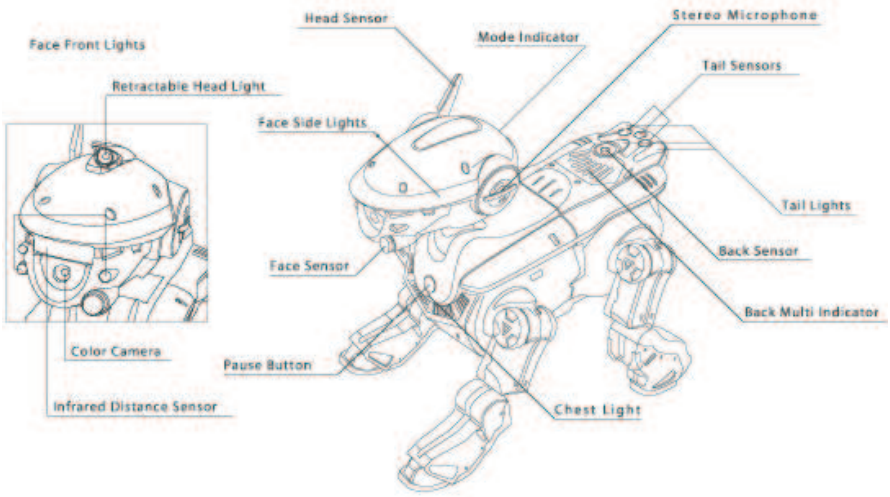

Figure 3.2: AIBO "ERS-220A" Exterior [21]

3.2 AIBO Software from Sony

The initial purpose of AIBO is different from traditional robots. As the name indicates, AIBO is designed to be a pal and companion. Whenever it is using the programs packages from Sony, AIBO does not do useful work, it doesn't make coffee or schedule meetings. AIBO is an entertainment robot able to walk on four legs, like a small pet. Even more, AIBO is continually learning from new experiences. AIBO can see and respond to being spoken to, stroked or entertained and will shape its behavior around its mood and instincts. Over time, AIBO's personality will change depending on the relationship it has with you and its environment [21]. An article in a newsletter from the New Jersey Association of Non-Profit Homes for the Aging [2] describes how some seniors got emotionally attached to AIBOs during a six week period.

3.3 Programming AIBO

In this section the programming environment for AIBO is described. In this project OPEN-R will be used, but an alternative tool is the AIBO Master Studio. OPEN-R and Tekkotsu is briefly described below.

3.3.1 OPEN-R & OPEN-R SDK

"OPEN-R" is the standard interface for the entertainment robot system that Sony is actively promoting. This interface greatly expands the capabilities of entertainment robots. The OPEN-R SDK is the cross development environment based on gcc(C++) where you can make software that works on the AIBO models ERS-210, ERS-220, ERS-210A, and ERS-220A. [11]

OPEN-R application software consists of several OPEN-R objects. An object is a concept that only exists at run-time. Each object has a counterpart in the form of an executable file, created at compile-time. Source code is compiled and linked to create this executable file before the file is copied to an AIBO Programming Memory Stick. When AIBO boots, the system software loads the file from the AIBO Programming Memory Stick and executes it as an object. Each object has it's own thread and runs concurrently with other objects in the system. An object can send messages to other objects. Objects are single-threaded, which means that an object

can process only one message at a time. An object is in an infinite loop waiting for messages. It cannot terminate itself. (extracted from [19]) A detailed description of the robot's external interface and their connection to variable names used in OPEN-R SDK is given in [18].

3.3.2 Tekkotsu

Tekkotsu is an open source project created and maintained at Carnegie Mellon University. Tekkotsu is an application framework for robotic platforms, developed for the Sony entertainment robot AIBO. The word "Tekkotsu" is Japanese, and means "iron bones", often used in the context of buildings' structural framework. It builds on the basic functionality in the OPEN-R framework, and aims to give programmers a structure on which to program on. As OPEN-R, Tekkotsu is based on C++.

Tekkotsu is an object oriented and event passing architecture, similar in design to the Java programming language. New AIBO Behavior can be added in a menu system and be turned on and off at will. Classes subscribes to an event router and is notified when an event they are subscribed to occurs. The framework is structured to prevent the amount of spaghetti code that can occur using only OPEN-R.

The framework separates user programmed classes in two main categories. *MotionCommands* deal with simple motion primitives, such as lighting the LEDs and setting the joints, while *Behaviors* act more like high level application and with processing. To put it in a very simple way, the Behaviors combine sensor inputs and MotionCommands to make "intelligent" actions.

Java and Matlab Tekkotsu tools

The Tekkotsu framework comes with a menu system, called TekkotsuMon. TekkotsuMon is the collection of tools which provide control and visualization/monitoring of the AIBO's run time state. There are three levels to the interface: local/physical, console, and GUI. [21]

At physical level, the menu can be accessed through the buttons on the AIBO, and it gives feedback through the different LEDs. This option is the only option when the user don't have access to a computer with a WLAN card.

It is possible to connect to the AIBO with telnet over WLAN. Different ports gives different outputs, it is possible to connect to the AIBO on port 59000, 10001 and 10002. Port 59000 is the standard port used by the operating system Aperios, through this port the menu system is accessible, and comments in the code using `cout` will be sent here as well. `Cout` is very useful when programming/debugging. Port 1001 and 1002 is used for framework messages. The output to these ports is described at the Tekkotsu website [26], under Tutorials → TekkotsuMon Tutorial.

The GUI tools give access the menu system from a GUI Window. There are both Java and Matlab applications, the Java application gives easier access to the menu system and allows remote controlling of the AIBO, both walking and head motions. The Matlab tools can give access with a map that is created with distance information from the IR-unit. The GUI tools have very much functionality, for more detailed information the reader is referred to [26].

Tekkotsu architecture

Figure 3.3 visualizes the Tekkotsu architecture. It shows the basic interaction between the processes and shared memory regions. The Tekkotsu framework runs three system processes on the AIBO, described in Table 3.2.

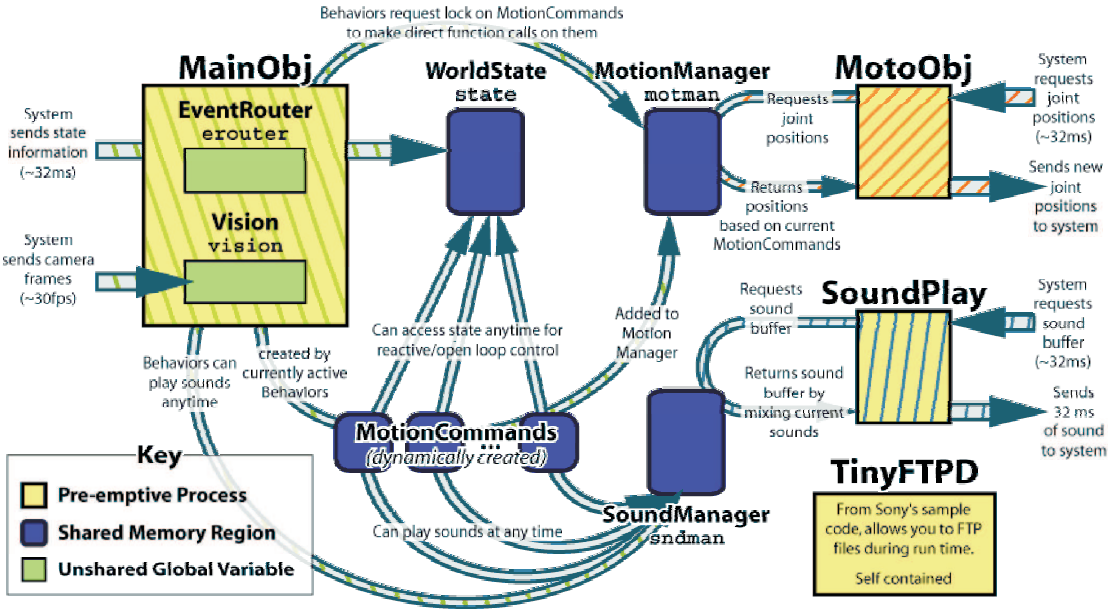

Figure 3.3: Overwiev of the Tekkotsu architecture [26]

Process	Function
MainObj	Most of the bulk processing, such as vision, decision making and tracing state of the world.
MotoObj	Updating positions of joints and values of LEDs.
SoundPlay	Mixes and sends sound data to the system to ensure playback doesn't shutter.

Table 3.2: Tekkotsu processes

Since there are only three system processes, the Behaviors and other code will be close to cooperative multitasking,

no two parts of Main can execute at the same time, and only one event is processed at a time. At regular intervals, the operating system sends messages to the Main process containing sensor readings and image frames. The operating system also sends power status messages to Main, but at a much lower frequency, and not that regularly. Whenever Main receives a message, it processes the new information and sends events as needed. It also takes the opportunity to check if any timers have expired and need to send timer events.

Motion also receives regular messages from the system, but only when the system has finished executing a buffer of joint positions and is ready for more, to which Motion responds by filling a new buffer and sending it to the system. The motion process uses double buffering to ensure continuous motion.

3.4 Communication with the AIBO

Using a wireless card, AIBO can communicate with a PC. This PC also needs to be equipped with a wireless LAN card, unless it is connected to a wired LAN network with a wireless LAN access point. The different wireless configurations is shown in Figure 3.4

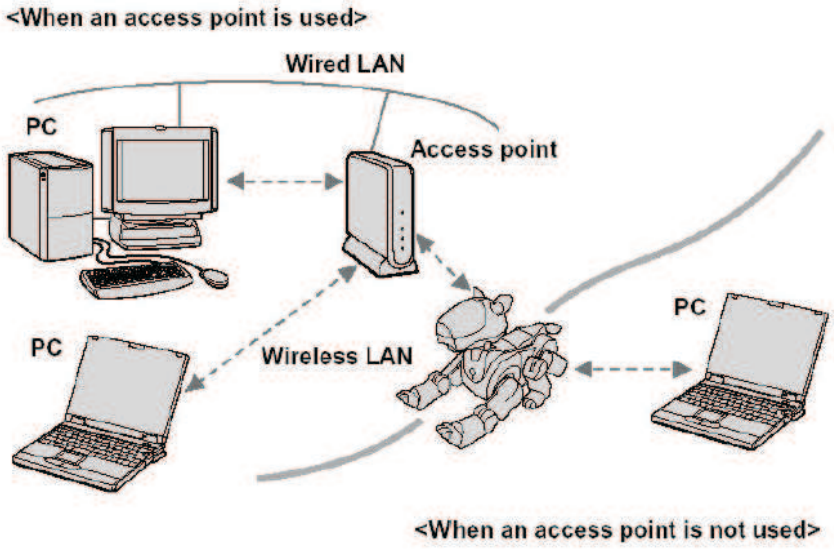

Figure 3.4: Wireless LAN network configurations

There are four different wireless LAN configurations:

- There is an access point and communication is conducted through the access point from a wireless LAN-enabled PC.
- There is an access point and communication is conducted through the access point from a PC connected to the wired LAN network and devices within the wireless LAN network.
- An access point is not used, all personal computers are equipped with wireless LAN cards, and at least one of the PCs is set to IBSS Peer-to-Peer mode.
- An access point is not used, all personal computers are equipped with wireless LAN cards, and none of the PCs is set to IBSS Peer-to-Peer mode.

For a more indepht description of the different wireless configurations, see [22].

Chapter 4

Concept

This project is carried out using the lifecycle model given in IEC 61508. As described as the first two phases in the IEC 61508, this chapter presents an overall idea of the system, it's goals and intended environment.

4.1 Overall idea

Very often areas have to be monitored, to make sure no unauthorized personell can get access to specific resources or injure themselves on production equipment. These areas are usually monitored with a combination of surveillance cameras and patrolling watchmen. The costs of 24-hour surveilling are quite large, especially for larger areas when it is necessary to have several watchmen working around the clock in order to keep the areas monitored in a satisfactory manner.

These costs can be greatly reduced by introducing autonomous robots patrolling the area and taking action whenever unauthorized personell is discovered. This system will represent phase 1 of a robot watchman that monitors a corridor. Figure 4.1 visualizes the system and system environment.

The robot watchman will walk down the corridor without walking into walls and damaging itself or getting stuck. Once the watchman detects an intruder, it will make a sound to indicate that an intruder has been discovered, and will take action against it. An operator will be controlling the robot watchman, and has opportunity to it start up and to shut it down.

This will unfortunately also make unauthorized personell able to manipulate the behavior of AIBO. Still, to manipulate AIBO an unauthorized person must know how to do it.

Unauthorized personell can also choose to damage AIBO physically. But the costs of AIBO repair is far smaller than the cost of human injury.

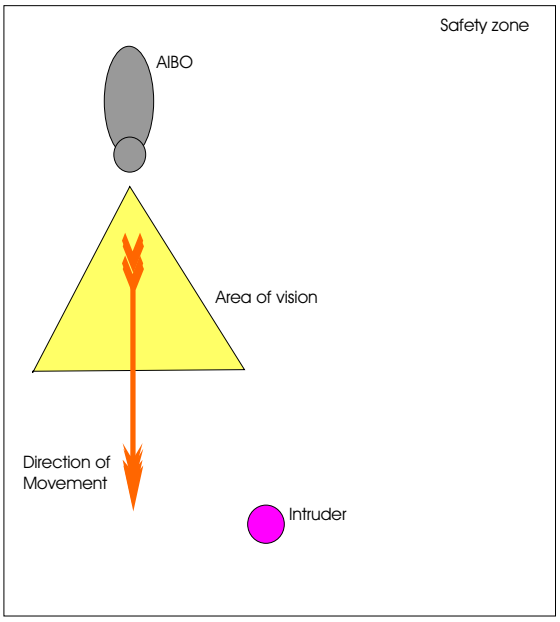

Figure 4.1: System environment

Chapter 5

Preliminary Hazard Analysis

To have a clear idea of the hazards and risks of the system, and include risk reduction in the development cycle on an early stage, a preliminary hazard analysis is necessary. This is the third step in the lifecycle defined in IEC 61508.

5.1 HazOp analysis

To carry out an HazOp, study nodes and guide words have to be chosen. The first two parts in this section regard the selection of study nodes and guide words. The final part of this section contains the result of the HazOp analysis.

5.1.1 Selection of nodes

To find all important hazards in a system, the subsystems to be studied must be carefully described. Only hazards in the selected subsystems will be found, and it is therefore vital to include all critical parts of the system in HazOp. Table 5.1 contains the selected subsystems in the AIBO watchman system. The rest of the system is out of scope of phase 1 of the development.

Subsystem	**Description**
Battery	The battery in the robot.
Joints	The different joints of the robot. The different joints are regarded as one study node, since the reaction to joint problems will be the same for all joints.
Camera	The robot uses a camera to get information about it's surroundings.
IR-unit	The robot uses an infra-red unit to measure distances to objects.

Table 5.1: Selected subsystems for HazOp

5.1.2 Selected guidewords

In an HazOp study, guidewords is an important part of the analysis. Before starting an analysis, guidewords that will be applied to the study nodes must be selected. Generally, there are two strategies for selecting the guidewords.

- Choose to use a list of predefined guidewords.
- Select the guidewords that the analysis group find appropriate.

In this analysis the guide words form Table 2.2 have been applied to the different study nodes. Not all guidewords were applicable in combination with the subsystems, these combinations are marked as N/A.

5.1.3 Results

In this section the results of the conducted HazOp analysis is documented in tables.

Subsystem: Battery						
Guideword	**Deviation**	**Consequence**	**Source**	**Solution**	**Category**	**Likelihood**
No	No power	Robot can not operate	Battery runs out of power. Battery is defect.	Backup battery. If the robot is operating, it should shut down in a controlled manner.	Critical	Frequent
More	N/A					
Less	N/A					
As well as	N/A					
Part of	N/A					
Reverse	N/A					
Other than	N/A					

Table 5.2: HazOp table for the battery subsystem

Subsystem: Joints						
Guideword	**Deviation**	**Consequence**	**Source**	**Solution**	**Category**	**Likelihood**
No	No movement	Joints lock in position. Robot can not complete it's task	Dirt get caught in joints. The robot can be weared out, or be damaged. The robot can be stuck on a wall.	Regular cleaning. Service controls of the robot. When joint-lock is detected, the joint motors should stop, and the joint should be loose.	Critical	Remote
More	Jerky movement of joints	The robot will wear out faster. Larger accident probability.	Programming error.	Set a maximum angular velocity. Power will slowly fade in to the joints at startup.	Marginal	Probable
Less	Movement in joints too slow	The robot moves to slowly to accomplish it's task in acceptable time.	Dirt in joints. The robot can be weared out or damaged.	Regular cleaning. Service controls of the robot.	Critical	Remote
As well as	N/A					
Part of	The joint can not move in all directions	The area the robot can cover will be limited. The robot may be unable to fulfill its tasks.	Dirt in joints. The robot can be weared out or damaged. Programming error.	Regular cleaning. Service controls of the robot. Thorough testing before release.	Critical	Improbable
Reverse	N/A					
Other than	N/A					

Table 5.3: HazOp table for the joints subsystem

Subsystem: Camera						
Guideword	**Deviation**	**Consequence**	**Source**	**Solution**	**Category**	**Likelihood**
No	No picture	Robot will not detect intruders	Camera is damaged. The camera lens is dirty.	Regular controls of robot. Robot notifies controller when there is no picture to minimalize mean time to recover.	Critical	Improbable
More	N/A					
Less	N/A					
As well as	N/A					
Part of	Low quality picture	Robot can fail to detect intruders	Camera is damaged. The camera lens is dirty.	Regular controls of the robot.	Critical	Remote
Reverse	N/A					
Other than	N/A					

Table 5.4: HazOp table for the camera subsystem

Subsystem: IR-unit						
Guideword	**Deviation**	**Consequence**	**Source**	**Solution**	**Category**	**Likelihood**
No	No distance measurements	Robot can not navigate	Defect component	The robot should notify the operator that there is no distance information. Repair the robot.	Critical	Improbable
Less	The robot will think objects to be closer than they are.	Navigation problems.	Defect component. Dust or other disturbances.	Regular maintenance. Repair.	Negligible	Improbable
More	The robot will think objects to be further away than they are.	Navigation problems. The robot can accidentally bump into walls and damage itself.	Defect component.	Regular maintenance. Repair.	Critical	Improbable
As well as	N/A					
Part of	The robot does not detect all objects or measure all distances.	Navigation problems.	Defect component. Dirt on measuring device.	Regular maintenance. Repair.	Critical	Improbable
Reverse	N/A					
Other than	N/A					

Table 5.5: HazOp table for the IR-unit subsystem

Chapter 6

Requirement specification

This chapter presents the functional and non-functional requirements of the AIBO watchman. This is the fourth phase in the lifecycle from IEC 61508. Functional requirements describe the functionality or services the system is expected to provide. [17] Non-functional requirements, as the name suggests, are those requirements which are not directly concerned with the specific functions delivered by the system. [17] In this chapter we first describe the functional requirements and then the safety requirements.

6.1 Functional requirements

This section describes the functional requirement of the robot watchman. The functional requirements express the tasks the robot must be able to perform. First, use case diagrams are included to visualize the functional requirements and the connections between them, and the requirements are presented in a numbered list. The end of this section contains more in-depth descriptions of the requirements.

The functional requirements are stated in the use cases in Figure 6.1 and Figure. The use case diagram represents a visual presentation of the functional requirements, and is often useful as it presents the requirements in a more understandable way for the stakeholders.

The use case in Figure 6.1 contains two actors, the AIBO watchman and an operator. The watchman is the robot that monitors the safety-critical zone, while the operator controls the robot. The only tasks of the operator are to start and stop the robot, and to turn on or shut down. Figure 6.2 visualizes the requirements that the AIBO should be able to perform on it's own, without operator guidance.

A numbered list of the functional requirements is given below:

R.1 Boot robot

R.2 Shut down robot

R.3 Start robot

R.4 Stop robot

R.5 Move straight forward

R.6 Walk forward

R.7 Recognize wall

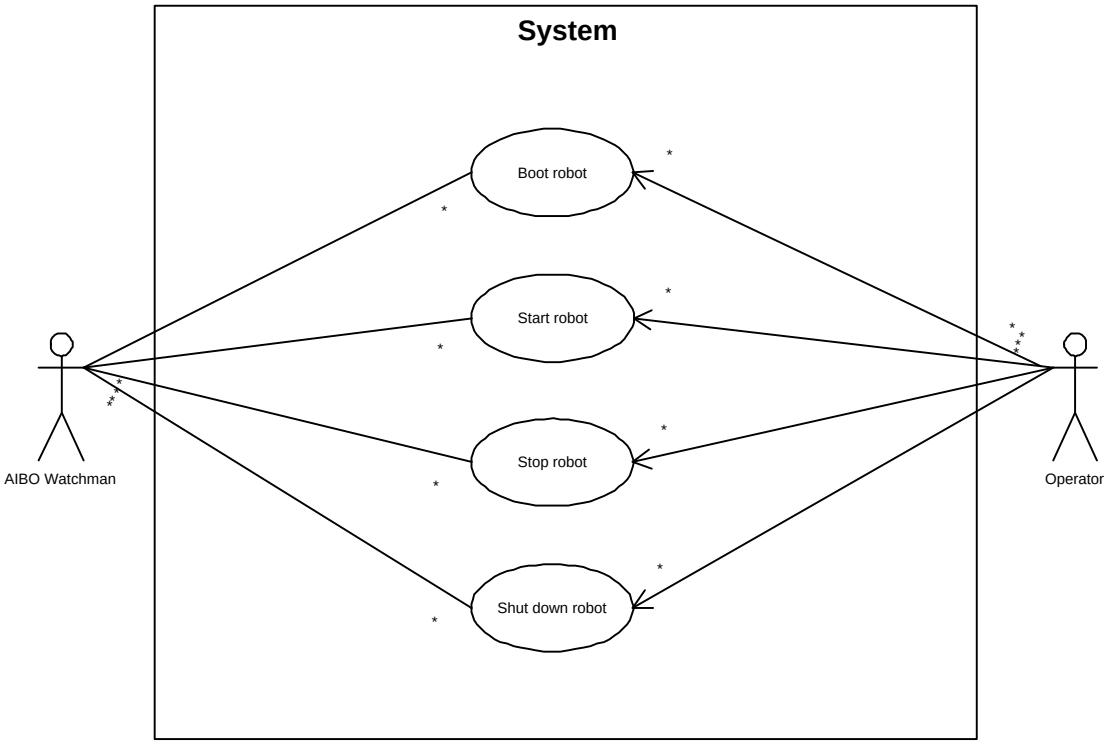

Figure 6.1: Use case diagram of robot with operator

R.8 Discover intruder

R.9 Make sound

R.10 Respond to intruder

R.1 Boot Robot Robot power is turned on

Prerequisites: The robot is not powered on.

Primary scenario:

1. An operator turns the power on by pushing the on/off button on the robot.
2. The robot is powered and ready to be started.

Alternative scenario: The battery of the robot is dead, and the robot is not powered.

R.2 Shut down Robot power is turned off.

Prerequisites: The robot is powered on.

Primary scenario:

1. An operator turns the power off by pushing the on/off button on the robot.
2. The robot is shut down.

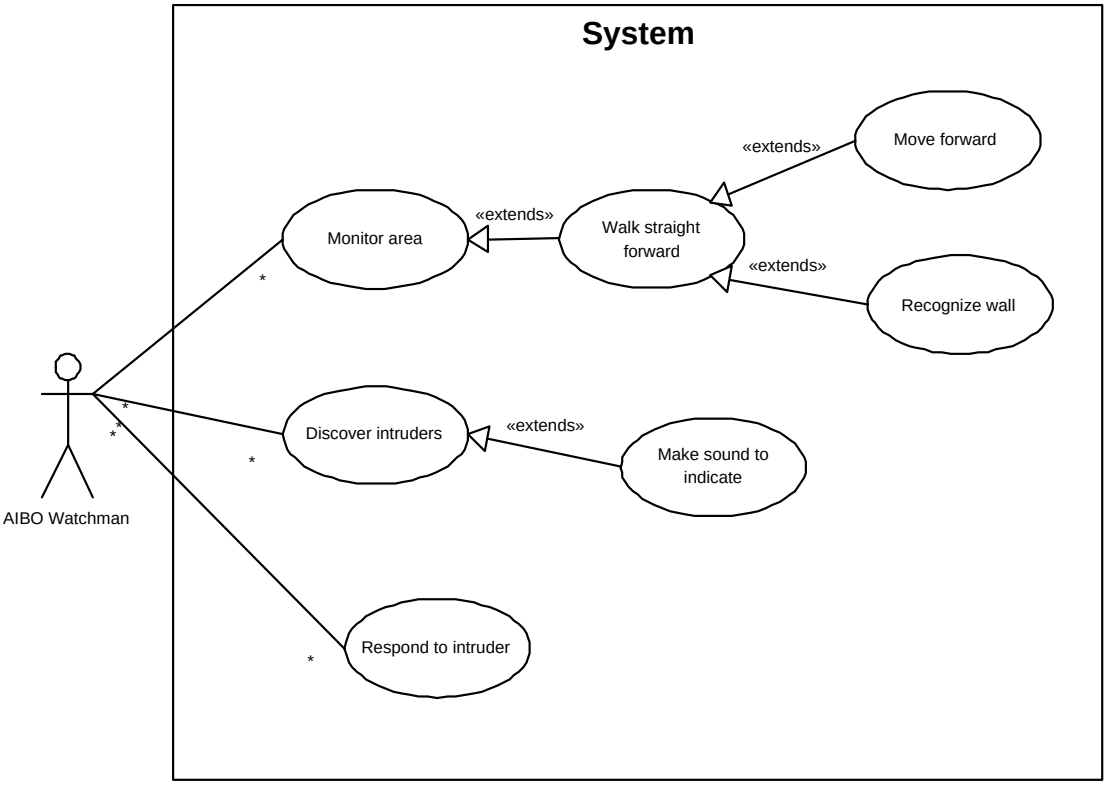

Figure 6.2: Use case diagram of autonomous robot

R.3 Start robot An operator must be able to start the robot

Prerequisites: The robot is powered on, and has sufficient battery power, the robot is not operating.

Primary scenario:

1. An operator starts the robot by double pressing the back button on the robot.
2. The robot starts operating.

Alternative scenario: The battery of the robot is dead, and the robot does not start operating.

R.4 Stop robot The robot is paused from whatever it is doing.

Prerequisites: The robot is operating.

Primary scenario:

1. An operator pauses the robot by double-pressing the back button.
2. The robot stops doing whatever task it is performing at the moment.

R.5 Move forward The robot moves forward.

Prerequisites: The robot is operating.

Primary scenario:

1. The robot moves forward

Alternative scenario: The battery of the robot has little power, and the robot doesn't move.

R.6 Walk forward To move forward along the corridor as in R.5, the robot must be able to walk forward.

Prerequisites: The robot is operating and does not detect a wall straight ahead.

Primary scenario:

1. The robot walks forward

Alternative scenario: Something get caught in the joints. The robot cannot walk and sits down, joints loose.

R.7 Recognize wall To move forward along the corridor as in R.5, the robot must be able to recognize a wall, and not bump into it.

Prerequisites: The robot is powered on, and has sufficient battery power. A wall exists in front of the robot

Primary scenario:

1. The robot detects the wall

Alternative scenario:

R.8 Discover intruder To make sure the area is without uninvited persons, the robot must be able to recognize an intruder.

Prerequisites: The robot is powered on, and has sufficient battery power. An intruder is present in the robot's area of vision.

Primary scenario:

1. The robot detects the intruder

Alternative scenario:

R.9 Make sound When the robot discovers an intruder it should make a sound.

Prerequisites: The robot is powered on, and has sufficient battery power. An intruder is just discovered by the robot.

Primary scenario:

1. The robot makes a sound to indicate it has detected an intruder

Alternative scenario:

R.10 Respond to intruder After the robot has discovered an intruder it should take a picture of the intruder. The picture is sent to the control room, and the operator decides the robot's further actions.

Prerequisites: The robot is powered on, and has sufficient battery power. An intruder is discovered.

This use case is out of scope for phase 1

6.2 Non-functional requirements

Non-functional requirements are constraints on the services or functions offered by the system. [17] Many non-functional requirements relate to the system as a whole rather than to individual system features, and not all non-functional requirements are concerned with the software of the developed system. In this study, the only non-functional requirements considered are safety requirements. Most safety requirements was discovered during the HazOp session, documented in Table 5.2 - 5.5.

S.1 The robot should not walk into walls.
From HazOp in Table 5.3, guideword **No**.

S.2 If the battery runs out of power, the robot should notify the operator and shut down in a controlled manner.
From HazOp in Table 5.2, guideword **No**.

S.3 When a joint-lock is detected, the robot should notify an operator. Joint motors must stop, the joint should be loose.
From HazOp in Table 5.3, guideword **No**.

S.4 Jerky movements in joints can cause the robot to wear out faster [11] and must be avoided. There should be a maximum angular velocity on all joints, and at startup, power should fade in slowly to the joints.
From HazOp in Table 5.3, guideword **More**.

S.5 Robot must notify the operator when the camera doesn't work, or there is no clear picture of the surroundings.
From HazOp in Table 5.4, guideword **No**.

S.6 Robot must notify the operator when the picture for the camera is of unacceptable low quality.
From HazOp in Table 5.4, guideword **Part of**.

S.7 Robot must notify the operator when there is no distance measurements.
From HazOp In Table 5.5, guideword **No**.

S.8 Robot must notify the operator if it bumps into walls but the distance reader detects the wall as further away.
From HazOp in Table 5.5, guideword **More**.

S.9 It is of great importance that the robot is operational. To make component failure less likely, the robot must be to service checks at regular intervals. The service check will include cleaning the camera and the joints and checking IR-unit/distance sensor for correctness.
From HazOp in Table 5.4, guidewords **No, Part of** and Table 5.5, guidewords **No, Less, More, Part of**.

S.10 The AIBO has very limited battery capacity (1.5 hours), and will only operate if more than 15% of the battery power remains. This means that the AIBO's battery needs to be recharged approximately every hour. To keep the safety- or security-critical area under constant surveillance, more than one robot is necessary.
Not from HazOp, but from general knowledge of the limitations of the robot. Could have been detected in HazOp with the additional guidewords in Table 2.3.

Chapter 7

Test plan

In IEC 61508 is planning of the tests for functional requirements and validation of safety requirements planned concurrently with realization of safety-related system. Testing and validation is ensures that the system is according to the systems specification, and there should be recognizable realtionship between requirements and the planned tests / validation.

Requirement	**Test**	**Repetitions**
R.1 Boot robot	The robot is turned on by pushing the power button	5
R.2 Shut down robot	The robot is shut down when pushing the power button	5
R.3 Start robot	The robot starts operating when the tail button is pushed	5
R.4 Stop robot	The robot is paused when the back button is double pressed	5
R.5 Move straight forward	The robot is able to move iin an approximately straight line in 8 meters	10
R.6 Walk forward	The robot is able to walk forward	5
R.7 Recognize wall	The robot detects a wall when placed in front of it	5 x 5 different angles
R.8 Discover intruder	The robot discovers an intruder when an intruder is present in the robot's area of vision	10
R.9 Make sound	The robot makes a sound	5
R.10 Respond to intruder	This requirement is out of scope of phaze 1	N/A

Table 7.1: Suggested tests for robot watchman

Requirement	Test	Repetitions
S.1 The robot should not walk into walls.	The robot is placed in front of a wall. It recognizes the wall and walks in another direction.	10
S.2 If the battery runs out of power, the robot should notify the operator and shut down in a controlled manner.	The robot is started and not shut down or paused until it runs out of battery power. It then notifies the operator and shuts down in a controlled manner.	10
S.3 When a joint-lock is detected, the robot should notify an operator. Joint motors must stop, the joint should be loose.	A joint lock is simulated by holding some moveable robots parts still while the robot tries to move them. The robot should notify the operator and the joints should be loose.	3 x {head, each leg }
S.4 Jerky movements in joints can cause the robot to wear out faster [11] and must be avoided. There should be a maximum angular velocity on all joints, and at startup, power should fade in slowly to the joints.	Regular testing cannot prove this. Thorough code reading neccesary	N/A
S.5 Robot must notify the operator when the camere doesn't work, or there is no clear picture of the surroundings.	Validation may prove difficult. Further studies into how the AIBO can detect that there is no picture from the camera must be carried out	N/A
S.6 Robot must notify the operator when the picture for the camera is of unacceptable low quality.	See test for F.5	N/A
S.7 Robot must notify the operator when there is no distance measurements.	Validation may prove difficult. Further studies into how the AIBO can detect that there is no distance readings from the ir-unit must be carried out	N/A
S.8 Robot must notify the operator if it bumps into walls but the distance reader detects the wall as further away.	To separate between bumping into a wall and joint lock S.3	
S.9 It is of great importance that the robot is operational. To make component failure less likely, the robot must be to service checks at regular intervals. The service check will include cleaning the camera and the joints and checking IR-unit/distance sensor for correctness.	This cannot be validated through testing. Routines for service checks is neccessary.	N/A
S.10 The AIBO has very limited battery capacity (1.5 hours), and will only operate if more than 15% of the battery power remains. This means that the AIBO'battery needs to be recharged approximately every hour. To keep the safety- or security-critical area under constant surveillance, more than one robot is nessecary.	This cannot be validated through testing. Routines for service checks is neccessary.	N/A

Table 7.2: Safety validation plan

Chapter 8

Design and Implementation

The Design and Implementation phase is the "safety-related systems realization" phase in the life cycle from IEC61508. The goal of this phase is to create a safety-related system according to the requirements. This chapter presents a visualization of the robot watchman action sequence and sequence diagrams visualizing the interaction between the different classes in the system.

The Tekkotsu framework give some direction to the software architecture, since it has good support for an event-driven architecture. For other possible known architectures, see [13].

8.1 Design and implementation of phase 1

Most of the functional requirements have been implemented. Figure 8.1 shows the action sequence for the watchman behavior. This figure is when the robot has booted, has watchman behavior loaded into the system and is not in emergency stop.

The sequence in Figure 8.1 is written as a numbered list below. The sequence diagrams in the next section will have textual descriptions relating them to this figure. Functionality that is not evident from Figure 8.1 and the list is that when the robot sees a pink ball it makes a barking sound, and when the ball is lost the robot makes a sad sound. The robot looks for pink balls all the time, event when the center tail button is not pushed.

1. An operator pushes the center tail button on the AIBO.
2. The AIBO turns it's head all the way to the left.
3. The AIBO turns it's head all the way to the right.
4. The AIBO turns it's head face forward.
5. The AIBO walks forward
6. A timer sets in after some time(15 seconds from step 1).
7. The AIBO stops walking. Steps 2-7 is repeated until the AIBO is stopped (low battery, emergency stop or turned off).

8.2 Sequence diagrams

Sequence diagrams shows the interaction between the different classes in the system. Since the robot watchman has an event-driven architecture, the sequence diagrams will show how the

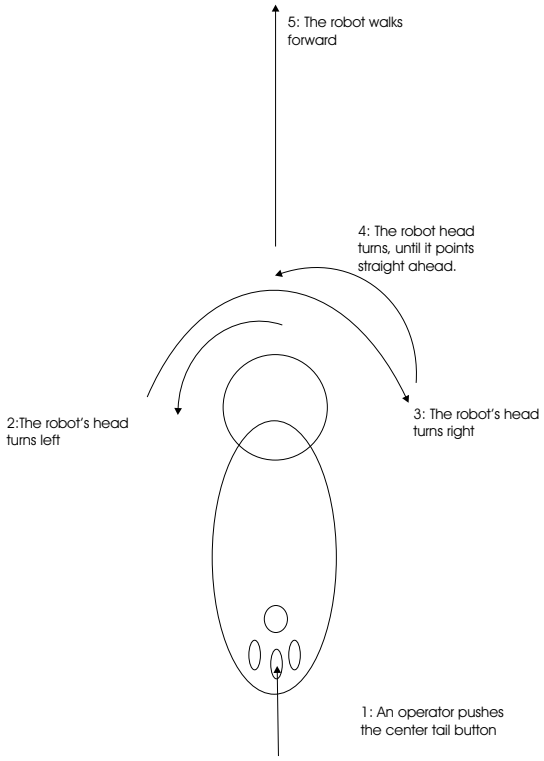

Figure 8.1: Robot action sequence

classes in the system interact at different events. The textual descriptions of the use cases contain references to Figure 8.1 so the sequence diagrams easier can be related to a robot action. The interaction when the watchman behavior is loaded into the active memory region and loaded out of it, is also shown. The source code these sequence diagrams describe can be found in Appendix C.1.

Figure 8.2 shows the sequence in WatchmanBehavior when the AIBO is booted. This is only a small part of the sequence when the AIBO boots, but the other startup processes are from the Tekkotsu framework. To relate this sequence diagram to Figure 8.1, this sequence should be finished before the operator pushes the center tail button.

When an operator pushes the center tail button, the sequence in Figure 8.3 will be executed. There are three important features in this sequence. A timer that repeats itself every 15 seconds is initiated (1), and the robot's head turns left (2), and a timer to indicate when the head has finished the turning motion is initiated (3). Related to Figure 8.1 the starting event in this sequence is the action at number 1 in the Figure 8.1, an operator pushes the center tail button. This will initiate a timer that repeats itself every 15 seconds, and is timer indicated in step number 6 in the list in Section 8.1. The robot's head turns left (number 2 in figure 8.1)and a timer that triggers after 2 seconds is initiated. 2 seconds is approximately the time it takes the AIBO to turn it's head left, and the timer is used as input in the next sequence in Figure 8.4.

After 2 seconds the timer event initiated from the sequence diagram in Figure 8.3 is sent to the WatchmanBehavior. When this happens the AIBO's head is pointing to the left. This starts the sequence diagram in Figure 8.4. To make sure the AIBO isn't moving, the WalkMC is set to

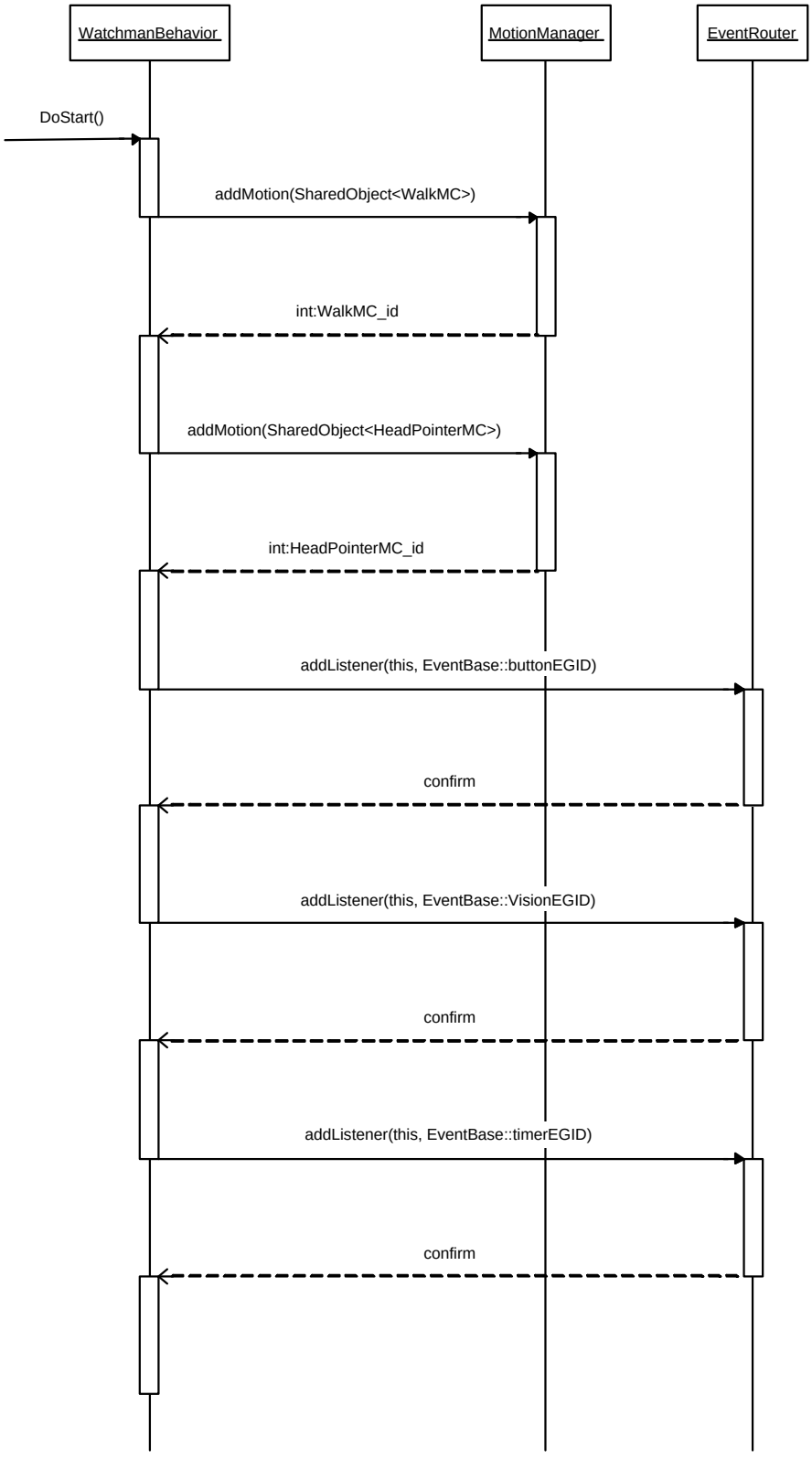

Figure 8.2: Sequence diagram of DoStart()

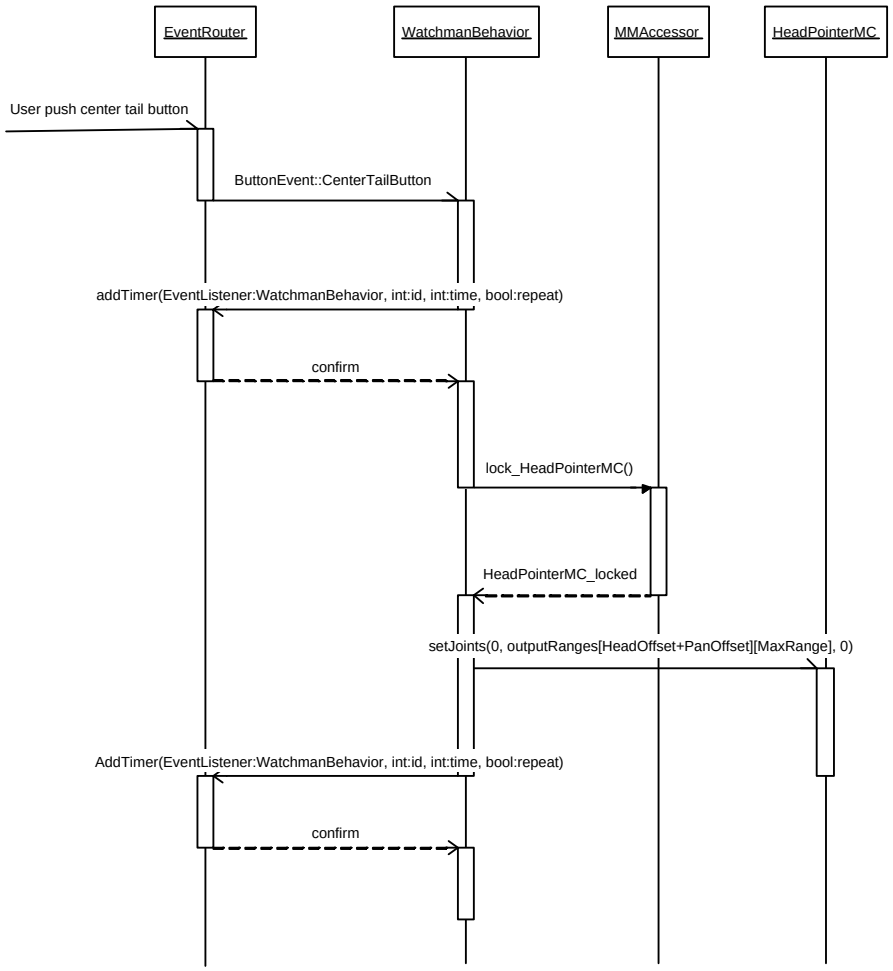

Figure 8.3: Sequence diagram when the tail button is pushed

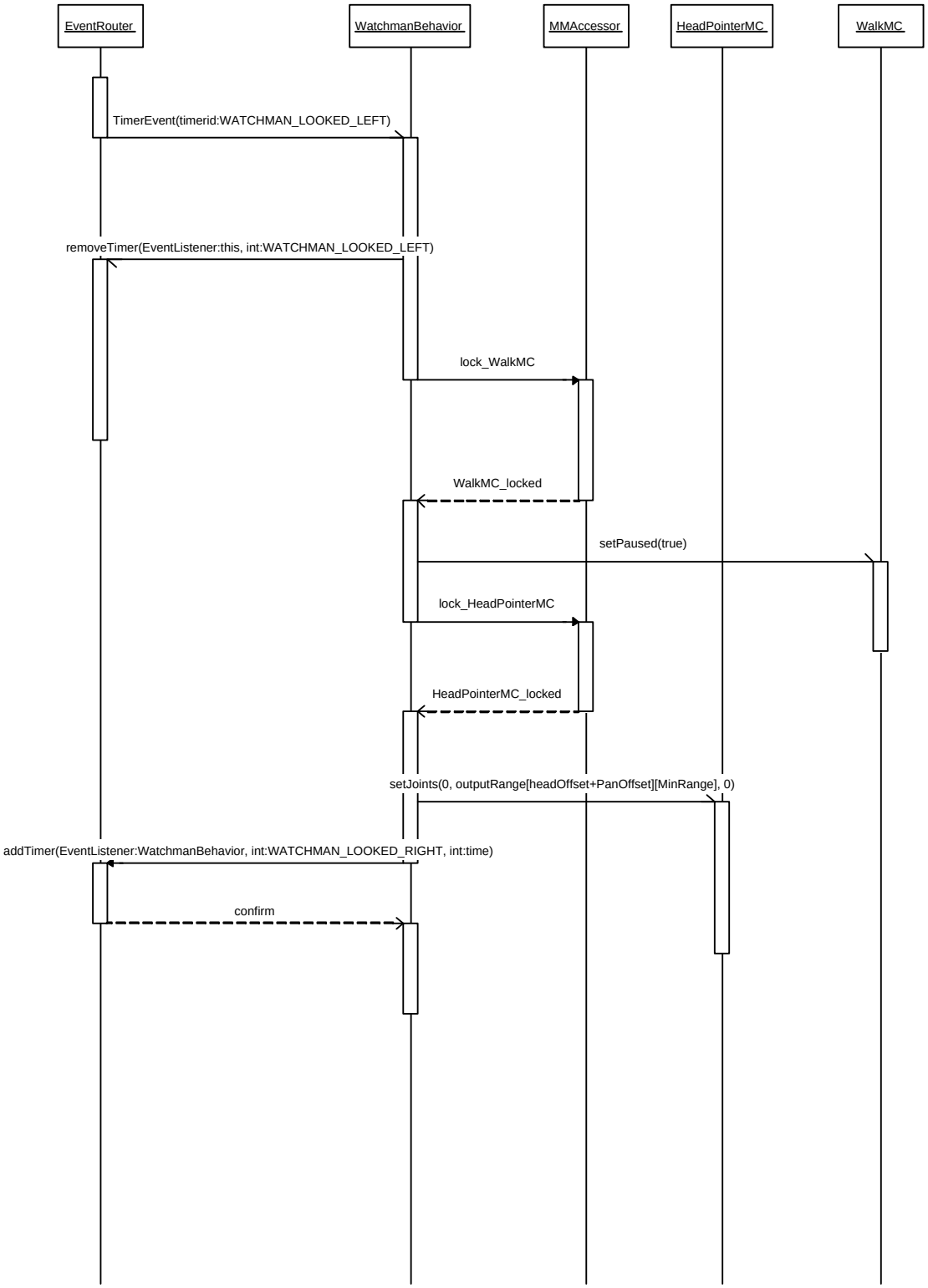

Figure 8.4: Sequence diagram when the AIBO has looked left, and is about to look right

paused and the AIBO then looks to the right. A new timer event is added to the EventRouter, it triggers after 4 seconds, since the head has to move a longer distance.

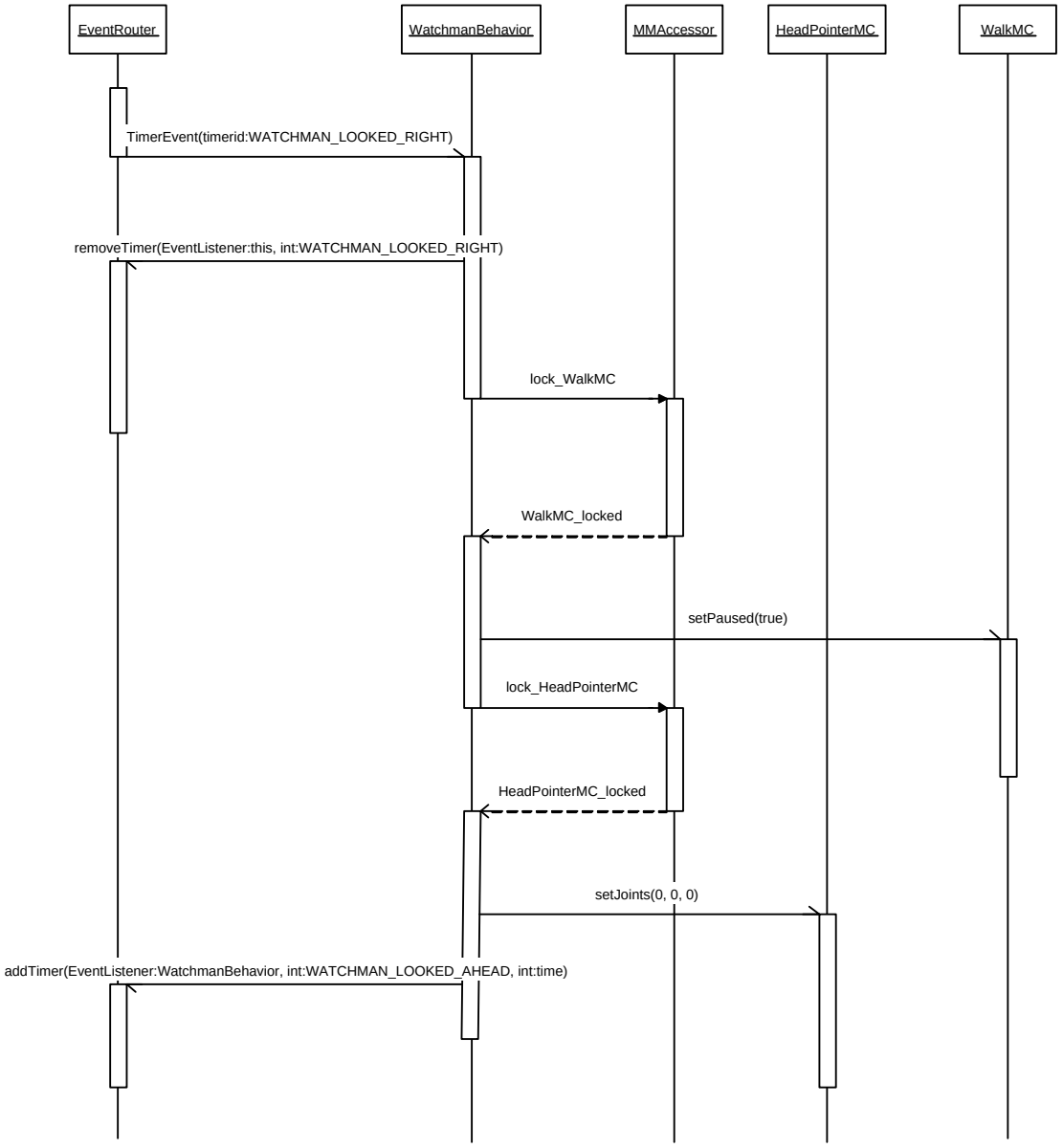

Figure 8.5: Sequence diagram when the AIBO has looked right and is about to look straight ahead

4 seconds after the sequence in Figure 8.4 a timer event is sent to the WatchmanBehavior. This is the starting event in the sequence diagram in Figure 8.5 This sequence ensures that the AIBO isn't walking by sending setPaused(true) to WalkMC and the turns the AIBO's head so it faces forward. A timer is made in the event router, it will trigger in 2 seconds, and start the sequence in Figure 8.6.

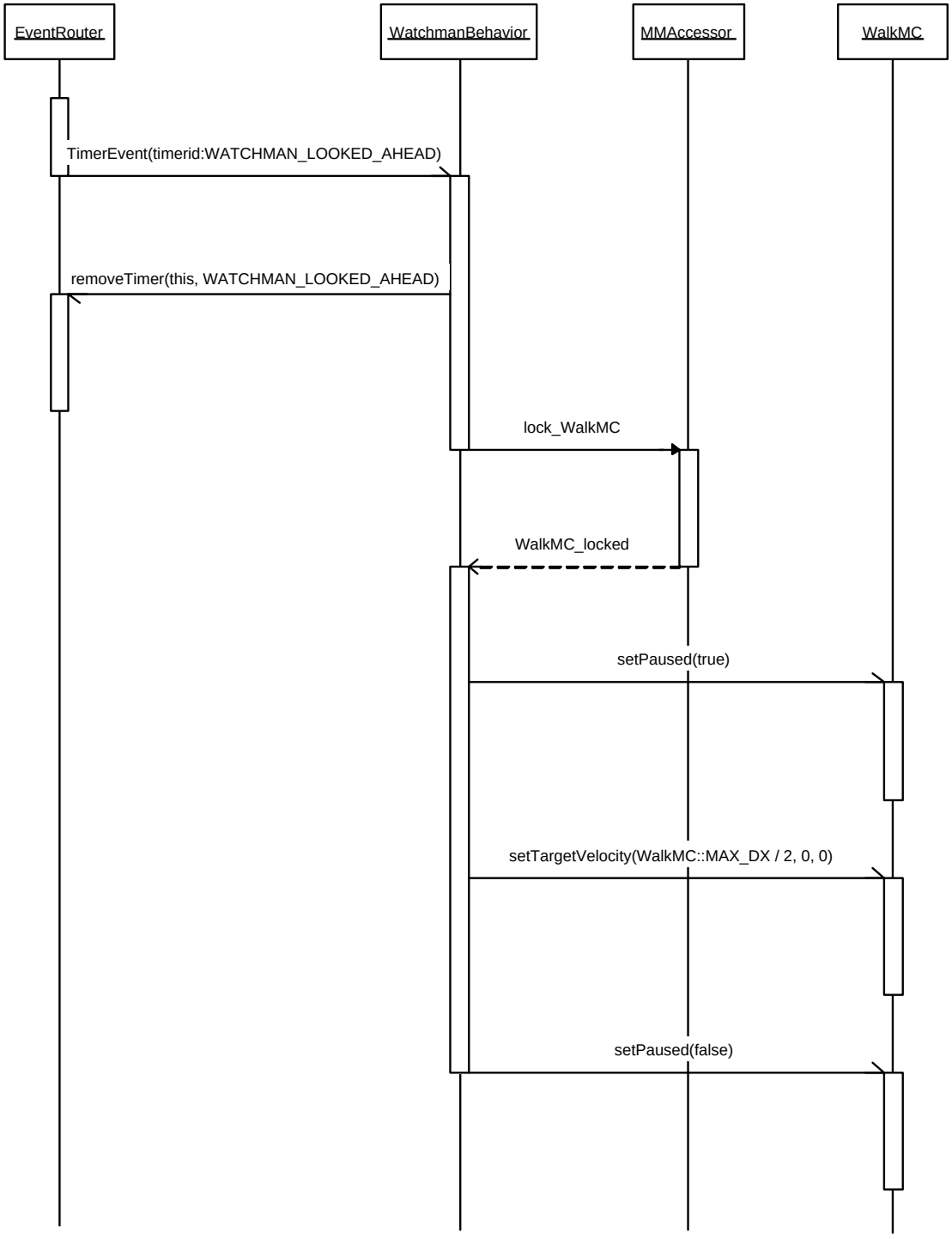

Figure 8.6: Sequence diagram when the AIBO has finished looking around and starts to move again

When the timer started from the sequence in Figure 8.5 triggers, the sequence in Figure 8.6 begins. The walking parameters in setTargetVelocity() is set, and the robot starts to walk after the message setPaused(false) is sent to WalkMC. If, in the future, somebody want to implement a more version of the robot, that doesn't walk into obstacles etc, this is where the walking angle is set. The robot keeps walking until the timer event (WATCHMAN_TIMER_EVENT) created by the sequence Figure 8.3 is sent to the WatchmanBehavior. That timer event starts the sequence in Figure 8.7

The first timer set in the sequence diagram in Figure 8.3, WATCHMAN_TIMER_EVENT, starts the sequence in Figure 8.7. The robot stops walking and starts looking around by turning the head to the left. The Functionality is similar to the one described in Figure 8.3, with the exceptions that now the robot stops walking, and the repeating timer from Figure 8.3 is not reset, since it is repeating. The robot program is now in a loop where the it looks left, looks right, looks straight forward, walks a little, stops walking, looks left, etc. The robot will continue doing the loop until it is stopped, either by low battery power, an emergency stop or by being turned off by an operator.

All the time the WatchmanBehavior is active, the AIBO looks for pink plastic balls. The sequence occurring when a pink ball is detected is shown in Figure 8.8. When a pink ball is detected by the visual system, it sends a message to the event router. The event router sends a message to WatchmanBehavior, which tells the SoundPlay object to play a wav-file. The result is that the robot makes a little barking sound.

When the robot has detected a pink ball, but the ball is lost again, another VisionEvent occurs. The sequence for this VisionEvent is shown in Figure 8.9. The actions are similar to when the robot detects a ball, but the type id of the event is different, and the sound manager plays a sad sound instead of a barking one.

Figure 8.10 shows the actions in WatchmanBehavior when the AIBO is turned off. Basically, this method cleans up all the settings made by DoStart in Figure 8.2.

8.3 Further work

The safety requirements has not been implemented. The requirements about safe shut-down and joint lock can probably be implemented by adding more events to listen to in the event router. Other safety requirements are more work-demanding, for example detecting if the picture received from the camera is of unacceptable low quality. It will be difficult to establish the difference of joint lock and damaged distance sensor, since the result of both damaged distance sensor, when the robot get stuck on a wall, and joint lock probably will be overload on the robot's motors. The somewhat jerky movements in the joints will be difficult to get around as it derives from the walking algorithm in the Tekkotsu framework. Table 8.1 lists the different requirements and their implementation status.

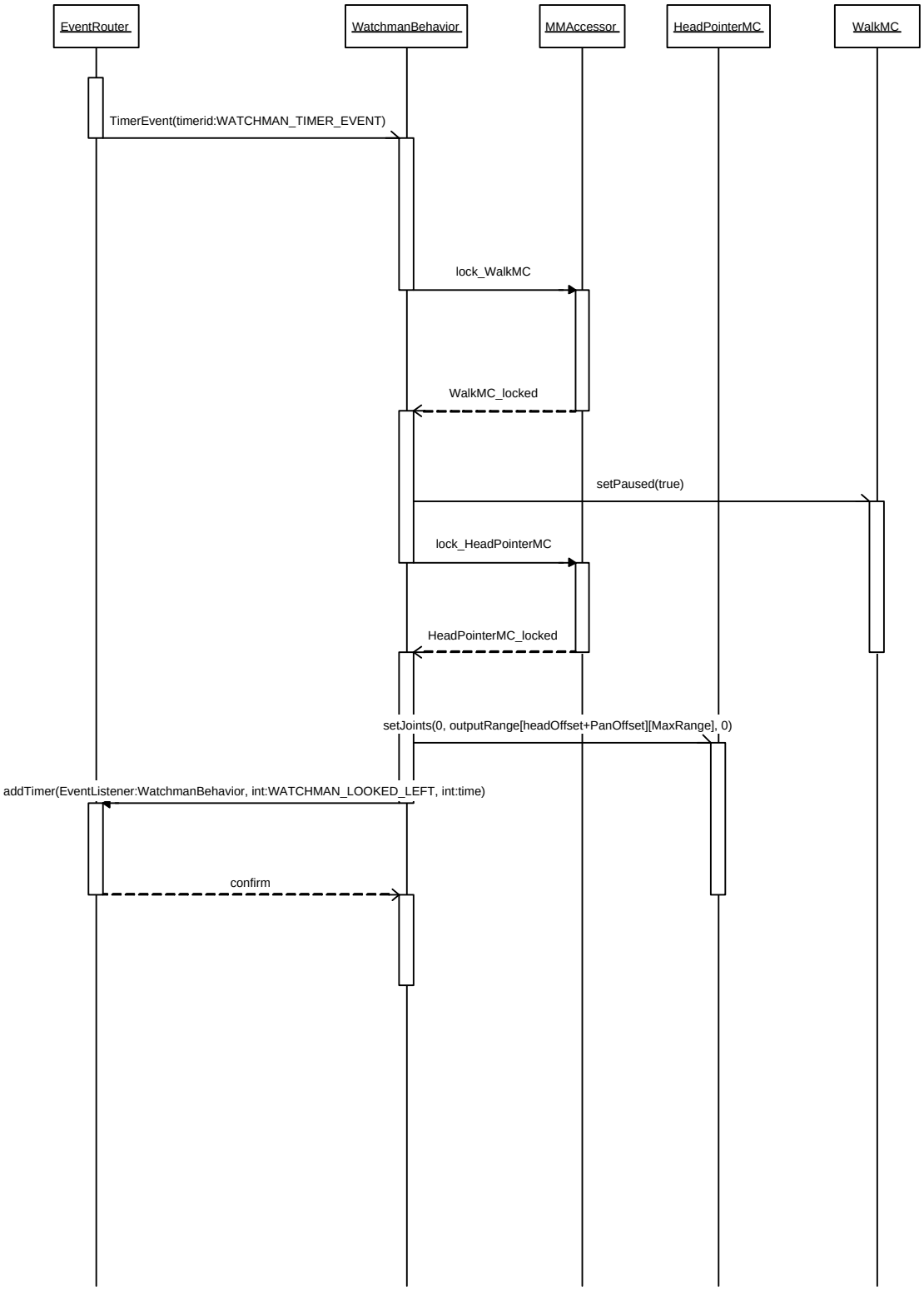

Figure 8.7: Sequence diagram when the main timer occurs

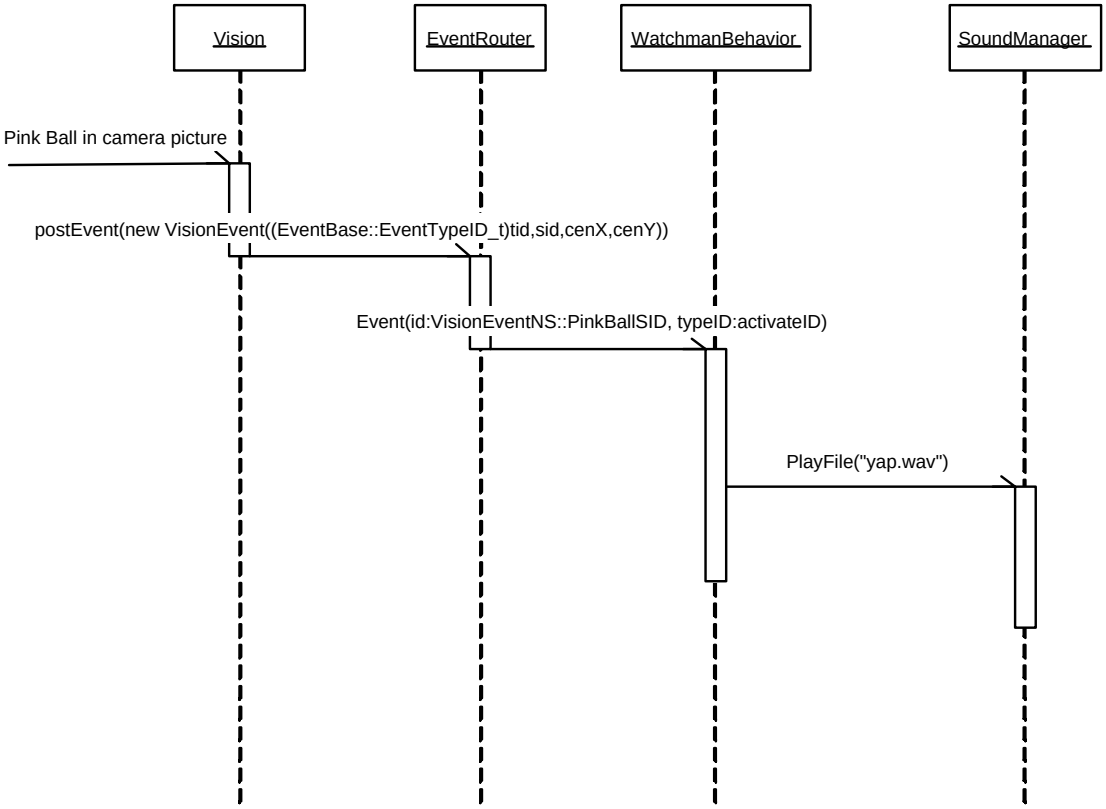

Figure 8.8: Sequence diagram when the AIBO sees a pink ball

Requirement	**Implementation status**
R.1Boot robot	Finished, the robot boots successfully.
R.2Shut down robot	Finished, the robot shuts down when the power button is pushed.
R.3Start robot	Finished, it is able to start the robot, although the procedure could be simplified, it involves a lot of pushing the correct buttons and low-quality feedback.
R.4Stop robot	Finished, the robot can easily be stopped.
R.5Move straight forward	Unfinished, is dependant on R.7.
R.6Walk forward	Finished, but with room for improvement, the motion is quite bumpy and some of the sensors (camera and IR) would get more accurate input with smoother motions.
R.7Recognize wall	Unfinished, the robot does not detect walls.
R.8Discover intruder	Finished, improvement of detection algorithm is desirable as the robot often mistakes other objects for intruders as well.
R.9Make sound	Finished.
R.10Respond to intruder	Unfinished. The manner the AIBO should respond to intruders in has not yet been decided.
All safety requirements	Unfinished.

Table 8.1: Implementation status

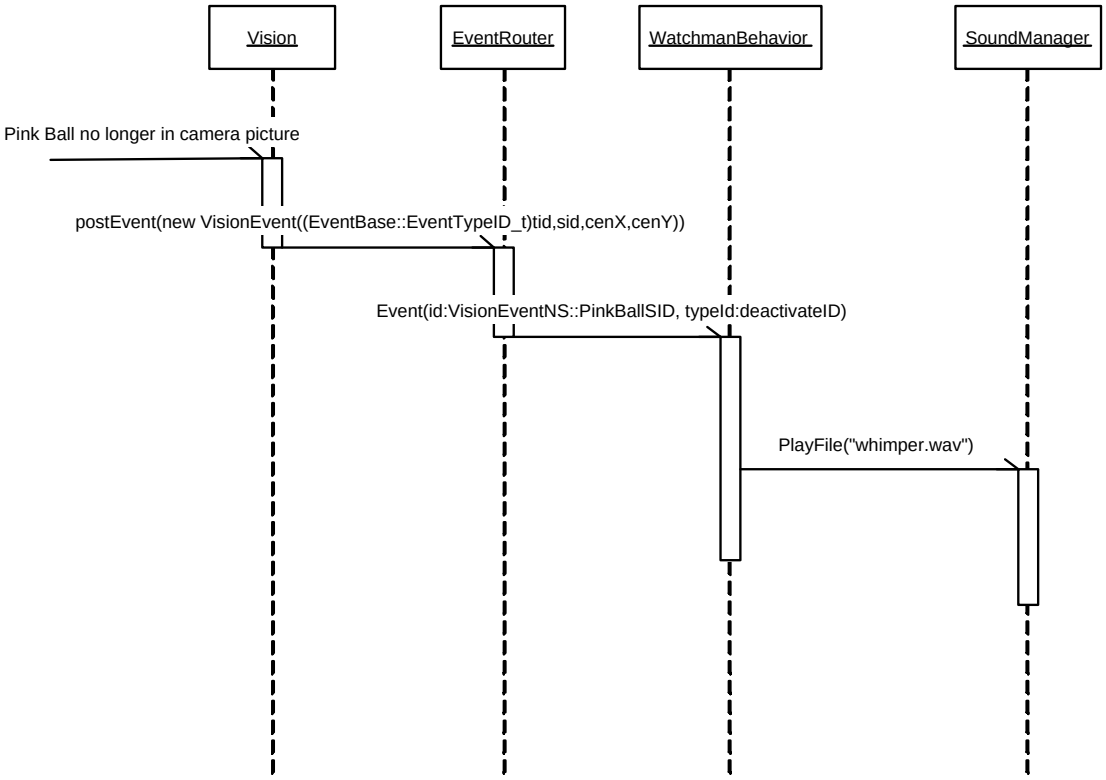

Figure 8.9: Sequence diagram when the AIBO looses sight of a pink ball

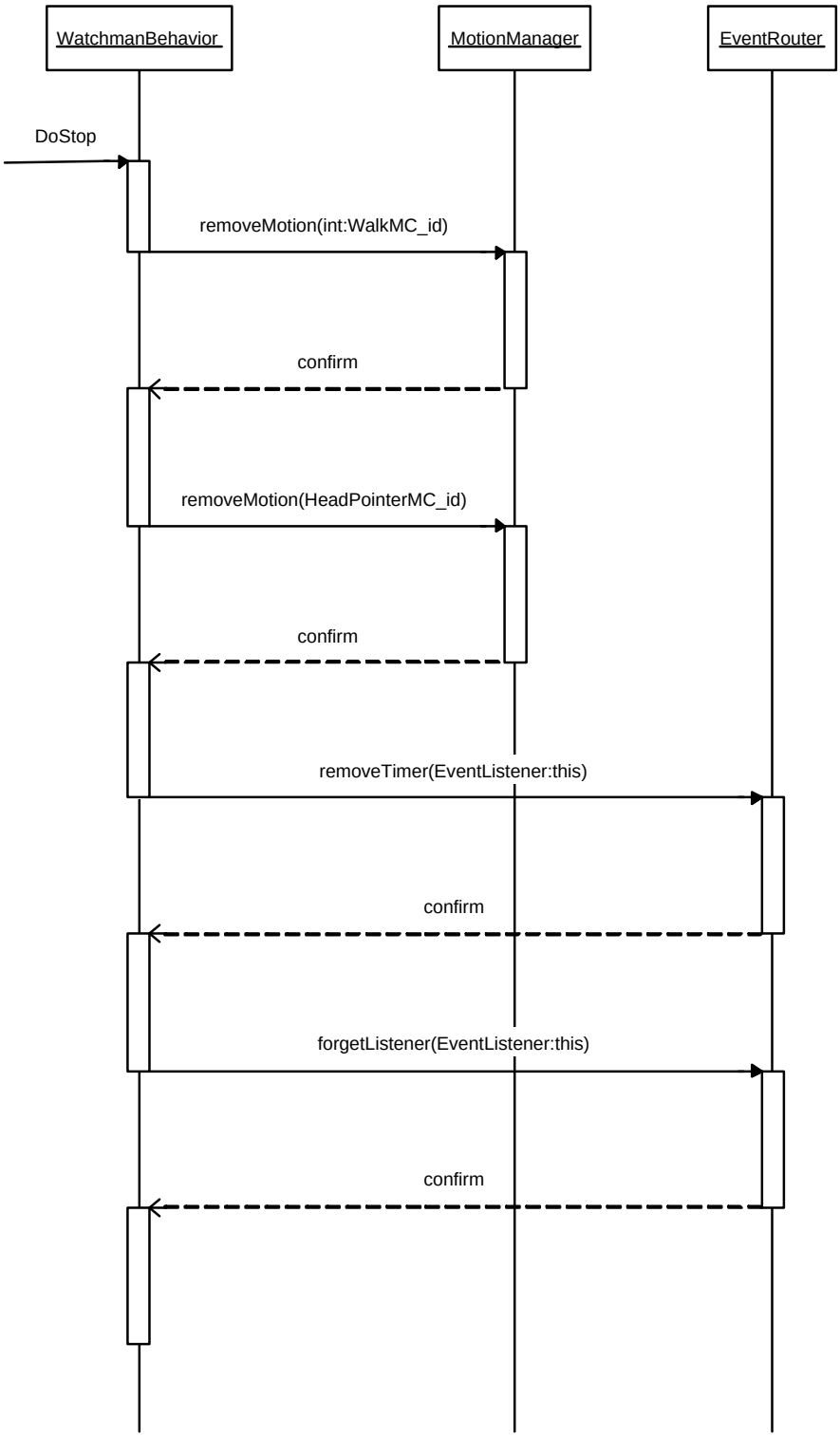

Figure 8.10: Sequence diagram of DoStop()

Chapter 9

Testing and safety validation

Testing and safety validation are two different phases in the lifecycle described in IEC 61508. Testing is a part of the "safety-related systems realization" phase, and safety validation is something that should be preformed after installation and commissioning. In the current state, the system will not be installed anywhere. Since safety validation and testing of functional requirements have several aspects in common, they are well suited to be presented together.

Testing of the product is necessary to verify that the system's functionality matches the functionality described in the requirements document of the system. The aim of the safety validation process is to prove that the product meets the safety requirements. [7]

Validation of the non-functional requirements can be harder, as non-functional requirements often "relate to the system as a whole rather than to individual system features" [17].

Not all the requirements has been implemented, the implementation status of the requirements is given in Table 8.1. The implemented requirements has been tested when they were implemented in an ad-hoc manner to see if they worked or if some aspect were unthought of. The safety requirements are not implemented, and therefore safety validation will be a waste of time at this point in the project, the validation will surely fail. Due to time limitation these phases in the lifecycle described in IEC 61508 has not been performed. The test plan in Table 7.1 and safety validation plan in Table 7.2 should be considered.

Chapter 10

Discussion, Conclusion and Further work

In this chapter, the results of the project are described and further work will be outlined.

10.1 Discussion

In this section we discuss some of the choices made during the project. The relation between functional and safety requirements, and how risk analysis have effect on that relation will be discussed. The importance of thorough testing will also be described briefly.

The development process of IEC 61508 was chosen as it is a well-tested and well-specified method that describes the life cycle in safety-critical systems. The life cycle model has also been widely accepted as the basis for specification, design and operation of Safety Instrumented Systems [15].

The selection of HazOp for the PHA came naturally, since it is one of the few structured hazard identification methods that can be used without in-depth knowledge about the components to be analyzed.

The most discussable choice in this project is the choice of the Tekkotsu framework. It has been a trade-off between safety and functionality as the Tekkotsu framework isn't safety-oriented. Tekkotsu motion commands (at least WalkMC) can cause jerky movements in the joints when the robot pauses and then starts again. But OPEN-R is not safe either, and less functionality would have been possible using only OPEN-R.

The relation between safety requirements and functional requirements is both interesting and important. Risk analysis is an important factor in that relation. Concept, or informal functional requirements, together with information about the intended physical environment are used as input to a preliminary hazard analysis (PHA). During PHA hazards are found, and then transformed to constitute safety requirements. In a thorough PHA one may also propose countermeasures that reduce the probability or consequence of the identified risks. These countermeasures are often specified as functional or safety requirements in the following iteration. When these functional requirements are added to the original requirements, a new increment with risk analysis should be carried out to ensure that the new requirements fulfill the safety requirements.

In a safety-critical system testing and safety validation crucial methods in terms of proving that the system is safe to use. Confidence is a very important factor related to the validation process. The user of the validation documents has to trust the validation quality, otherwise the validation has no meaning [25]. The number of repetitions of each test must be large enough to convince the reader that the system will probably never fail under the circumstances described in the test. It is necessary to repeat the test of a complicated function or requirement more times than a simpler one.

10.2 Conclusion

In this project an AIBO prototype of an autonomous robot watchman is designed and implemented. The implementation was carried out according to the first phases of the IEC 61508 life cycle. A preliminary hazard analysis have been carried out using HazOp. The hazards found in HazOp was then used as a basis for identifying and specifying safety requirements. All functional requirements of phase one have been implemented. Due to time limitations, the safety requirements identified have not been implemented. Further, the functional requirements have only been tested in an informal way meaning that the implemented functional requirements were tested when they were implemented, and not formally tested and verified after implementation. The safety requirements are not implemented, and therefore not validated either.

10.3 Further work

More work are required in order to achieve a safe watchman prototype. Several more increments of risk analysis, implementation and testing are needed to implement the complete set of both functionality and safety demands for the watchman prototype. The requirements identified in this report should be implemented, and some features could be added to extend the watchman's functionality.

The implementation status of the different requirements is given in Table 8.1. Formal testing has not been carried out, due to time limitations. Future testing and validation of the requirements should at least contain the tests outlined in Table 7.1 and Table 7.2. The requirements that are unfinished, or can be improved are discussed. Examples of extensions to the current functionality of the AIBO watchman is as following.

R.6 Walk forward
At the current stage the robot is able to walk forward. However, the motions are rather bumpy. Picture quality needs to be improved and therefore we need to implement smoother walking algorithm. Possibly, the IR-unit would benefit from this as well.

R.7 Recognize wall
The robot doesn't detect walls. A mapping functionality that detects obstacles using IR will probably be released in a future version of the Tekkotsu framework. This would then be a subject for future work.

R.5 Move straight forward
This requirement is dependant on the functionality described in the functional requirement R.7. An algorithm for planning where to walk is necessary if the robot shall move in complex areas. This is not yet implemented. The robot should be able to turn around corners once a path finding algorithm and wall recognition has been successfully implemented.

R.8 Discover intruder
The robot does detect pink balls, but has abnormal reactions to it, since it sees pink balls everywhere, specially in "noisy" surroundings (with several objects,) or when there are almost-pink objects around. The almost-pink objects can be, for example, an orange sofa or a red fire extinguisher. It can also detect pink balls in other areas, and seem more sensitive when there are moving objects around it. The source of this problem could be in how the AIBO codes different color or the camera calibration. A better recognition algorithm, with edge detection as well as color recognition, is recommended.

R.10 Respond to intruder
This requirement is out of scope of phase 1. When an intruder is discovered, the AIBO could take a picture and use face recognition to verify its identity. If the person was an intruder, human watchmen should be notified.

Examples of extended functionality to the AIBO watchman

The robot could be supplied with a map over it's surroundings i.e. the floor it is surveilling. This would extend the feasibility of the watchman and intorduce several new interesting aspects, for example that some areas are more important than other and need more constant surveillance. The path-planning algorithm could be more complex, to ensure that no areas would go unprotected for a long time.

Whrn a watchman robot needs to cover a large area, it will be necessary with several robot watchmen working together to achieve this. They will need a communication protocol for cooperation and to be able to send messages to each other, for example informing each other which areas they just have checked and where they are planning to go next.

At IDI there is a goal to make a small-scale safe mobile digging-machine. This is described in Section 1.2. The results of this project can't be directly applied to the digging machine project as of yet, but some of the results are however applicable. For example the experience from the Tekkotsu framework could be useful for other persons in accomplishing the mobile digging-machine project.

Bibliography

[1] ASURA RoboCup Software Kit. http://www.asura.ac/download/.

[2] Kathleen Fackelmann. For Humans and Robots, It's Puppy Love. *NJANPHA - News Source*, 2003.

[3] B.A Gran, N. Stathiakis, G. Dahll, R. Fredriksen, A.P-J. Thumem, E. Henriksen, E. Skipenes, M. Soldal Lund, K. Stølen, S.H. Houmb, E. Mork-Knudsen, and E.D. Wistløff. The coras methodology for model-based risk assesment. IST-2000-25031 coras (risk assessment of security critical systems), public deliverable.

[4] International Electrotechnical Commission. *61508-1, Functional safety of electrical/electronic/programmable electronic safety systems - Part 1: General requirements*, 1998.

[5] Nancy Leveson. *Safeware - System Safety and Computers*. Adison-Wesley Publishing Company, 1995. ISBN: 0-201-11972-2.

[6] Ørjan Markhus Lillevik. Software architecture and safety.

[7] Timo Malm and Maarit Kivipuro. *Annex 8 - Safety Validation of Complex Components*. STSARCES.

[8] Ministry of Defence. *Defence Standard 00-56 issue 2: Safety Management Requirements for Defence Systems, part 1: Requirements*, 1996.

[9] Ministry of Defence. *Defence Standard 00-58 issue2: HAZOP Studies on Systems Containing Programmable Electronics, part1: Requirements*, 2000.

[10] Hyacinth S. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 11:1–40, September 1996.

[11] OPEN-R official web site. http://openr.aibo.com/.

[12] RoboCup Official Site. www.robocup.org.

[13] Marco Schumacher. Mobile robot, 1995. http://www-2.cs.cmu.edu/afs/cs/project/compose/www/html/ModProb/MR.html.

[14] Securitas AS. Mobilt vakthold. URL: http://www.securitas.no/.

[15] Sintef web site. http://www.sintef.no.

[16] Stein-Roar Skånhaug. Architectural aspects of software for mobile robot systems, 2003.

[17] Ian Sommerville. *Software Engineering*. Pearson Education Limited, Sixth edition, 2001. ISBN: 0-201-39815-X.

[18] Sony Corporation. OPEN-R SDK Model Information for ERS-220, 2003.

[19] Sony Corporation. OPEN-R SDK Programmer's Guide, 2003.

[20] Entertainment Robot AIBO ERS-220 Operating Instructions, 2001.

[21] Sony Global - AIBO Global Link. http://www.aibo.com.

[22] AIBO Wireless LAN Card Operation instructions, 2000.

[23] Karine Sørby. Relationship between safety and security in a security-safety critical system: Safety consequences of security threats., 2003.

[24] Neil Storey. *Safety-critical computer systems*. Adison-Wesley Publishing Company, 1996. ISBN: 0-201-42787-7.

[25] STSARCES. *Final report - Safety-Related Complex Electronic Systems*, February 2000.

[26] Tekkotsu Homepage. http://www.tekkotsu.org/.

[27] Telenor. Fremtidshuset. URL: http://www.fremtidshuset.com/.

Appendix A

Glossary

This appendix contains the glossary of the report.

Agent - a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user. [10]

Reliability - the probability that a piece of equipment or component will perform its intended function satisfactorily for a prescribed time and under stipulated environment conditions. [5]

Failure - the non performance or inability of the system or component to perform its intended function for a specified time under specified environmental conditions. [5]

Error - a design flaw or deviation from a desired or intended state. [5]

Accident - an undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss. [5]

Incident - an event that involves no loss (or only minor loss) but with the potential for loss under different circumstances. [5]

Hazard - a state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object), will lead inevitably to an accident. [5]

Risk - a combination of the frequency or probability of a specified hazardous event, and its consequences. [24]

Safety - freedom from accidents or losses. [5]

Appendix B

Technical set-up

In this appendix contains technical information about the project. The hardware used in this project is described and the installation procedure for the programming environment is given. The appendix also holds a copy of the wlandflt.txt file from the memory stick that contains the AIBO WLAN settings. At the end of the Appendix, technical problems will be described.

I write the technical set-up for the tools I used in the project (programming environment, communication with AIBO etc).

B.1 Hardware

This section lists the hardware used in this project. The hardware used in this project has been:

- A PC running Windows XP
- Sony Memory Stick Reader/Writer (MSAC-US1)
- Compaq WL110 Wireless PC Card
- Sony Entertainment Robot AIBO ERS-220A
- Linux servers at NTNU

B.2 Installing AIBO programming environment

To install AIBO programming environment I recommend using a Linux environment. Not Cygwin, the Unix shell for Windows recommended by SONY on the OPEN-R website, but Linux. I had some problems with Cygwin and the Tekkotsu framework, and the Tekkotsu developers only use Cygwin for testing, they do not have Cygwin support or help. This installation procedure is for Linux only, I'm not sure it will work with anything else.

B.2.1 OPEN-R SDK Installation

This section describes how to install OPEN-R SDK in a Linux environment. For other operating systems, the reader is referred to the OPEN-R website [11]. Several files are necessary to successfully install OPEN-R SDK. They are accessible from the OPEN-R website.
All systems: OPEN_R_SDK-1.1.4-r1.tar.gz
For Linux: gcc-3.2.tar.gz (do not unzip)
binutils-2.13.tar.gz (do not unzip)
newlib-1.10.0.tar.gz (do not unzip)

build-devtools-3.2-r1.sh

Unzip OPEN_R_SDK-1.1.4-r1.tar.gz anywhere you'd like to have the OPEN-R framework. If you have access to /usr/local/ you can unzip it there, but it is no problem running the framework from somewhere else. If you unzip it in another folder, you have to add a line in your .bashrc file. It should look like this:
export OPENRSDK_ROOT = /myfolder/OPEN_R_SDK

Save gcc-3.2.tar.gz, binutils-2.13.tar.gz, newlib-1.10.0.tar.gz and build-devtools-3.2-r1.sh in a temporary folder. In build-devtools-3.2-r1.sh, edit the line:
PREFIX=/usr/local/OPEN_R_SDK

This reflects where you installed the OPEN-R SDK. If you installed OPEN-R in /usr/local/, you don't have to change it at all. Run build-devtools-3.2-r.1.sh from the temporary folder, and wait for it to install. The OPEN-R SDK should be working now. You can test it by running one of the sample programs in OPEN_R_SDK-sample-1.1.4-r1.tar.gz from the OPEN-R website.

B.2.2 Tekkotsu Installation

Download Tekkotsu_1.5.tar.gz and unzip it in /usr/local/, or another folder.

If you unzipped Tekkotsu in another place than /usr/local/ or you don't have the memory stick at mnt/memstick, add two lines in your .bashrc file:

export TEKKOTSU_ROOT = /myfolder/Tekkotsu_1.5 (the root Tekkotsu folder) export MEMSTICK_ROOT = /myfolder/ (the place your files will be compiled to, to be copied to the memory stick later.)

B.3 AIBO WLAN settings

The AIBO communicates through a WLAN. AIBO needs a file so it can find the WLAN settings, the file is called either wlandflt.txt or wlanconf.txt. The WLAN settings are presented in Table B.1.

HOSTNAME=AIBO
ETHER_IP=10.0.1.100
ETHER_NETMASK=255.255.255.0
IP_GATEWAY=10.0.1.100
ESSID=AIBONET
WEPENABLE=1
WEPKEY=AIBO2
APMODE=2
CHANNEL=3

Table B.1: AIBO WLAN settings

B.4 Technical problems

The memory stick reader/writer is not always found on the PC since the PC tends to forget that it's there. To make the PC find the reader/writer again, the add hardware function of the control panel can be used. the PC then searches for devices on all its ports and eventually finds the reader writer again.

Installing OPEN-R and Tekkotsu was quite a challenge. Sony recommend using cygwin for OPEN-R, while Tekkotsu recommends installing a feature called rsync for file transfer. rsync is available for cygwin, however when cygwin has rsync installed, gcc used to compile files for OPEN-R is corrupted. I figured the solution was to install OPEN-R on a Linux server. Then I had a new problem, as I don't have administrator rights on the server, and therefore not access to /usr/local/.

The version of OPEN-R SDK I tried to install, from the file OPEN_R_SDK-1.1.3-r2.tar.gz , could be installed other places than /usr/local/, but a special compiling command was necessary, and Tekkotsu didn't have support for that special command.

After looking in several mail archives and the release of a new Tekkotsu version, I tried installing OPEN-R and Tekkotsu again. A new version of OPEN-R was available, and I downloaded it, hoping that it would be possible to install OPEN-R somewhere else than /usr/local/.

The user documentation for OPEN-R was not updated, so after looking in a small text file, called CHANGES_E.txt, I finally found the solution, an environment variable called OPENRSDK_ROOT. When this was discovered my problems were solved, and the rest of the installation was really easy.

Appendix C

Program files created or modified in this project

This appendix contains a copy of the source files created or modified during this project. The file WatchmanBehavior.h is created during this project. It's functionality is described in Chapter 8. The other file in this appendix, StartupBehavior.cc, originates from the Tekkotsu framework and is the default startup behavior in Tekkotsu. It initiates the menu system described in Subsection 3.3.2. The watchman behavior is added into the menu initiated from StarupBehavior.

C.1 WatchmanBehavior.h

This is the that contains the Watchman Behavior. It makes the AIBO look around, walk and look around again. It has a lot of references to other classes. Copies of these classes .h files can be found in Appendix D.

```
//I think the two lines below is included to ensure that only one instance of
//WatchmanBehavior.h is loaded into memory.
#ifndef INCLUDED_WatchmanBehavior_h_
#define INCLUDED_WatchmanBehavior_h_

//The timer event ids. Makes the code more readable and helps avoiding
//misunderstandings.
#define WATCHMAN_TIMER_EVENT 1
#define WATCHMAN_LOOKED_LEFT 2
#define WATCHMAN_LOOKED_RIGHT 3
#define WATCHMAN_LOOKED_AHEAD 4

// Referenced files
#include "Behaviors/BehaviorBase.h"
#include "Motion/MotionManager.h"
#include "Motion/MMAccessor.h"
#include "Motion/WalkMC.h"
#include "Motion/HeadPointerMC.h"
#include "Shared/SharedObject.h"
#include "SoundPlay/SoundManager.h"

class WatchmanBehavior : public BehaviorBase {

public:
```

```
  WatchmanBehavior() : BehaviorBase (), walk_id(MotionManager::invalid_MC_ID),
head_point_id(MotionManager::invalid_MC_ID) {
  };

  //When the Behavior is loaded into memory.
  virtual void DoStart() {

    //Superclass DoStart
    BehaviorBase::DoStart();

    //Adds the walk motion command to the motion manager
    walk_id = motman->addMotion(SharedObject<WalkMC>());

    //Adds head pointer motion command to motion manager
    head_point_id = motman->addMotion(SharedObject<HeadPointerMC>());

    //Adds head pointer motion command to motion manager
    head_point_id = motman->addMotion(SharedObject<HeadPointerMC>());

    //Adds an event listener in the global event router on buttons.
    erouter->addListener(this, EventBase::buttonEGID);

    //Adds an event listener in the global event router on vision events
    erouter->addListener(this, EventBase::visionEGID);

    //Adds an event listener in the global event router on timer events
    erouter->addListener(this, EventBase::timerEGID);
  }

  // To clean up when the Behavior is loaded out of memory.
  virtual void DoStop() {

    //Removes walk motion command from the motion manager
    motman->removeMotion(walk_id);

    //Removes head pointer motion command from motion manager
    motman->removeMotion(head_point_id);

    //Removes timers to this event from event router
    erouter->removeTimer(this);

    //Remove all the event listeners this Behavior added in the event router
    erouter->forgetListener(this);

    //Superclass DoStop()
    BehaviorBase::DoStop();
  }

  //To identify the Behavior in Tekkotsu menus
  virtual std::string getName() const {
    return "WatchmanBehavior";
  }
```

```
  // When an event happens. The Behavior is only notified about events types
  //it has registered as important in with the erouter->addListener command.
  virtual void processEvent(const EventBase& event){

    //To start the robot after boot, an operator pushes the center tail button.
    if(event.getGeneratorID() == EventBase::buttonEGID &&
    event.getSourceID() ==TailCenterButOffset ){

      //A main timer is added. This makes the robot look around every 15 seconds.
      erouter->addTimer(this, WATCHMAN_TIMER_EVENT, 15000, true);

      //After the timer is started, the robot looks left.
      lookAround(WATCHMAN_TIMER_EVENT);
    }

    //Timer event when the robot stops walking and looks around. This happens
    //for all timer events, event if the robot is paused to ensure that it is
    //not moving while looking around.
    if(event.getGeneratorID() == EventBase::timerEGID){

      //Getting pointer to the walk motion command
      MMAccessor<WalkMC> walk_mc(walk_id);

      //Pausing the robots walking.
      walk_mc->setPaused(true);

      //Writing a line to a console window connected through telnet on port 59000.
      cout << "Stop walking" << endl;

      //It the source id is a regular timer event (every 15 seconds) the robot
      //starts looking left
      if(event.getSourceID() == WATCHMAN_TIMER_EVENT){
cout << "lookaround timer event" << endl;
lookAround(WATCHMAN_TIMER_EVENT); //looks left
      }

      //This is the timer when the robot has looked left, and should start to look right
      if(event.getSourceID() == WATCHMAN_LOOKED_LEFT){
cout << "looked left timer" << endl;

//To be sure the timer will not repeat itself, it is removed from the event router.
erouter->removeTimer(this, WATCHMAN_LOOKED_LEFT);
lookAround(WATCHMAN_LOOKED_LEFT); //looks right
      }

      //This timer is when the robot has looked right and is about to
      //look straight ahead again.
      if(event.getSourceID() == WATCHMAN_LOOKED_RIGHT){
cout << "looked right timer" << endl;

//to be sure the timer does not repeat itself, it is removed from the event router.
erouter->removeTimer(this, WATCHMAN_LOOKED_RIGHT);
lookAround(WATCHMAN_LOOKED_RIGHT);//looks ahead
```

```
      }

      //This timer occurs when the robot is looking straight ahead and is about
      //to start walking.
      if(event.getSourceID() == WATCHMAN_LOOKED_AHEAD){
cout<< "Finished looking around. " << endl;

//Removes the timer to ensure that it will not repeat itself.
erouter->removeTimer(this, WATCHMAN_LOOKED_AHEAD);

//When a map is included, this is where the map should be processed
        //and the walk direction parameters should change into something more
        //intelligent, according to obstacles.

//Right now the robot is walking straight ahead, or trying to, at least, it does
        //not move in a prefectly straight line.
walk_mc->setTargetVelocity(WalkMC::MAX_DX / 2, 0, 0);

//The walk motion command is not paused anymore, and the robot starts walking again.
walk_mc->setPaused(false);
cout << "walking" << endl;
      }
    }

    //If the robot sees a pink ball for the first time, a vision event is generated,
    //and the event type is activate. The robot makes a little barking sound.
    if(event.getSourceID() == VisionEventNS::PinkBallSID &&
        event.getTypeID()==EventBase::activateETID){

      //A command is given to the sound manager to play the wav file yap.wav
      sndman->PlayFile("yap.wav");
      cout << "A ball!! :)" << endl;
    }

    //When the pink ball passes out of the robot's area of vision, a vision
    //event with source pink ball and type deactivate is recieved. The robot makes a sad sound.
    if(event.getSourceID()==VisionEventNS::PinkBallSID &&
         event.getTypeID() == EventBase::deactivateETID ){

      //A command is given to the sound manager to play the wav file whimper.wav
      sndman->PlayFile("whimper.wav");
      cout << "Lost ball! :(" <<endl;
    }
  }

protected:

  // This are the id-numbers the walk motion command and head pointer motion
  //command has in the motion manager.
  MotionManager::MC_ID walk_id;
  MotionManager::MC_ID head_point_id;

  //This is the method for the robot to look around. Is called from ProcessEvent at
```

```
  //the timer events WATHCMAN_TIMER_EVENT, WATCHMAN-LOOKED_LEFT and
  //WATCHMAN_LOOKED_RIGHT. The parameter event_int is the timer event id.
  virtual void lookAround(int event_int){

    //Gets access to the head pointer motion command, used to point the head in
    //different directions.
    MMAccessor<HeadPointerMC>head_mc(head_point_id);

    //Sets max speed on the heads movement from side to side. It is a little slow,
    //so the infrared sensor has time to get information about distances, and for the
    //camera to get good pictures. The infra red sensor is not used now, and I'm not
    //sure if it needs to be this slow or if it can be faster without ruining the
    //picture quality and distance readings.
    head_mc->setMaxSpeed(PanOffset, 1.0);

    //The robot does different things for the different event ids.
    switch(event_int){
    case WATCHMAN_TIMER_EVENT:
      //Turns the head left. outputRanges is described in RobotInfo
      head_mc->setJoints(0.0, outputRanges[HeadOffset+PanOffset][MaxRange], 0.0);

      //Adds a timer to the event router. Testing has showed that it takes approximately
      //2 seconds for the head to turn left.
      erouter->addTimer(this,WATCHMAN_LOOKED_LEFT, 2000, false ); break;

    case WATCHMAN_LOOKED_LEFT:

      //Turns the head right
      head_mc->setJoints(0.0, outputRanges[HeadOffset+PanOffset][MinRange], 0.0);

      //Adds a timer to the event router. Since the moving from left to right is
      //twice as far as moving from center to left or from right to center, the timer
      //is longer too.
      erouter->addTimer(this, WATCHMAN_LOOKED_RIGHT, 4000, false); break;

    case WATCHMAN_LOOKED_RIGHT:

      //The robot should look straight ahead again.
      head_mc->setJoints(0.0, 0.0, 0.0);

      //Adding timer to the event router.
      erouter->addTimer(this, WATCHMAN_LOOKED_AHEAD, 2000, false); break;

      // If the event is something other than WATCHMAN_TIMER_EVENT, WATHCMAN_LOOKED_LEFT
      //or WATCHMAN_LOOKED_RIGHT, something strange has happened, as is it never supposed
      //to happen. A message to the console (if connected) is sent to notify that
      //something is wrong.
    default: cout << "Something is quite wrong..." << endl;
    }
  }
};

//Ends the #ifndef on the top of the file.
```

```
#endif
```

C.2 StartupBehavior.cc

StartupBehavior.cc is the default startup file in the Tekkotsu framework. It initiates the menu described in Subsection 3.3.2. StartupBahavior.cc is modified include WatchmanBehavior in the menu system. A copy of the modified StartupBehavior.cc is included in this subsection, the most important referenced files can be found in Appendix D.

```
#include "StartupBehavior.h"

#include "Behaviors/Controller.h"
#include "Behaviors/Controls/BatteryCheckControl.h"
#include "Behaviors/Controls/ControlBase.h"
#include "Behaviors/Controls/HelpControl.h"
#include "Behaviors/Controls/RebootControl.h"
#include "Behaviors/Controls/ShutdownControl.h"

//---------Line added for Watchman project----------
#include "Behaviors/Controls/BehaviorSwitchControl.h"

#include "Motion/MotionCommand.h"
#include "Motion/PostureMC.h"
#include "Motion/EmergencyStopMC.h"
#include "Motion/PIDMC.h"
#include "Motion/MotionSequenceMC.h"
#include "Motion/MMAccessor.h"

#include "Vision/Vision.h"

#include "SoundPlay/SoundManager.h"

#include "Shared/ERS210Info.h"

//---------Line added for Watchman project----------
#include "WatchmanBehavior.h"


StartupBehavior theStartup; //!< used to initialize the global ::startupBehavior, used by MMCombo
BehaviorBase& startupBehavior=theStartup; //!< used by MMCombo as the init behavior

StartupBehavior::StartupBehavior()
    : BehaviorBase(), spawned(),setup(),
    stop_id(MotionManager::invalid_MC_ID),
    pid_id(MotionManager::invalid_MC_ID)
{
    AddReference(); // this is a global, so there's a global reference
}

StartupBehavior::~StartupBehavior() {cout << "Goodbye" << endl;}

void StartupBehavior::DoStart() {
    BehaviorBase::DoStart();
```

```
    //This will "fade" in the PIDs so the joints don't jerk to full power, also looks cooler
    pid_id=motman->addMotion(SharedObject<PIDMC>(0),MotionManager::kEmergencyPriority+1,false);
    //also, pause before we start fading in, PIDs take effect right away, before the
    //emergencystop is picked up
    erouter->addTimer(this,0,4*FrameTime*NumFrames,true);

    //This is the default emergency stop
    const SharedObject<EmergencyStopMC> stop;
    stop->LoadFile("/ms/data/motion/liedown.pos"); //This *should* be replaced
         //by the current position, but just in case, better than setting everything to 0's
    stop->setStopped(true,false); //if you want to start off paused
    //sndman->StopPlay(); //so it doesn't play the halt sound the first time
           //(but the false in previous line does that now)
    stop_id=motman->addMotion(stop,MotionManager::kEmergencyPriority);

    //This displays the current battery conditions
    BatteryCheckControl batchk;
    batchk.activate(MotionManager::invalid_MC_ID,NULL);
    // const SharedObject<LedMC> led; //! @todo LedMC's don't support
       //autopruning yet, it should for uses like this, or could the one in Controller
    // batchk.activate(motman->addMotion(led,true));
    batchk.deactivate();

    //This is for the menu system
    Controller * controller=new Controller;
    controller->DoStart();
    controller->setEStopID(stop_id);
    controller->setRoot(SetupMenus());
    wireless->setReceiver(sout, Controller::console_callback);
    spawned.push_back(controller);

    sndman->PlayFile("roar.wav");

    //This will close the mouth so it doesn't look stupid
    //Now done by setting the emergency stop directly in processEvent, but left as demo code:
    /* const SharedObject<PostureMC> closemouth;
closemouth->setJointCmd(MouthOffset,outputRanges[MouthOffset][MaxRange],1);
motman->addMotion(closemouth,MotionCommand::kEmergencyPriority+2,true);
    */
}

void StartupBehavior::DoStop() {
    for(std::vector<BehaviorBase*>::iterator it=spawned.begin(); it!=spawned.end(); it++)
(*it)->DoStop();
    motman->removeMotion(stop_id);
    BehaviorBase::DoStop();
}

/*!Uses a few timer events at the beginning to fade in the PID values, and closes the mouth too*/
void StartupBehavior::processEvent(const EventBase&) {
    static unsigned int start_time=-1U;
    const unsigned int tot_time=2047; //if this is set to 2048, i sometimes get funny
```

```
     //errors: [oid:80000034,prio:1] AGRMSDriver::SetGain() : 0x0A IS USED FOR GAIN SHIFT VALUE.
    if(start_time==-1U) { //first time
start_time=get_time();
MMAccessor<EmergencyStopMC>(stop_id)->takeSnapshot(); //take new snapshot
          //with hopefully valid data
    } else {
float power=(get_time()-start_time)/(float)tot_time;
if(power>1)
    power=1;
{ MMAccessor<PIDMC>(pid_id)->setAllPowerLevel(power); }
if(state->robotDesign & WorldState::ERS210Mask)
    { MMAccessor<EmergencyStopMC>(stop_id)->setOutputCmd(ERS210Info::MouthOffset,
                outputRanges[ERS210Info::MouthOffset][MaxRange]); }
    }
    if((get_time()-start_time)>=tot_time) {
erouter->removeTimer(this);
motman->removeMotion(pid_id);
pid_id=MotionManager::invalid_MC_ID;
    }
}

/*class WalkToPinkBallFactory : public Factory<WalkToTargetMachine> {
  public:
  static WalkToTargetMachine* construct() { return new
      WalkToTargetMachine(VisionEventNS::PinkBallSID); }
  };
*/

ControlBase* StartupBehavior::SetupMenus() {
    std::cout << "CONTROLLER-INIT..." << std::flush;
    ControlBase* root=new ControlBase("Root Control");
    setup.push(root);

    // additional controls will be submenus of root:
    {
SetupModeSwitch();
SetupBackgroundBehaviors();
SetupTekkotsuMon();
SetupStatusReports();
SetupFileAccess();
SetupWalkEdit();
addItem(new ControlBase("Shutdown?"));
startSubMenu();
{
    addItem(new ShutdownControl());
    addItem(new RebootControl());
}
endSubMenu();
addItem(new HelpControl(root));

//————————-Line added for Watchman project addItem(new BehaviorSwitchControl¡WatchmanBehavior¿("WatchmanBel
false));

    }
```

```
    if(endSubMenu()!=root)
cout << "\n*** WARNING *** menu setup stack corrupted" << endl;
    cout << "DONE" << endl;
    return root;
}

void StartupBehavior::startSubMenu() {
    setup.push(setup.top()->getSlots().back());
}

void StartupBehavior::addItem(ControlBase * control) {
    setup.top()->pushSlot(control);
}

ControlBase* StartupBehavior::endSubMenu() {
    ControlBase * tmp=setup.top();
    setup.pop();
    return tmp;
}
```

Appendix D

Referenced Tekkotsu Files

In this appendix a copy of the referenced .h files from WatchmanBehavior.h is provided. We have also included some other files, since they are active in the Tekkotsu menu, and their functionallity greatly enhances the performance of WatchmanBehavior. The files with functionality adding to WatchmanBehavior are "Behaviors/Demos/AutogetupBehavior.h" which helps the AIBO get up on its feet if it realizes that it has fallen through the gravity sensor, and "Motion/EmergencyStopMC.h" that provides a stop mechanism accessable through a wireless network or by double pressing the back button on the AIBO.

D.1 Behaviors/BehaviorBase.h

```
//-*-c++-*-
#ifndef INCLUDED_BehaviorBase_h_
#define INCLUDED_BehaviorBase_h_

#include "Events/EventListener.h"
#include "Shared/ReferenceCounter.h"
#include <string>

//! The basis from which all other Behaviors should inherit
/*! Makes use of ReferenceCounter so that behaviors can automatically delete themselves if wanted\
 *  Make sure your own DoStart and DoStop call BehaviorBase::DoStart (or Stop) to
 * allow this behavior... otherwise you'll get memory leaks */
class BehaviorBase : public ReferenceCounter, public EventListener {
public:
    //! constructor
    BehaviorBase() : ReferenceCounter(), EventListener(), started(false) {}
    //! copy constructor; assumes subclass handles copying approriately - i.e. if
       @a b is active, the copy will be as well, even though DoStart was never called..
    BehaviorBase(const BehaviorBase& b) : ReferenceCounter(b), EventListener(b),
         started(b.started) {}
    //! assignment operator; assumes subclass handles assignment appropriately - i.e.
       if @a b is active, the copy will be as well, even though DoStart was never called..
    BehaviorBase& operator=(const BehaviorBase& b) { ReferenceCounter::operator=(b);
      EventListener::operator=(b); started=b.started; return *this; }

    //! destructor - if is active when deleted, will call DoStop() first
    virtual ~BehaviorBase() {
SetAutoDelete(false);
```

```
if(started)
    DoStop();
//{ if(started) { references++; DoStop(); references--; } }
    }

    //! By default, merely adds to the reference counter (through AddReference())
    //@note you should still call this from your overriding methods
    virtual void DoStart() {
//std::cout << getName() << " started " << this << std::endl;
if(!started) {
    started=true;
    AddReference();
}
    }

    //! By default, subtracts from the reference counter, and deletes if zero @note
             you should still call this when you override this @warning call this at
             the end of your DoStop(), not beginning (it might @c delete @c this )
    virtual void DoStop() {
//std::cout << getName() << " stopped " << this << std::endl;
if(started) {
    started=false;
    RemoveReference();
}
    }

    //! By defining here, allows you to get away with not supplying a
        processEvent() function for the EventListener interface.  By default, does nothing.
    virtual void processEvent(const EventBase& /*event*/) {};

    //! Identifies the behavior in menus and such
    virtual std::string getName() const =0;

    //! Gives a short description of what this class of behaviors does...
          you should override this (but don't have to)
    static std::string getClassDescription() { return ""; }

    //! Gives a short description of what this particular instantiation does
        (in case a more specific description is needed)
    virtual std::string getDescription() const { return getClassDescription(); }

    //! Returns true if the behavior is currently running
    virtual bool isActive() const { return started; }

protected:
    bool started; //!< true when the behavior is active
};

/*! @file
 * @brief Defines BehaviorBase from which all Behaviors should inherit
 * @author ejt (Creator)
 *
 * $Author: ejt $
```

```
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.9 $
 * $State: Rel $
 * $Date: 2003/10/10 15:54:04 $
 */

#endif
```

D.2 Behaviors/Demos/AutoGetupBehavior.h

```
//-*-c++-*-
#ifndef INCLUDED_AutoGetupBehavior_h_
#define INCLUDED_AutoGetupBehavior_h_

#include "Behaviors/BehaviorBase.h"
#include "Shared/WorldState.h"
#include "Events/EventRouter.h"
#include "Shared/SharedObject.h"
#include "Motion/MotionManager.h"
#include "Motion/MotionSequenceMC.h"
#include "Shared/Config.h"
#include "SoundPlay/SoundManager.h"

//! a little background behavior to keep the robot on its feet
class AutoGetupBehavior : public BehaviorBase {
public:
    //! constructor
    AutoGetupBehavior() : BehaviorBase(), back(0), side(0), gamma(.9),
       sensitivity(.85*.85), waiting(false) {}
    //! destructor
    virtual ~AutoGetupBehavior() {}

    //! Listens for the SensorSourceID::UpdatedSID
    virtual void DoStart() {
BehaviorBase::DoStart();
erouter->addListener(this,EventBase::sensorEGID,SensorSourceID::UpdatedSID);
    }
    //! Stops listening for events
    virtual void DoStop() {
erouter->forgetListener(this);
BehaviorBase::DoStop();
    }
    //! Run appropriate motion script if the robot falls over
    virtual void processEvent(const EventBase &event) {
if(event.getGeneratorID()==EventBase::motmanEGID) {
    //previous attempt at getting up has completed
    cout << "Getup complete" << endl;
    erouter->removeListener(this,EventBase::motmanEGID);
    waiting=false;
    return;
}
back=back*gamma+(1-gamma)*state->sensors[BAccelOffset];
side=side*gamma+(1-gamma)*state->sensors[LAccelOffset];
```

```
if(!waiting && back*back+side*side>sensitivity*WorldState::g*WorldState::g) {
    //fallen down
    cout << "I've fallen!" << endl;
    sndman->PlayFile("yipper.wav");
    std::string gu;
    //config->motion.makePath will return a path relative to config->motion.root
                    (from config file read at boot)
    if(fabs(back)<fabs(side))
gu=config->motion.makePath("gu_side.mot");
    else if(back<0)
gu=config->motion.makePath("gu_back.mot");
    else
gu=config->motion.makePath("gu_front.mot");
    SharedObject< MotionSequenceMC<MotionSequence::SizeMedium> > getup(gu.c_str());
    MotionManager::MC_ID id=motman->addMotion(getup,MotionManager::kHighPriority);
    erouter->addListener(this,EventBase::motmanEGID,id,EventBase::deactivateETID);
    waiting=true;
}
    }
    virtual std::string getName() const { return "AutoGetupBehavior"; }
    static std::string getClassDescription() { return "Monitors gravity's influence
       on the accelerometers - if it seems the robot has fallen over, it runs appropriate
       getup script"; }

protected:
    float back;          //!< exponential average of backwards accel
    float side;          //!< exponential average of sideways accel
    float gamma;         //!< default 0.9, gamma parameter for exponential average of above
    float sensitivity;   //!< default 0.85*0.85, squared threshold to consider having
                                                    fallen over, use values 0-1
    bool  waiting;       //!< true while we're waiting to hear from completion of
                                MotionSequence, won't try again until this is cleared
};

/*! @file
 * @brief Defines AutoGetupBehavior, a little background behavior to keep the robot on its feet
 * @author ejt (Creator)
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.9 $
 * $State: Rel $
 * $Date: 2003/09/25 15:26:22 $
 */

#endif
```

D.3 Motion/MotionManager.h

```
//-*-c++-*-
#ifndef INCLUDED_MotionManager_h
#define INCLUDED_MotionManager_h
```

```
#include "MotionCommand.h"
#include "OutputCmd.h"
#include "OutputPID.h"
#include "Shared/RobotInfo.h"
#include "Shared/ListMemBuf.h"
#include "Shared/MutexLock.h"

#ifdef PLATFORM_APERIOS
#include "Shared/SharedObject.h"
#include "MotionManagerMsg.h"
#endif

#ifdef PLATFORM_APERIOS
#include <OPENR/OPENR.h>
#include <OPENR/OPENRAPI.h>
#include <OPENR/OSubject.h>
#include <OPENR/ObjcommEvent.h>
#include <OPENR/OObject.h>
#endif

//! The purpose of this class is to serialize access to the MotionCommands
     and simplify their sharing between memory spaces
/*! Since MotionObject and MainObject run as separate processes, they could potentially try to acc
 *  same motion command at the same time, leading to unpredictable behavior.  The MotionManager en
 *  a set of locks to solve this\n
 *  The other problem is that we are sharing between processes.  MotionManager will do what's nece
 *  MotionCommand's to all the processes (currently just MainObj and MotoObj)\n
 *  You should be able to create and add a new motion in one line:
 *  @code
 *  motman->addMotion(SharedObject<YourMC>([arg1,[arg2,...]]) [, priority [, autoprune] ]);
 *  @endcode
 *  But if you want to do some more initializations not handled by the constructor (the @p arg1, @
 *  params) then you would want to do something like the following:
 *  @code
 *  SharedObject<YourMC> tmpvar([arg1,[arg2,...]]);
 *  tmpvar->cmd1();
 *  tmpvar->cmd2();
 *  //...
 *  motman->addMotion(tmpvar [, ...]);
 *  @endcode
 *
 *  Notice that tmpvar is of type SharedObject, but you're calling YourMC functions on it...
 *  SharedObject is a "smart pointer" which will pass your function calls on to the
 *  underlying templated type.  Isn't C++ great? :)
 *
 *  @warning Once the MotionCommand has been added you must check it
 *  out to modify it or risk concurrent access problems.
 *
 *  @see MotionCommand for information on creating new motion primitives.
 *
 *  @see MMAccessor for information on accessing motions after you've added them to MotionManager.
 */
class MotionManager {
```

```
public:
    //! This is the number of processes which will be accessing the MotionManager
    /*! Probably just MainObject and MotionObject
     *  Isn't really a hard maximum, but should be actual expected, need to know when
       they're all connected */
    static const unsigned int MAX_ACCESS=2;

    static const unsigned int MAX_MOTIONS=64;   //!< This is the maximum number of
        Motions which can be managed, can probably be increased reasonably without trouble

    typedef MotionManagerMsg::MC_ID MC_ID;      //!< use this type when referring to
                          the ID numbers that MotionManager hands out
    static const MC_ID invalid_MC_ID=MotionManagerMsg::invalid_MC_ID; //!< for errors
             and undefined stuff

    MotionManager();                            //!< Constructor, sets all the outputs to 0

#ifdef PLATFORM_APERIOS
    void InitAccess(OSubject* subj);            //!< @b LOCKS @b MotionManager
            Everyone who is planning to use the MotionManager needs to call this before
            they access it or suffer a horrible fate
    void receivedMsg(const ONotifyEvent& event); //!< @b LOCKS @b MotionManager
        This gets called by an OObject when it receives a message from one of the other
        OObject's MotionManagerComm Subject
#endif

    //!@name MotionCommand Safe
    void setOutput(const MotionCommand* caller, unsigned int output, const OutputCmd& cmd)
       //!< @b LOCKS @b MotionManager Requests a value be set for the specified output,
       copies cmd across frames
    void setOutput(const MotionCommand* caller, unsigned int output, const OutputCmd& cmd,
 unsigned int frame); //!< @b LOCKS @b MotionManager Requests a value be set for the
             specified output in the specified frame
    void setOutput(const MotionCommand* caller, unsigned int output, const OutputCmd
        cmd[NumFrames]); //!< @b LOCKS @b MotionManager Requests a value be set for the
              specified output across frames
    void setOutput(const MotionCommand* caller, unsigned int output, const OutputPID& pid);
        //!< @b LOCKS @b MotionManager Requests a PID be set for the specified output,
            notice that this might be overruled by a higher priority motion
    void setOutput(const MotionCommand* caller, unsigned int output, const OutputCmd& cmd,
        const OutputPID& pid); //!< @b LOCKS @b MotionManager Requests a value and PID be
         set for the specified output
    void setOutput(const MotionCommand* caller, unsigned int output, const OutputCmd
      cmd[NumFrames], const OutputPID& pid); //!< @b LOCKS @b MotionManager Requests a
        value and PID be set for the specified output
    const OutputCmd& getOutputCmd(unsigned int output) const { return cmds[output]; }
       //!< Returns the value of the output last sent to the OS.  Note that this will
         differ from the sensed value in state, even when staying still.  There is no
          corresponding getOutputPID because this value *will* duplicate the value in state.
    void setPriority(MC_ID mcid, float p) { cmdlist[mcid].priority=p; }//!< sets the
             priority level of a MotionCommand
    float getPriority(MC_ID mcid) const { return cmdlist[mcid].priority; } //!< returns
             priority level of a MotionCommand
```

```
    //@}

    //@{
    inline MC_ID begin() const         { return skip_ahead(cmdlist.begin()); }
         //!< returns the MC_ID of the first MotionCommand
    inline MC_ID next(MC_ID cur) const { return skip_ahead(cmdlist.next(cur)); }
         //!< returns the MC_ID of MotionCommand following the one that is passed
    inline MC_ID end() const           { return cmdlist.end();      }
         //!< returns the MC_ID of the one-past-the-end MotionCommand (like the STL)
    inline unsigned int size() const   { return cmdlist.size();     }
        //!< returns the number of MotionCommands being managed
    //@}

    //!You can have one MC check out and modify another, but make sure the other
           MC doesn't call setOutput()
    //!@name MotionCommand "Risky"
    MotionCommand * checkoutMotion(MC_ID mcid,bool block=true); //!< locks the command
             and possibly performs RTTI conversion; supports recursive calls
    void checkinMotion(MC_ID mcid); //!< marks a MotionCommand as unused
    MotionCommand * peekMotion(MC_ID mcid) { return mcid==invalid_MC_ID?NULL:
     cmdlist[mcid].baseaddrs[_MMaccID]; } //!< allows access to a MotionCommand without
        checking it out @warning @b never call a function based on this, only access
         member fields through it
    unsigned int checkoutLevel(MC_ID mcid) { return mcid==invalid_MC_ID?0:cmdlist[mcid]
        .lock.get_lock_level(); } //!< returns the number of times @a mcid has been
               checked out minus the times it's been checked in
    bool isOwner(MC_ID mcid) { return mcid==invalid_MC_ID?false:(cmdlist[mcid].lock.owner()==_MMac
    //@}

    //!@name MotionCommand Unsafe
    //@{
#ifdef PLATFORM_APERIOS
    MC_ID addMotion(const SharedObjectBase& sm); //!< @b LOCKS @b MotionManager Creates a
        new MotionCommand, automatically sharing it between processes (there is some lag
        time here)
    MC_ID addMotion(const SharedObjectBase& sm, float priority); //!< @b LOCKS @b
        MotionManager allows a quick way to set a priority level of a new MotionCommand
    MC_ID addMotion(const SharedObjectBase& sm, bool autoprune); //!< @b LOCKS @b
        MotionManager allows a quick was to set the autoprune flag
    MC_ID addMotion(const SharedObjectBase& sm, float priority, bool autoprune);
       //!< @b LOCKS @b MotionManager Call one of these to add a MotionCommand to the
           MotionManager, using the SharedObject class
#endif //PLATFORM_APERIOS
    void removeMotion(MC_ID mcid); //!< @b LOCKS @b MotionManager removes the specified
       MotionCommand
    //@}

    //@{
    void lock()    { MMlock.lock(_MMaccID); } //!< gets an exclusive lock on MotionManager
        - functions marked @b LOCKS @b MotionManager will cause (and require) this to happen autom
    bool trylock() { return MMlock.try_lock(_MMaccID); } //!< tries to get a lock without blocking
    void release() { MMlock.release(); } //!< releases a lock on the motion manager
    //@}
```

```
    //@{
    void getOutputs(float outputs[NumFrames][NumOutputs]);  //!< @b LOCKS @b MotionManager
         called by MotionObject to fill in the output values for the next ::NumFrames
         frames (only MotoObj should call this...)
    void updateWorldState();                                //!< call this when you want
       MotionManager to set the WorldState to reflect what things should be for unsensed
         outputs (LEDs, ears) (only MotoObj should be calling this...)
#ifdef PLATFORM_APERIOS
    bool updatePIDs(OPrimitiveID primIDs[NumOutputs]);      //!< call this when you want
        MotionManager to update modified PID values, returns true if changes made
        (only MotoObj should be calling this...)
#endif
    //@}

    //! holds the full requested value of an output
class OutputState {
public:
    //!@name Constructors
    //!Constructor
    OutputState();
    OutputState(unsigned int out, float pri, MC_ID mc, const OutputCmd cmds[NumFrames]);
    OutputState(unsigned int out, float pri, MC_ID mc, const OutputCmd& cmd);
    OutputState(unsigned int out, float pri, MC_ID mc, const OutputCmd& cmd, unsigned int frame);
    OutputState(unsigned int out, float pri, MC_ID mc, const OutputPID& p);
    OutputState(unsigned int out, float pri, MC_ID mc, const OutputCmd cmds[NumFrames], const OutputPID&
    OutputState(unsigned int out, float pri, MC_ID mc, const OutputCmd& cmd, const OutputPID& p);
    //@}
    float priority;             //!< priority level
    MC_ID mcid;                 //!< MC_ID of requester
    OutputCmd frames[NumFrames]; //!< values of output planned ahead
    OutputPID pid;               //!< pid of output
};

    //!Just to give you a general idea what values to use for different priority levels
    //!@name Priority Level Constants
    static const float kIgnoredPriority;    //!< won't be expressed, handy if you want
                to temporarily pause something
    static const float kBackgroundPriority; //!< will only be expressed if *nothing*
              else is using that joint
    static const float kLowPriority;        //!< for stuff that's not background
         but lower than standard
    static const float kStdPriority;        //!< for every-day commands
    static const float kHighPriority;       //!< for stuff that should over ride standard stuff
    static const float kEmergencyPriority;  //!< for really important stuff, such as the emergency stop

protected:
    //! used to request pids for a given joint
    struct PIDUpdate {
        //!constructor
        PIDUpdate() : joint((unsigned int)-1) {}
        //!constructor
        PIDUpdate(unsigned int j, const float p[3]) : joint(j) {
```

```
    for(unsigned int i=0; i<3; i++)
        pids[i]=p[i];
}
        unsigned int joint; //!< the joint ID (see RobotInfo.h for offset values)
        float pids[3]; //!< the PID values to use (see ::Pid )
    };
    ListMemBuf<PIDUpdate,NumPIDJoints> pidchanges;  //!< stores PID updates, up to one
        per joint (if same is set more than once, it's just overwrites previous update)
    void setPID(unsigned int j, const float p[3]); //!< @b LOCKS @b MotionManager you can
       call this to set the PID values directly (instead of using a motion command) Be careful tho

    typedef unsigned short accID_t; //!< type to use to refer to accessors of
             MotionManager (or its locks)

    void func_begin() { MMlock.lock(_MMaccID); } //!< called at the begining of many
            functions to lock MotionManager
    void func_end() { MMlock.release(); } //!< called at the end of a function which
       called func_begin() to release it
    template<class T> T func_end(T val) { func_end(); return val; } //!< same as func_end()
             , except passes return value through

    MC_ID skip_ahead(MC_ID mcid) const; //!< during iteration, skips over motioncommands
           which are still in transit from on OObject to another

    //!All the information we need to maintain about a MotionCommand
    struct CommandEntry {
        //! Constructor, sets everything to basics
        CommandEntry() : lastAccessor((unsigned short)-1),lock(),priority(MotionManager::kStdPrior
    for(unsigned int i=0; i<MAX_ACCESS; i++) {
        baseaddrs[i]=NULL;
#ifdef PLATFORM_APERIOS
  rcr[i]=NULL;
#endif
    }
}
        MotionCommand * baseaddrs[MAX_ACCESS]; //!< for each accessor, the base address of the mot
#ifdef PLATFORM_APERIOS
        RCRegion * rcr[MAX_ACCESS];            //!< for each accessor the shared memory
              region that holds the motion command
#endif
        accID_t lastAccessor;                  //!< the ID of the last accessor to touch
           the command (which implies if it wants to touch this again, we don't have to convert ag
        MutexLock<MAX_ACCESS> lock;            //!< a lock to maintain mutual exclusion
        float priority;                        //!< MotionCommand's priority level
private:
      CommandEntry(const CommandEntry&); //!< this shouldn't be called...
      CommandEntry& operator=(const CommandEntry&); //!< this shouldn't be called...
    };
    ListMemBuf<CommandEntry,MAX_MOTIONS,MC_ID> cmdlist;     //!< the list where
          MotionCommands are stored, remember, we're in a shared memory region with
            different base addresses - no pointers!
    MC_ID cur_cmd; //!< MC_ID of the MotionCommand currently being updated by getOutputs(),
      or NULL if not in getOutputs.  This is used by the setOutput()'s to tell which MotionCommand
```

```
    inline MC_ID pop_free() { return cmdlist.new_front(); } //!<pulls an entry from
        cmdlist's free section and returns its index
    inline void push_free(MC_ID a) { cmdlist.erase(a); }    //!<puts an entry back into
        cmdlist's free section

    MutexLock<MAX_ACCESS> MMlock;          //!< The main lock for the class

    typedef ListMemBuf<OutputState,MAX_MOTIONS> cmdstatelist_t; //!< shorthand for a list of OutputState
    cmdstatelist_t cmdstates[NumOutputs];  //!< requested positions by each of the MC's
         for each of the outputs
    float cmdSums[NumOutputs];             //!<Holds the final values for the outputs of
          the last frame generated
    OutputCmd cmds[NumOutputs];            //!<Holds the weighted values and total
          weight for the outputs of the last frame

#ifdef PLATFORM_APERIOS
    accID_t numAcc;                        //!<The number of accessors who have
        registered with InitAccess()
    OSubject* subjs[MAX_ACCESS];           //!<The OSubject for each process (accessor)
        on which it should be broadcast when a command is added
#endif

    static int _MMaccID;          //!<Stores the accessor for the current process

private:
    MotionManager(const MotionManager&); //!< this shouldn't be called...
    MotionManager& operator=(const MotionManager&); //!< this shouldn't be called...
};

//!anyone who #includes MotionManager.h will be wanting to use the global motman...
  don't want multiple of these! created by MotoObj
extern MotionManager * motman;

/*! @file
 * @brief Describes MotionManager, simplifies sharing of MotionCommand's and provides
   mutual exclusion to their access
 * @author ejt (Creator)
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.15 $
 * $State: Rel $
 * $Date: 2003/09/12 21:38:47 $
 */

#endif
```

D.4 Motion/MMAccessor.h

```
#ifndef INCLUDED_MMAccessor_h_
#define INCLUDED_MMAccessor_h_
```

```
#include "MotionManager.h"

//! This class allows convenient ways of accessing a motion command
/*! Since MotionCommands must be checked out of the motion manager and then checked back
 *  in when they are done, this is a common source of errors, leading to deadlock.  This class
 *  will check the motion out when it's created, and check it back in when it goes out of scope\n
 *  It supports recursive checkin/checkouts. \n
 *  Uses global ::motman
 *
 *  So, instead of doing things like this:
 *  @code
 *  YourMotionCommand* ymc = dynamic_cast<YourMotionCommand*>(motman->checkoutMotion(your_mc_id));
 *  //do 'stuff' with ymc, e.g.: ymc->rollOver();
 *  motman->checkinMotion(your_mc_id);
 *  @endcode
 *  ...which can be error prone in many regards - if 'stuff' returns without checking in, or you
 *  forget to check in, or you get lazy and leave it checked out longer than you should, which can
 *  cause general performance issues (or worse, deadlock)
 *  Using MMAccessor makes it much easier to solve these problems, and is easier to code:
 *  @code
 *  MMAccessor<YourMotionCommand> mma(your_mc_id);
 *  //do 'stuff' with mma, e.g.: mma->rollOver();
 *  @endcode
 *  We can call a return at any point and the motion command will automatically be checked in, and
 *  since C++ guarrantees that the destructor of mma will be called, we don't have to worry about
 *  forgetting about it.  We can limit the scope by placing {}'s around the segment in question:
 *  @code
 *  //pre-stuff
 *  {
 *    MMAccessor<YourMotionCommand> mma(your_mc_id);
 *    //do 'stuff' with mma, e.g.: mma->rollOver();
 *  }
 *  //post-stuff - has no knowledge of mma, out of its scope
 *  @endcode
 *  And, for those astute enough to notice that the theoretical @a rollOver() function is called o
 *  MMAccessor when it's actually a member of YourMotionCommand, this is because MMAccessor behave
 *  'smart pointer', which overloads operator->() so it is fairly transparent to use.
 *
 *  See also the templated checkin(Ret_t ret) function for more examples of streamlined usage.
 *
 *  MMAccessor is a small class, you may consider passing it around instead of a MotionManager::MC
 *  if appropriate.  (Would be appropriate to avoid multiple checkin/outs in a row from different
 *  functions, but not as appropriate for storage and reuse of the same MMAccessor.
 */
template<class MC_t>
class MMAccessor {
 public:

   //! constructor, checks out by default
   /*! @param id the motion command to check out
    *  @param ckout if true (default) will checkout upon creation.  otherwise it just
       gets current address (so you can peek at member fields, which should be safe) */
```

```
    MMAccessor(MotionManager::MC_ID id,bool ckout=true) : mc_id(id), mcptr(NULL) {
if(ckout)
    checkout();
else
    mcptr=(MC_t*)motman->peekMotion(id);
    }

    //! constructor, allows objects to provide uniform access to MotionCommands,
        regardless of whether they are currently in the MotionManager
    MMAccessor(MotionCommand * ptr) : mc_id(invalid_MC_ID), mcptr(ptr) {}

    //! copy constructor - will reference the same mc_id - checkin/checkouts are
      independent between this and @a a - motman does counting of checkin/checkouts
    MMAccessor(const MMAccessor& a) : mc_id(a.mc_id), mcptr(a.mcptr) {
if(motman->isOwner(mc_id))
    checkout();
    }

    //! destructor, checks in if needed
    ~MMAccessor() {
checkin();
    }

    //! allows assignment of MMAccessor's, similar to the copy constructor - the two
           MMAccessor's will control the same MotionCommand
    MMAccessor<MC_t> operator=(const MMAccessor<MC_t>& a) {
mc_id=a.mc_id;
mcptr=a.mcptr;
if(motman->isOwner(mc_id))
    checkout();
return *this;
    }

    //! So you can check out if not done by default (or you checked in already)
    inline MC_t* checkout() {
if(mc_id!=MotionManager::invalid_MC_ID)
    mcptr=dynamic_cast<MC_t*>(motman->checkoutMotion(mc_id));
return mcptr;
    }

    //! Returns the motion command's address so you can call functions
    inline MC_t* mc() const { return mcptr; }

    //! Checks in the motion
    /*! Don't forget, you can also just limit the scope using extra { }'s */
    inline void checkin() {
if(motman->isOwner(mc_id)) {
    motman->checkinMotion(mc_id);
    mcptr=NULL; // fail fast if we use it after checkin
}
    }

    //! Checks in the motion, passing through the value it is passed.
```

```
    /*! @return the same value it's passed
     *
     *  Useful in situations like this:
     *  @code
     *  MMAccessor<myMC> mine(myMC_id);
     *  if(mine.mc()->foo())
     *    //do something with motman here
     *  @endcode
     *  But we want to check in @a mine ASAP - if we don't reference it
     *  anywhere in the if statement, we're leaving the MC locked longer
     *  than we need to.  How about instead doing this:
     *  @code
     *  bool cond;
     *  {MMAccessor<myMC> mine(myMC_id); cond=mine.mc()->foo();}
     *  if(cond)
     *    //do something with motman here
     *  @endcode
     *  But that uses an extra variable... ewwww... so use this function
     *  as a pass through to checkin the MC:
     *  @code
     *  MMAccessor<myMC> mine(myMC_id);
     *  if(mine.checkin(mine.mc()->foo()))
     *    //do something with motman here
     *  @endcode*/
    template<class Ret_t> Ret_t checkin(Ret_t ret) {
checkin();
return ret;
    }

    MC_t* operator->() { return mc(); } //!< smart pointer to the underlying MotionCommand
    const MC_t* operator->() const { return mc(); } //!< smart pointer to the underlying MotionCom
    MC_t& operator*() { return *mc(); } //!< smart pointer to the underlying MotionCommand
    const MC_t& operator*() const { return *mc(); } //!< smart pointer to the underlying MotionCom
    MC_t& operator[](int i) { return mc()[i]; } //!< smart pointer to the underlying MotionCommand
    const MC_t& operator[](int i) const { return mc()[i]; } //!< smart pointer to the underlying M

 protected:
    MotionManager::MC_ID mc_id; //!< the MC_ID that this Accessor was constructed with
    MC_t* mcptr;                //!< a pointer to the motion command, should always be valid even
};

/*! @file
 * @brief Defines MMAccessor, allows convenient ways to check MotionCommands in and out
     of the MotionManager
 * @author ejt (Creator)
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.8 $
 * $State: Rel $
 * $Date: 2003/09/12 21:38:46 $
 */
```

```
#endif
```

D.5 Motion/WalkMC.h

```
//-*-c++-*-

//This class is ported from Carnegie Mellon's 2001 Robosoccer entry, and falls under their license:
/*=============================================================================
    CMPack'02 Source Code Release for OPEN-R SDK v1.0
    Copyright (C) 2002 Multirobot Lab [Project Head: Manuela Veloso]
    School of Computer Science, Carnegie Mellon University
  -----------------------------------------------------------------------------
    This software is distributed under the GNU General Public License,
    version 2.  If you do not have a copy of this licence, visit
    www.gnu.org, or write: Free Software Foundation, 59 Temple Place,
    Suite 330 Boston, MA 02111-1307 USA.  This program is distributed
    in the hope that it will be useful, but WITHOUT ANY WARRANTY,
    including MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
  -----------------------------------------------------------------------------
    Additionally licensed to Sony Corporation under the following terms:

    This software is provided by the copyright holders AS IS and any
    express or implied warranties, including, but not limited to, the
    implied warranties of merchantability and fitness for a particular
    purpose are disclaimed.  In no event shall authors be liable for
    any direct, indirect, incidental, special, exemplary, or consequential
    damages (including, but not limited to, procurement of substitute
    goods or services; loss of use, data, or profits; or business
    interruption) however caused and on any theory of liability, whether
    in contract, strict liability, or tort (including negligence or
    otherwise) arising in any way out of the use of this software, even if
    advised of the possibility of such damage.
  =============================================================================
*/

#ifndef INCLUDED_WalkMC_h
#define INCLUDED_WalkMC_h

#include "MotionCommand.h"
#include "Geometry.h"
#include "Kinematics.h"
#include "Path.h"
#include "Shared/get_time.h"
#include "OutputCmd.h"

//! A nice walking class from Carnegie Mellon University's 2001 Robosoccer team,
     modified to fit this framework, see their <a href="../CMPack_license.txt">license</a>
/*! Moves the feet through a looping path in order to walk - default parameters use
 *  a walk low to the ground so you don't walk over the ball.
 *
 *  This portion of the code falls under CMPack's license:
 *  @verbinclude CMPack_license.txt
 *
```

```
 *  @bug the legs try (briefly) to straighten out when first starting to move
 */
class WalkMC : public MotionCommand {
public:
    typedef SplinePath<vector3d,double> splinepath; //!<for convenience
    typedef HermiteSplineSegment<vector3d,double> spline; //!<for convenience

    //! holds state about each leg's path
    struct LegWalkState {
LegWalkState() : airpath(), air(0) {} //!< constructor
spline airpath; //!< the path to follow
bool air; //!< true if in the air
    };

    //! holds parameters about how to move each leg
    struct LegParam {
LegParam() : neutral(), lift_vel(), down_vel(), lift_time(0), down_time(0) {}
           //!< constructor
vector3d neutral; //!< defines the "neutral" point of each leg - where it is in midstep
vector3d lift_vel; //!< give the velocities to use when raising the paw
vector3d down_vel; //!< give the velocities to use when lowering the paw
double lift_time; //!< the time (as percentage of WalkParam::period) in the
           cycle to lift (so you can set different offsets between the paws)
double down_time; //!< the time (as percentage of WalkParam::period) in the cycle
         to put down (so you can set different offsets between the paws)
    };

    //! holds more general parameters about the walk
    struct WalkParam {
WalkParam() : body_height(0), body_angle(0), hop(0), sway(0), period(0),
  reserved() {} //!< constructor
LegParam leg[4]; //!< a set of LegParam's, one for each leg
double body_height; //!< the height to hold the body (mm)
double body_angle; //!< the angle to hold the body (rad - 0 is level)
double hop;  //!< sinusoidal hop amplitude
double sway; //!< sinusoidal sway in y direction
long period; //!< the time between steps
long reserved; //!< live with it
    };

    //! constructor
    WalkMC(const char* pfile=NULL);

    virtual void DoStart(); //!< sends an activate LocomotionEvent
    virtual void DoStop();  //!< sends a deactivate LocomotionEvent

    virtual int updateOutputs(); //!< calculates current positions of the paws

    //! Returns true if we are walking. This modified isDirty allows the AIBO to slow
     down to a stop rather than stopping immediately.
    virtual int isDirty();

    //! always true - never autoprunes
```

```
    virtual int isAlive() { return true; }

    //! loads parameters from a file (@todo use LoadSave)
    void load(const char* pfile);
    //! saves parameters to a file (@todo use LoadSave)
    void save(const char* pfile) const;
    //! set the direction to walk - can specify x (forward), y (left), and angular
        (counterclockwise) velocities
    void setTargetVelocity(double dx,double dy,double da);
    //! returns current velocity we're trying to go
    const vector3d& getTargetVelocity() { return target_vel_xya; }
    //! returns the velocity we're actually moving (subject to clipping at max_accel_xya),
          doesn't reflect value of getPaused()...
    const vector3d& getCurVelocity() const { return vel_xya;}
    //! returns the time we've been traveling along the current vector
    unsigned int getTravelTime() { return get_time()-travelTime; }

    void setPaused(bool p) { isPaused=p; } //!< if set to true, will stop moving
    bool getPaused() const { return isPaused; } //!< if is true, we aren't moving
    void setHeight(double h) { wp.body_height=h; } //!< sets WalkParam::body_height of #wp
    double getHeight() { return wp.body_height; } //!< gets WalkParam::body_height of #wp
    void setAngle(double a) { wp.body_angle=a; } //!< sets WalkParam::body_angle of #wp
    double getAngle() { return wp.body_angle; } //!< gets WalkParam::body_angle of #wp
    void setHop(double h) { wp.hop=h; } //!< sets WalkParam::hop of #wp
    double getHop() { return wp.hop; } //!< gets WalkParam::hop of #wp
    void setSway(double h) { wp.sway=h; } //!< sets WalkParam::sway of #wp
    double getSway() { return wp.sway; } //!< gets WalkParam::sway of #wp
    void setPeriod(long p) { wp.period=p; } //!< sets WalkParam::period of #wp
    long getPeriod() { return wp.period; } //!< gets WalkParam::period of #wp
    void setSlowMo(float p) { slowmo=p; } //!< sets slowmo
    float* getSlowMo() { return &slowmo; } //!< gets slowmo

    WalkParam & getWP() { return wp; }; //!< returns the current walk parameter structure

    //! takes current leg positions from WorldState and tries to match the point in the
        cycle most like it
    void resetLegPos();

    static const float MAX_DX; //!< ==180 mm/sec
    static const float MAX_DY; //!< ==140 mm/sec
    static const float MAX_DA; //!< ==1.8 rad/sec
    // tss "SmoothWalk" modification follows (actually only comment is changed)
    // static const vector3d max_accel_xya; //!< vector version of MAX_DX,MAX_DY,MAX_DA
    static const vector3d max_accel_xya; //!< maximum acceleration of x, y, and a velocity

protected:
    //! holds current joint commands
    OutputCmd cmds[NumOutputs][NumFrames];

protected:
    //! does some setup stuff, calls load(pfile)
    void init(const char* pfile);
```

```
    bool isPaused; //!< true if we are paused

    WalkParam wp; //!< current walking parameters (note that it's not static - different
        WalkMC's can have different setting, handy...
    LegWalkState legw[NumLegs]; //!< current state of each leg
    vector3d legpos[NumLegs]; //!< current position of each leg
    splinepath body_loc; //!< the path the body goes through while walking (?)
    splinepath body_angle; //!< the path the body goes through while walking (?)

    vector3d pos_delta; //!< how much we've moved
    double angle_delta; //!< how much we've turned

    unsigned int travelTime; //!< the time since the last call to setTargetVelocity -
        handy to check the time we've been traveling current vector
    int time; //!< time of last call to updateJointCmds() (scaled by slowmo)
    int TimeStep; //!< time to pretend passes between each call to updateJointCmds() -
        usually RobotInfo::FrameTime
    float slowmo; //!< scales time values to make the walk move in slow motion for analysis (or fa

    // tss "SmoothWalk" addition follows
    /*! The CycleOffset variable is used to ensure that each time the AIBO
     *  starts walking, it starts at the same point in the walk cycle as
     *  where it stopped. This measure is intended to decrease the amount
     *  of jerking (and hence deviation) that occurs when the AIBO starts
     *  walking forward and then suddenly stops. */
    int CycleOffset;
    /*! Each CycleOffset corresponds to a different TimeOffset once the
     *  robot starts walking again. Consider this example: the robot
     *  stops 2/3 of the way through the cycle, then starts again 1/3
     *  of the way through the cycle on the absolute clock. The time
     *  offset to advance the clock to the right part of the cycle is
     *  1/3 of a cycle, so we set TimeOffset to 1/3 cycle and add that to
     *  every clock value used in the walk code. */
    int TimeOffset;
    /*! Every time we stop, we know we'll have a new CycleOffset, and we'll
     *  need to compute a new TimeOffset. This boolean says as much. */
    bool NewCycleOffset;
    // tss "SmoothWalk" addition ends

    vector3d vel_xya; //!< the current velocity we're moving
    vector3d target_vel_xya; //!< the velocity that was requested
};

/* struct LegState{
   long attr,reserved;
   point3d pos;
   double angles[3];
   };

   struct HeadState{
   long attr,reserved;
   vector3d target;
   double angles[3];
```

```
   };

   struct BodyState{
   BodyPosition pos;
   LegState leg[4];
   HeadState head;
   };
*/

/*! @file
 * @brief Describes WalkMC, a MotionCommand for walking around
 * @author CMU RoboSoccer 2001-2002 (Creator)
 * @author ejt (ported)
 * @author PA Gov. School for the Sciences 2003 Team Project - Haoqian Chen,
          Yantian Martin, Jon Stahlman (modifications)
 *
 * @verbinclude CMPack_license.txt
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.14 $
 * $State: Rel $
 * $Date: 2003/10/07 01:00:40 $
 */

#endif
```

D.6 Motion/HeadPointerMC.h

```
//-*-c++-*-
#ifndef INCLUDED_HeadPointerMC_h
#define INCLUDED_HeadPointerMC_h

#include "MotionCommand.h"
#include "OutputCmd.h"
#include "Shared/RobotInfo.h"

//! This class gives some quick and easy functions to point the head at things
class HeadPointerMC : public MotionCommand {
public:
    //! constructor, defaults to active, BodyRelative, all joints at 0
    HeadPointerMC();
    //! destructor
    virtual ~HeadPointerMC() {}

    //! Various modes the head can be in.  In the future may want to add ability to
          explicitly track an object or point in the world model
    enum CoordFrame_t {
BodyRelative,    //!<holds neck at a specified position, like a PostureEngine, but neck specific
GravityRelative  //!<uses accelerometers to keep a level head, doesn't
         apply for pan joint, but in future could use localization for pan
    };
```

```
    void   setWeight(double w);   //!< sets the weight values for all the neck joints
    inline void   setWeight(RobotInfo::TPROffset_t i, double weight) { dirty=true; headJoints[i].w
    inline void   setActive(bool a)    { active=a; } //!< sets #active flag, see isDirty()
    inline bool   getActive() const{ return active; } //!< returns #active flag, see isDirty()

    void   noMaxSpeed() { for(unsigned int i=0; i<NumHeadJoints; i++) maxSpeed[i]=0; }
              //!< sets #maxSpeed to 0 (no maximum)
    void   defaultMaxSpeed(); //!< restores #maxSpeed to default settings from Config::Motion_Conf
    void   setMaxSpeed(TPROffset_t i, float x) { maxSpeed[i]=x*FrameTime/1000; }
            //!< sets #maxSpeed, pass it rad/sec
    float  getMaxSpeed(TPROffset_t i) { return maxSpeed[i]*1000/FrameTime; }
              //!< returns #maxSpeed in rad/sec

    //! converts a value @a v in @a srcmode to a value in @a tgtmode that would leave
      the joint angle for joint @a i constant (you probably won't need to call this directly)
    inline double convert(RobotInfo::TPROffset_t i, double v, CoordFrame_t srcmode,
          CoordFrame_t tgtmode) const {
return (srcmode==tgtmode)?v:convFromBodyRelative(i,convToBodyRelative(i, v, srcmode),tgtmode);
    }

    //!Note that none of these are @c virtual, so you don't have to checkout to use them,
         you @e should be able to use MotionManager::peekMotion()
    //!@name Joint Accessors
    void   setJoints(double tilt, double pan, double roll);
         //!< Directly sets the neck values, uses current mode
    void   setMode(CoordFrame_t m, bool convert=true); //!< sets all the joints to the
        given mode, will convert the values to the new mode if @a convert is true
    void   setJointMode(RobotInfo::TPROffset_t i, CoordFrame_t m, bool convert=true);
        //!< set a specific head joint's mode, will convert from previous mode's value to
         next mode's value if convert is true.  Pass one of RobotInfo::TPROffset_t, not a full off
    inline void   setJointValue(RobotInfo::TPROffset_t i, double value){ dirty=true;
       headValues[i]=value;} //!< set a specific head joint's value (for whatever mode
          it's in), pass one of RobotInfo::TPROffset_t, not a full offset!
    inline void   setJointValueAndMode(RobotInfo::TPROffset_t i, double value,
     CoordFrame_t m)  { dirty=true; headValues[i]=value; headModes[i]=m; }
      //!< set a specific head joint, pass one of RobotInfo::TPROffset_t, not a full offset!
    inline void   setJointValueFromMode(RobotInfo::TPROffset_t i, double value,
      CoordFrame_t m) { dirty=true; headValues[i]=convert(i,value,m,headModes[i]); }
       //!< set a specific head joint auto converting @a value from mode @a m to the
            current mode, pass one of RobotInfo::TPROffset_t, not a full offset!
    inline CoordFrame_t getJointMode(RobotInfo::TPROffset_t i) const{ return headModes[i];
       } //!< returns the current mode for joint @a i (use RobotInfo::TPROffset_t, not global offs
    inline double getJointValue(RobotInfo::TPROffset_t i) const{ return headValues[i]; }
       //!< returns the current value (relative to the current mode) of joint @a i, use
        getOutputCmd() if you want to know the actual @b target joint value (to get the
        actual current joint position, look in WorldState
    //@}

    //!@name Inherited:
    virtual int              updateOutputs( );     //!< Updates where the head is looking
    virtual const OutputCmd& getOutputCmd(unsigned int i);  //!< returns one of the
           #headJoints entries or ::unusedJoint if not a head joint
    virtual int isDirty(){ return (dirty && active)?1:0; } //!< true if a change has been
```

```
            made since the last updateJointCmds() and we're active
    virtual int              isAlive()                      { return true; }
    //@}

protected:
    double convToBodyRelative(TPROffset_t i, double v, CoordFrame_t mode) const;
        //!< converts to a body relative measurement for joint @a i
    double convFromBodyRelative(TPROffset_t i, double v, CoordFrame_t mode) const;
        //!< converts from a body relative measurement for joint @a i

    bool dirty;   //!< true if a change has been made since last call to updateJointCmds()
    bool active;                    //!< set by accessor functions, defaults to true
    OutputCmd headJoints[NumHeadJoints]; //!< stores the last values we sent from updateOutputs
    float headValues[NumHeadJoints]; //!< stores the target value of each joint, relative to #headModes
    CoordFrame_t headModes[NumHeadJoints]; //!< stores the current mode of each joint, for instance so t

    float maxSpeed[NumHeadJoints];//!< initialized from Config::motion_config, but can be overridden by
};

/*! @file
 * @brief Describes HeadPointerMC, a class for various ways to control where the head is looking
 * @author ejt (Creator)
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.7 $
 * $State: Rel $
 * $Date: 2003/09/07 22:14:01 $
 */

#endif
```

D.7 Motion/EmergencyStopMC.h

```
#ifndef INCLUDED_EmergencyStopMC_h
#define INCLUDED_EmergencyStopMC_h

#include "PostureMC.h"
#include "LedEngine.h"

//!overrides all joints with high priority freeze, blinks tail pink/red/blue cycle
/*! Uses MotionCommand::kEmergencyPriority.  Monitors the feedback on joints and
 *  adjusts joints to react to pressures above a certain threshold.  This allows
 *  you to mold the body while it's in this mode, while retaining enough stiffness
 *  to hold against gravity.
 *
 *  This MotionCommand is intended to always be left running.  It polls WorldState::state
 *  for a double-tap on the back button, which causes it to set its joint values.
 *  to whatever their current state is.  LEDs are left blank, except the tail,
 *  which is used to indicate that the emergency stop is on.
 *
 *  A second double-tap will cause it to set all joints to 0 weight
 *
```

```
 *  The tail LEDs only ever go up to .5, so that if you really care whether the tail
 *  light was set by an underlying behavior/motion, you should be able to tell by
 *  looking closely (if blue is going from .5 to 1, that's because it's already set)
 */
class EmergencyStopMC : public PostureMC {
 public:
EmergencyStopMC(); //!< constructor
virtual ~EmergencyStopMC() {} //!< destructor
virtual int updateOutputs(); //!< checks for feedback or double tap

virtual void takeSnapshot(); //!< records current positions of joints

void setActive(bool a); //!< allows you to modify #active
bool getActive() { return active; } //!< returns #active
void setStopped(bool p, bool sound=true); //!< allows you to modify #paused
bool getStopped() const { return paused; } //!< returns #paused
void setDblTapDuration(unsigned int d) { duration=d; } //!< sets #duration
unsigned int getDblTapDuration() const { return duration; } //!< returns #duration
void setResetSensitivity(float r) { pidcutoff=static_cast<unsigned char>(r*255); }
         //!< takes a value [0,1] to set #pidcutoff
float getResetSensitivity() { return pidcutoff/255.0; } //!< returns a value
                [0-1], corresponding to #pidcutoff

 protected:
void freezeJoints(); //!< code to execute when locking joints
void releaseJoints(); //!< code to execute when releasing joints

bool paused; //!< true if the joints are current locked up
bool stilldown; //!< true if the back button was down on last updateJointCmds
bool active; //!< true if the EmergencyStopMC is monitoring the back button
              (if false, won't pause on a double-tap)
unsigned int period; //!< period of cycles on tail LEDs
unsigned int timeoflastbtn; //!< time of the last button press
unsigned int timeofthisbtn; //!< time of the current button press
unsigned int timeoflastfreeze; //!< the time estop was last turned on
unsigned int duration; //!<the maximum time (in milliseconds) of consecutive
               button-down's to count as a double tap
unsigned char piddutyavgs[NumPIDJoints]; //!< a running average of PID feedback
          ("duty"), so one bad reading doesn't cause a movement, need a consistent pressure
unsigned char pidcutoff; //!<abs pid duty cycle above which we just reset joint to current
LedEngine ledengine; //!< used to do LED effects on the tail
};

/*! @file
 * @brief Describes EmergencyStopMC, overrides all joints, allows modelling, blinks tail
 * @author ejt (Creator)
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.8 $
 * $State: Rel $
 * $Date: 2003/09/07 22:14:01 $
 */
```

```
#endif
```

D.8 Shared/SharedObject.h

```
//-*-c++-*-
#ifndef INCLUDED_SharedObject_h
#define INCLUDED_SharedObject_h

#include <OPENR/RCRegion.h>

//! It's nice to have a parent class of SharedObject (which is what you probably want to
   be reading) so that you can pass around the data structure without worrying about what
     type is inside the shared memory region.
/*! See MotionManager for an example on how to use this. */
class SharedObjectBase {
public:
    void* data() const { return rcr->Base(); } //!< returns a pointer to the data region
    RCRegion * getRegion() const { return rcr; } //!< returns the OPEN-R memory region,
       should you need it

protected:
    SharedObjectBase() : rcr(NULL) {} //!< constructor, protected because you
     shouldn't need to create this directly, just a common interface to all templates
        of SharedObject

    //< destructor, automatically dereferences #rcr
    virtual ~SharedObjectBase() {
//std::cout << "~SharedObjectBase(): rcr->NumberOfReference()==" <<
            rcr->NumberOfReference() << std::endl;
if(rcr && rcr->NumberOfReference()>0)
    rcr->RemoveReference();
else
    std::cout << "WARNING: SharedObjectBase destructed without reference" << std::endl;
//std::cout << "~SharedObjectBase()NOW: rcr->NumberOfReference()==" << rcr->NumberOfReference() << std:
    }
    RCRegion * rcr; //!< the pointer to the shared memory region this is in charge of

private:
    SharedObjectBase(const SharedObjectBase&); //!< this shouldn't be called...
    SharedObjectBase& operator=(const SharedObjectBase&); //!< this shouldn't be called...
};

//! This templated class allows convenient creation of any type of class wrapped in
       a shared memory region
/*! @see MotionManager for an example on how to use this.*/
template<class MC>
class SharedObject : public SharedObjectBase {
public:
    //!if you really need more than 5 arguments for your class, well, you're one
      crazy puppy but if you really want to, just make more like shown... (yay templates!)
    //!@name templated contructors - allows you to pass constructor arguments on to the
     object being created
```

```
    //! Creates the class with the default constructor
    SharedObject() : SharedObjectBase() {
rcr=createRCRegion();
new (rcr->Base()) MC;
    }
    //! Creates the class, passing its constructor t1
    template<class T1> SharedObject(T1 t1) : SharedObjectBase() {
rcr=createRCRegion();
new (rcr->Base()) MC(t1);
    }
    //! Creates the class, passing its constructor t1 and t2
    template<class T1, class T2> SharedObject(T1 t1, T2 t2) : SharedObjectBase(){
rcr=createRCRegion();
new (rcr->Base()) MC(t1,t2);
    }
    //! Creates the class, passing its constructor t1, t2, and t3
    template<class T1, class T2, class T3> SharedObject(T1 t1, T2 t2, T3 t3) : SharedObjectBase(){
rcr=createRCRegion();
new (rcr->Base()) MC(t1,t2,t3);
    }
    //! Creates the class, passing its constructor t1, t2, t3 and t4
    template<class T1, class T2, class T3, class T4> SharedObject(T1 t1, T2 t2, T3 t3,
          T4 t4) : SharedObjectBase(){
rcr=createRCRegion();
new (rcr->Base()) MC(t1,t2,t3,t4);
    }
    //! Creates the class, passing its constructor t1, t2, t3, t4 and t5 - if you need
                more arguments, just add them
    template<class T1, class T2, class T3, class T4, class T5> SharedObject(T1 t1, T2 t2,
       T3 t3, T4 t4, T5 t5) : SharedObjectBase(){
rcr=createRCRegion();
new (rcr->Base()) MC(t1,t2,t3,t4,t5);
    }
    //@}

    MC* operator->() const { return dataCasted(); } //!< smart pointer to the underlying
        class
    MC& operator*() const { return *dataCasted(); } //!< smart pointer to the
       underlying class
    MC& operator[](int i) const { return dataCasted()[i]; } //!< smart pointer to the
        underlying class
protected:
    //! creates and returns RCRegion of correct size for current class.  Adds a reference
            (which is removed in the destructor)
    static RCRegion * createRCRegion() {
RCRegion * r = new RCRegion(calcsize());
//std::cout << "createRCRegion(): rcr->NumberOfReference()==" << r->NumberOfReference() << std::en
r->AddReference();
//std::cout << "createRCRegion()NOW: rcr->NumberOfReference()=="
             << r->NumberOfReference() << std::endl;
return r;
    }
```

```
    MC* dataCasted() const { return static_cast<MC*>(data()); }
        //!< returns a correctly typed pointer to the object's memory

    //!Calculates the size of the memory region to be used, rounding up to the nearest page size
    /*! Not sure this is completely necessary, but may be nice.  Of course, this also means even
     *  small regions are going to be at least 4K (current page size)  If memory gets tight or we
     *  get a lot of little regions floating around, this might be worth checking into */
    static unsigned int calcsize() {
size_t size = sizeof(MC);
sError error;
size_t page_size;
error = GetPageSize(&page_size);
if (error != sSUCCESS) {
    cout << "error: " << error << " getting page size in SharedMem" << endl;
    page_size = 4096;
}

int new_size,num_pages;
num_pages = (size+page_size-1)/page_size;
new_size = num_pages*page_size;
//cout << "req" << size << "new" << new_size << "ps" << page_size << endl;
/* cout << "data size is " << sizeof(MC) << endl;
cout << "msg size is " << MotionManagerMsg::SIZEOF_MSG << endl;
cout << "SIZE is " << rcr->Size() << endl;
cout << "PAGE is " << page_size << endl;
cout << "BASE is " << (void*)rcr->Base() << endl; */
return new_size;
    }
};

/*! @file
 * @brief Defines SharedObject, a wrapper for objects in order to facilitate sending them between proces
 * @author ejt (Creator)
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.3 $
 * $State: Rel $
 * $Date: 2003/09/12 23:42:12 $
 */

#endif //INCLUDED_SharedObject_h
```

D.9 Shared/ERS220Info.h

```
//-*-c++-*-

// Mad props to Daishi MORI, the 220 master chief, for porting to the 220 ;)

#ifndef INCLUDED_ERS220Info_h
#define INCLUDED_ERS220Info_h

#include <math.h>
```

```
#ifndef PLATFORM_APERIOS
typedef unsigned short word; //!< otherwise defined in Types.h
#else
#include <Types.h>
#endif

#if TGT_ERS2xx
#include "ERS2xxInfo.h"
#endif

//! Contains information about the ERS-220 Robot, such as number of joints,
    PID defaults, timing information, etc.
namespace ERS220Info {

#if TGT_ERS2xx
    using namespace ERS2xxInfo;
#else
    // ******************************
    //       ROBOT CONFIGURATION
    // ******************************


    const unsigned int FrameTime=8;  //!< time between frames in the motion system (milliseconds)
    const unsigned int NumFrames=4; //!< the number of frames per buffer (don't forget also double
    const unsigned int SlowFrameTime=128;  //!< time between frames for the ears
    (which move slower for some reason, don't want to mix with other outputs) (milliseconds)
    const unsigned int NumSlowFrames=1;//!< the number of frames per buffer being sent to
         ears (double buffered as well)
    const unsigned int SoundBufferTime=32; //!< the number of milliseconds per sound
           buffer... I'm not sure if this can be changed

    //!Corresponds to entries in PrimitiveName, defined at the end of this file,
        these are the primary grouping
    /*!Right now all binary joints are slow, but perhaps this won't always be the
       case... hence the IsFast/Slow bitmasks to select which type, in order to be more general */
    //!@name Output Types Information
    const unsigned NumPIDJoints   = 15; //!< The number of joints which use PID motion - everythin
    const unsigned NumLEDs        = 20; //!< The number LEDs which can be controlled
    const unsigned NumBinJoints   =  0; //!< The number of binary joints (210 has ears)
    const unsigned NumOutputs     = NumPIDJoints + NumBinJoints + NumLEDs; //!< the total number o

    const bool IsFastOutput[NumOutputs] = {
// for PID joints
true, true, true,
true, true, true,
true, true, true,
true, true, true,
true, true, true,
// for LEDs
true, true, true,           // face left side LEDs x3
true, true, true,           // face right side LEDs x3
true,                       // head mode LED x1
true, true, true,           // back left multi LEDs x3
```

```
true, true, true,           // back right multi LEDs x3
true, true, true,           // tail LEDs x3
true, true, true,           // face front LEDs x3
true,                       // retractable head light x1
// for binary joints (none supported/exist on 220)
    }; //!< true for joints which can be updated every 32 ms (all but the ears on a 210)

    //! we need this so you can tell programmatically which joints are "real" and
           which are "fake" in compatability mode
    const bool IsRealERS220[NumOutputs] = {
// for PID joints
true, true, true,
true, true, true,
true, true, true,
true, true, true,
true, true, true,
// for LEDs
true, true, true,           // face left side LEDs x3
true, true, true,           // face right side LEDs x3
true,                       // head mode LED x1
true, true, true,           // back left multi LEDs x3
true, true, true,           // back right multi LEDs x3
true, true, true,           // tail LEDs x3
true, true, true,           // face front LEDs x3
true,                       // retractable head light x1
// for binary joints (none supported/exist on 220)
    }; //!< true for joints which can be updated every 32 ms (all but the ears on a 210)

    const unsigned JointsPerLeg   =  3; //!< The number of joints per leg
    const unsigned NumLegs        =  4; //!< The number of legs
    const unsigned NumLegJoints   =  JointsPerLeg*NumLegs;
           //!< the TOTAL number of joints on ALL legs
    const unsigned NumHeadJoints  =  3; //!< The number of joints in the neck
    const unsigned NumTailJoints  =  0; //!< The number of joints assigned to the tail
    const unsigned NumMouthJoints =  0; //!< the number of joints that control the mouth
    const unsigned NumEarJoints   =  0; //!< The number of joints which control the ears
            (NOT per ear, is total)
    const unsigned NumButtons     =  11; //!< the number of buttons that are available,
        see ButtonOffset_t
    const unsigned NumSensors     =  1+3+1+5;  //!< 1 dist, 3 accel, 1 thermo,
         5 from power, see SensorOffset_t


    // ********************************
    //         OUTPUT OFFSETS
    // ********************************


    //!Corresponds to entries in PrimitiveName, defined at the end of this file
    //!@name Output Offsets
    const unsigned PIDJointOffset = 0; //!< The beginning of the PID Joints
    const unsigned LegOffset   = PIDJointOffset;//!< the offset of the beginning of the leg joints
    const unsigned HeadOffset  = LegOffset+NumLegJoints;   //!< the offset of the beginning of the head
```

```
    const unsigned LEDOffset   = PIDJointOffset + NumPIDJoints; //!< the offset of LEDs
        in WorldState::outputs and MotionCommand functions

    const unsigned BinJointOffset = NumOutputs; //!< The beginning of the binary joints
    //@}

    //! the ordering of legs
    enum LegOrder_t {
LFrLegOrder = 0, //!< left front leg
RFrLegOrder,     //!< right front leg
LBkLegOrder,     //!< left back leg
RBkLegOrder      //!< right back leg
    };

    //! The offsets of the individual legs
    enum LegOffset_t {
LFrLegOffset = LegOffset+LFrLegOrder*JointsPerLeg, //!< beginning of left front leg
RFrLegOffset = LegOffset+RFrLegOrder*JointsPerLeg, //!< beginning of right front leg
LBkLegOffset = LegOffset+LBkLegOrder*JointsPerLeg, //!< beginning of left back leg
RBkLegOffset = LegOffset+RBkLegOrder*JointsPerLeg  //!< beginning of right back leg
    };

    //! The offsets within appendages (the legs)  Note that the ordering matches the
      actual physical ordering of joints on the appendage (and not that of the head's
        TPROffset_t's)
    enum REKOffset_t {
RotatorOffset=0, //!< moves leg forward or backward along body
ElevatorOffset,  //!< moves leg toward or away from body
KneeOffset       //!< moves knee
    };

    //! The offsets of appendages with tilt (elevation), pan (heading), and roll
     joints (i.e. head)  Note that the ordering matches the actual physical ordering of
          joints on the appendage (and not that of the leg's REKOffset_t's)
    enum TPROffset_t {
TiltOffset = 0, //!< tilt/elevation (vertical)
PanOffset,      //!< pan/heading (horizontal)
RollOffset      //!< roll (rotational)
    };

    //! The offsets of the individual LEDs on the head and tail.  Note that L/R are
          robot's POV.  See also LEDBitMask_t
    enum LEDOffset_t {
FaceFrontLeftLEDOffset = LEDOffset, //!< head face side light (front left - blue)
FaceFrontRightLEDOffset,    //!< head face side light (front right - blue)
FaceCenterLeftLEDOffset,    //!< head face side light (center left - blue)
FaceCenterRightLEDOffset,   //!< head face side light (center right - blue)
FaceBackLeftLEDOffset,      //!< head face side light (back left - red)
FaceBackRightLEDOffset,     //!< head face side light (back right - red)
ModeLEDOffset,              //!< mode indicator (back of the head - orange)
BackLeft1LEDOffset,         //!< back multi-indicator (left #1 - blue)
BackLeft2LEDOffset,         //!< back multi-indicator (left #2 - blue)
```

```
BackLeft3LEDOffset,         //!< back multi-indicator (left #3 - blue)
BackRight3LEDOffset,        //!< back multi-indicator (right #3 - blue)
BackRight2LEDOffset,        //!< back multi-indicator (right #2 - blue)
BackRight1LEDOffset,        //!< back multi-indicator (right #1 - blue)
TailLeftLEDOffset,          //!< tail light (left - blue)
TailCenterLEDOffset,        //!< tail light (center - red)
TailRightLEDOffset,         //!< tail light (right - blue)
FaceFrontBLEDOffset,        //!< face front light B (blue)
FaceFrontALEDOffset,        //!< face front light A (blue)
FaceFrontCLEDOffset,        //!< face front light C (red)
RetractableHeadLEDOffset,   //!< retractable head light

// aliases for backward compatibility
BotLLEDOffset = FaceFrontLeftLEDOffset,   //!< bottom left (red - sad) (ERS-210)
BotRLEDOffset = FaceFrontRightLEDOffset,  //!< bottom right (red - sad) (ERS-210)
MidLLEDOffset = FaceCenterLeftLEDOffset,  //!< middle left (green - happy) (ERS-210)
MidRLEDOffset = FaceCenterRightLEDOffset, //!< middle right (green - happy) (ERS-210)
TopLLEDOffset = FaceBackLeftLEDOffset,    //!< top left (red - angry) (ERS-210)
TopRLEDOffset = FaceBackRightLEDOffset,   //!< top right (red - angry) (ERS-210)
TopBrLEDOffset = ModeLEDOffset,           //!< top bar (green) (ERS-210)
TlBluLEDOffset = TailLeftLEDOffset,      //!< blue tail light (ERS-210)
TlRedLEDOffset = TailRightLEDOffset,      //!< red tail light (ERS-210)
    };

    //! Bitmasks for use when specifying combinations of LEDs (see LEDEngine )
        Note that L/R are robot's POV
    //!@name LED Bitmasks
    typedef unsigned int LEDBitMask_t;//!< So you can be clear when you're refering to a LED bitmask
    const LEDBitMask_t FaceFrontLeftLEDMask   = 1<<(FaceFrontLeftLEDOffset-LEDOffset);
    const LEDBitMask_t FaceFrontRightLEDMask  = 1<<(FaceFrontRightLEDOffset-LEDOffset);
    const LEDBitMask_t FaceCenterLeftLEDMask  = 1<<(FaceCenterLeftLEDOffset-LEDOffset);
    const LEDBitMask_t FaceCenterRightLEDMask = 1<<(FaceCenterRightLEDOffset-LEDOffset);
    const LEDBitMask_t FaceBackLeftLEDMask    = 1<<(FaceBackLeftLEDOffset-LEDOffset);
    const LEDBitMask_t FaceBackRightLEDMask   = 1<<(FaceBackRightLEDOffset-LEDOffset);
    const LEDBitMask_t ModeLEDMask            = 1<<(ModeLEDOffset-LEDOffset);
    const LEDBitMask_t BackLeft1LEDMask       = 1<<(BackLeft1LEDOffset-LEDOffset);
    const LEDBitMask_t BackLeft2LEDMask       = 1<<(BackLeft2LEDOffset-LEDOffset);
    const LEDBitMask_t BackLeft3LEDMask       = 1<<(BackLeft3LEDOffset-LEDOffset);
    const LEDBitMask_t BackRight3LEDMask      = 1<<(BackRight3LEDOffset-LEDOffset);
    const LEDBitMask_t BackRight2LEDMask      = 1<<(BackRight2LEDOffset-LEDOffset);
    const LEDBitMask_t BackRight1LEDMask      = 1<<(BackRight1LEDOffset-LEDOffset);
    const LEDBitMask_t TailLeftLEDMask        = 1<<(TailLeftLEDOffset-LEDOffset);
    const LEDBitMask_t TailCenterLEDMask      = 1<<(TailCenterLEDOffset-LEDOffset);
    const LEDBitMask_t TailRightLEDMask       = 1<<(TailRightLEDOffset-LEDOffset);
    const LEDBitMask_t FaceFrontBLEDMask      = 1<<(FaceFrontBLEDOffset-LEDOffset);
    const LEDBitMask_t FaceFrontALEDMask      = 1<<(FaceFrontALEDOffset-LEDOffset);
    const LEDBitMask_t FaceFrontCLEDMask      = 1<<(FaceFrontCLEDOffset-LEDOffset);
    const LEDBitMask_t RetractableHeadLEDMask = 1<<(RetractableHeadLEDOffset-LEDOffset);

    // aliases for backward compatibility
    const LEDBitMask_t BotLLEDMask = 1<<(BotLLEDOffset-LEDOffset); //!< bottom left (red - sad)
    const LEDBitMask_t BotRLEDMask = 1<<(BotRLEDOffset-LEDOffset); //!< bottom right (red - sad)
    const LEDBitMask_t MidLLEDMask = 1<<(MidLLEDOffset-LEDOffset); //!< middle left (green - happy)
```

```
const LEDBitMask_t MidRLEDMask = 1<<(MidRLEDOffset-LEDOffset); //!< middle right (green - happ
const LEDBitMask_t TopLLEDMask = 1<<(TopLLEDOffset-LEDOffset); //!< top left (red - angry)
const LEDBitMask_t TopRLEDMask = 1<<(TopRLEDOffset-LEDOffset); //!< top right (red - angry)
const LEDBitMask_t TopBrLEDMask= 1<<(TopBrLEDOffset-LEDOffset); //!< top bar (green)
const LEDBitMask_t TlRedLEDMask= 1<<(TlRedLEDOffset-LEDOffset); //!< red tail light
const LEDBitMask_t TlBluLEDMask= 1<<(TlBluLEDOffset-LEDOffset); //!< blue tail light

const LEDBitMask_t FaceLEDMask
= FaceFrontLeftLEDMask
| FaceFrontRightLEDMask
| FaceCenterLeftLEDMask
| FaceCenterRightLEDMask
| FaceBackLeftLEDMask
| FaceBackRightLEDMask
| FaceFrontALEDMask
| FaceFrontBLEDMask
| FaceFrontCLEDMask
| ModeLEDMask;              //!< LEDs for face

const LEDBitMask_t HeadLEDMask
= FaceLEDMask
| RetractableHeadLEDMask;   //!< LEDs on head (face plus retractable light)

const LEDBitMask_t BackLEDMask
= BackLeft1LEDMask
| BackLeft2LEDMask
| BackLeft3LEDMask
| BackRight1LEDMask
| BackRight2LEDMask
| BackRight3LEDMask; //!< LEDs on back

const LEDBitMask_t TailLEDMask
= TailLeftLEDMask
| TailCenterLEDMask
| TailRightLEDMask;  //!< LEDs for tail

const LEDBitMask_t AllLEDMask  = ~0; //!< selects all of the leds
//@}


// ******************************
//          INPUT OFFSETS
// ******************************


//! The order in which inputs should be stored
//!@name Input Offsets

//! holds offsets to different buttons in WorldState::buttons[]
/*! Should be a straight mapping to the ButtonSourceIDs
 *
 *  Note that the chest (power) button is not a normal button.  It kills
 *  power to the motors at a hardware level, and isn't sensed in the
```

```
     *  normal way.  If you want to know when it is pressed (and you are
     *  about to shut down) see PowerSourceID::PauseSID.
     *
     *  @see WorldState::buttons @see ButtonSourceID_t */
    enum ButtonOffset_t {
LFrPawOffset = LFrLegOrder,
RFrPawOffset = RFrLegOrder,
LBkPawOffset = LBkLegOrder,
RBkPawOffset = RBkLegOrder,
ChinButOffset= 4,
BackButOffset,
HeadFrButOffset, //!< for the "antenna" - this is <.2 if pushed back all the way
HeadBkButOffset, //!< for the "antenna" - this is >.98 if pulled forward,
               <.2 if pushed back partly
TailLeftButOffset,
TailCenterButOffset,
TailRightButOffset,
    };

    //! holds offset to different sensor values in WorldState::sensors[]
    /*! @see WorldState::sensors[] */
    enum SensorOffset_t {
IRDistOffset = 0,  //!< in millimeters
BAccelOffset, //!< backward acceleration, in @f$m/s^2@f$, negative if sitting
             on butt (positive for faceplant)
LAccelOffset, //!< acceleration to the robot's left, in @f$m/s^2@f$,
            negative if lying on robot's left side
DAccelOffset, //!< downward acceleration, in @f$m/s^2@f$, negative if
             standing up... be careful about the signs on all of these...
ThermoOffset, //!< in degrees Celcius
PowerRemainOffset, //!< percentage, 0-1
PowerThermoOffset, //!<  degrees Celcius
PowerCapacityOffset, //!< milli-amp hours
PowerVoltageOffset, //!< volts
PowerCurrentOffset //!< milli-amp negative values (maybe positive while charging?)
    };

    //@}


    //! The length of the strings used for each of the outputs in outputNames
           (doesn't include null term)
    const unsigned outputNameLen = 9;
    //! A name of uniform length for referring to joints - handy for posture files, etc.
    const char* const outputNames[NumOutputs] = {
"LFr:rotor",
"LFr:elvtr",
"LFr:knee~",
"RFr:rotor",
"RFr:elvtr",
"RFr:knee~",
"LBk:rotor",
"LBk:elvtr",
```

```
"LBk:knee~",
"RBk:rotor",
"RBk:elvtr",
"RBk:knee~",

"NECK:tilt",
"NECK:pan~",
"NECK:roll",

"LED:botL~",
"LED:botR~",
"LED:midL~",
"LED:midR~",
"LED:topL~",
"LED:topR~",
"LED:topBr",

"LED:bkL1~",                // "LED:tlBlu" of ERS-210
"LED:bkL2~",                // "LED:tlRed" of ERS-210
"LED:bkL3~",
"LED:bkR3~",
"LED:bkR2~",
"LED:bkR1~",
"LED:tailL",
"LED:tailC",
"LED:tailR",
"LED:faceB",
"LED:faceA",
"LED:faceC",
"LED:light",                 // retractable head light
    };


    //! the joint identifier strings used to refer to specific joints in OPEN-R
          (but not needed for others)
    /*!@showinitializer
     * @warning IMPORTANT!!!!  DO NOT CHANGE THE ORDER OF ITEMS IN THIS TABLE!!!\n
     *
     * The offset consts defined in this file correspond to this table and will make life easier
     * if you feel the need to reorder things, but they aren't used perfect @e everywhere \n
     * In particular, assumptions are made that the pid joints will be in slots 0-numPIDJoints
     * and that the fast outputs (ie NOT ears) will be in slots 0-NumFastOutputs\n
     * There may be other assumptions not noted here!!!
     * @note These entries DON'T correspond to the CPC index numbers defined in WorldState
          (this only lists joints, and in a different order defined by OPEN-R, that one
           has sensors as well*/
    const char* const PrimitiveName [NumOutputs] = {
"PRM:/r2/c1-Joint2:j1",       //!< the left front leg   the rotator
"PRM:/r2/c1/c2-Joint2:j2",    //!< the left front leg   the elevator
"PRM:/r2/c1/c2/c3-Joint2:j3", //!< the left front leg   the knee
"PRM:/r4/c1-Joint2:j1",       //!< the right front leg   the rotator
"PRM:/r4/c1/c2-Joint2:j2",    //!< the right front leg    the elevator
"PRM:/r4/c1/c2/c3-Joint2:j3", //!< the right front leg   the knee
```

```
"PRM:/r3/c1-Joint2:j1",       //!< the left hind leg   the rotator
"PRM:/r3/c1/c2-Joint2:j2",    //!< the left hind leg   the elevator
"PRM:/r3/c1/c2/c3-Joint2:j3", //!< the left hind leg   the knee
"PRM:/r5/c1-Joint2:j1",       //!< the right hind leg   the rotator
"PRM:/r5/c1/c2-Joint2:j2",    //!< the right hind leg   the elevator
"PRM:/r5/c1/c2/c3-Joint2:j3", //!< the right hind leg   the knee

"PRM:/r1/c1-Joint2:j1",       //!< the neck  tilt (12)
"PRM:/r1/c1/c2-Joint2:j2",    //!< the neck   pan
"PRM:/r1/c1/c2/c3-Joint2:j3", //!< the neck   roll

"PRM:/r1/c1/c2/c3/l1-LED2:l1", //!< lower  left  LED (15)
"PRM:/r1/c1/c2/c3/l4-LED2:l4", //!< lower  right LED
"PRM:/r1/c1/c2/c3/l2-LED2:l2", //!< middle left  LED
"PRM:/r1/c1/c2/c3/l5-LED2:l5", //!< middle right LED
"PRM:/r1/c1/c2/c3/l3-LED2:l3", //!< upper  left  LED
"PRM:/r1/c1/c2/c3/l6-LED2:l6", //!< upper  right LED
"PRM:/r1/c1/c2/c3/l7-LED2:l7", //!< top          LED

"PRM:/r6/l1-LED2:l1", //!< back 1st left LED (corresponds to tail blue LED of ERS-210)
"PRM:/r6/l2-LED2:l2", //!< back 2nd left LED (corresponds to tail red  LED of ERS-210)
"PRM:/r6/l3-LED2:l3", //!< back 3rd left LED
"PRM:/r6/l4-LED2:l4", //!< back 3rd right LED
"PRM:/r6/l5-LED2:l5", //!< back 2nd right LED
"PRM:/r6/l6-LED2:l6", //!< back 1st right LED

"PRM:/r6/l9-LED2:l9", //!< tail left LED
"PRM:/r6/l7-LED2:l7", //!< tail center LED
"PRM:/r6/l8-LED2:l8", //!< tail right LED

"PRM:/r1/c1/c2/c3/l8-LED2:l8", //!< face front LED B
"PRM:/r1/c1/c2/c3/l9-LED2:l9", //!< face front LED A
"PRM:/r1/c1/c2/c3/la-LED2:la", //!< face front LED C
"PRM:/r1/c1/c2/c3/lb-LED2:lb", //!< retractable head light
    };

    //Old PID table:
    /*const word Pid[NumPIDJoints][6] = {
      { 0x16, 0x04, 0x08, 0x0E, 0x02, 0x0F },
      { 0x14, 0x04, 0x06, 0x0E, 0x02, 0x0F },
      { 0x23, 0x04, 0x05, 0x0E, 0x02, 0x0F },
      { 0x16, 0x04, 0x08, 0x0E, 0x02, 0x0F },
      { 0x14, 0x04, 0x06, 0x0E, 0x02, 0x0F },
      { 0x23, 0x04, 0x05, 0x0E, 0x02, 0x0F },
      { 0x16, 0x04, 0x08, 0x0E, 0x02, 0x0F },
      { 0x14, 0x04, 0x06, 0x0E, 0x02, 0x0F },
      { 0x23, 0x04, 0x05, 0x0E, 0x02, 0x0F },
      { 0x16, 0x04, 0x08, 0x0E, 0x02, 0x0F },
      { 0x14, 0x04, 0x06, 0x0E, 0x02, 0x0F },
      { 0x23, 0x04, 0x05, 0x0E, 0x02, 0x0F },

      { 0x0A, 0x08, 0x0C, 0x0E, 0x02, 0x0F },
```

```
      { 0x0D, 0x08, 0x0B, 0x0E, 0x02, 0x0F },
      { 0x10, 0x08, 0x0C, 0x0E, 0x02, 0x0F }, // P was 0x0C, updated as seen on
        https://www.openr.org/page1_2001/gain.html 8/13/2002

      { 0x0A, 0x00, 0x18, 0x0E, 0x02, 0x0F },
      { 0x07, 0x00, 0x11, 0x0E, 0x02, 0x0F },

      { 0x0E, 0x08, 0x10, 0x0E, 0x02, 0x0F }, //  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
          { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
      };*/

    //! This table holds the default PID values for each joint.  @see PIDMC
    const float DefaultPIDs[NumPIDJoints][3] =
    {
{ 0x16/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x08/(double)(1<<(16-0xF)) },
{ 0x14/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x06/(double)(1<<(16-0xF)) },
{ 0x23/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x05/(double)(1<<(16-0xF)) },
{ 0x16/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x08/(double)(1<<(16-0xF)) },
{ 0x14/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x06/(double)(1<<(16-0xF)) },
{ 0x23/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x05/(double)(1<<(16-0xF)) },
{ 0x16/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x08/(double)(1<<(16-0xF)) },
{ 0x14/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x06/(double)(1<<(16-0xF)) },
{ 0x23/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x05/(double)(1<<(16-0xF)) },
{ 0x16/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x08/(double)(1<<(16-0xF)) },
{ 0x14/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x06/(double)(1<<(16-0xF)) },
{ 0x23/(double)(1<<(16-0xE)), 0x04/(double)(1<<(16-0x2)), 0x05/(double)(1<<(16-0xF)) },

{ 0x0A/(double)(1<<(16-0xE)), 0x08/(double)(1<<(16-0x2)), 0x0C/(double)(1<<(16-0xF)) },
{ 0x0D/(double)(1<<(16-0xE)), 0x08/(double)(1<<(16-0x2)), 0x0B/(double)(1<<(16-0xF)) },
{ 0x0A/(double)(1<<(16-0xE)), 0x08/(double)(1<<(16-0x2)), 0x0C/(double)(1<<(16-0xF)) }

// { 0x0A/(double)(1<<(16-0xE)), 0x00/(double)(1<<(16-0x2)), 0x18/(double)(1<<(16-0xF)) },
// { 0x07/(double)(1<<(16-0xE)), 0x00/(double)(1<<(16-0x2)), 0x11/(double)(1<<(16-0xF)) },

// { 0x0E/(double)(1<<(16-0xE)), 0x08/(double)(1<<(16-0x2)), 0x10/(double)(1<<(16-0xF)) }
    };

//!These values are Sony's recommended maximum joint velocities, in rad/ms
    /*! a value <= 0 means infinite speed (e.g. LEDs) */
    const float MaxOutputSpeed[NumOutputs] = {
2.8143434e-03,     //Legs LR,FB,REK
2.4980025e-03,
2.8361600e-03,
2.8143434e-03,
2.4980025e-03,
2.8361600e-03,
2.8143434e-03,
2.4980025e-03,
2.8361600e-03,
2.8143434e-03,
2.4980025e-03,
2.8361600e-03,
```

```
2.1053034e-03,     //Head TPR
3.0106930e-03,
3.0106930e-03,

0,0,0,//LEDs
0,0,0,
0,
0,0,0,
0,0,0,
0,0,0,
0,0,0,
0
    };

#ifndef RAD
    //!Just a little macro for converting degrees to radians
#define RAD(deg) (((deg) * M_PI ) / 180.0)
    //!a flag so we undef these after we're done - do you have a cleaner solution?
#define __RI_RAD_FLAG
#endif

    //! Defines the min and max index of entries in #outputRanges and #mechanicalLimits
    enum MinMaxRange_t { MinRange,MaxRange };

    //! This table holds the software limits of each of the outputs
    const double outputRanges[NumOutputs][2] =
    {
{ RAD(-117),RAD(117) },{ RAD(-11),RAD(89) },{ RAD(-27),RAD(147) }, //left front REK
{ RAD(-117),RAD(117) },{ RAD(-11),RAD(89) },{ RAD(-27),RAD(147) }, //right front REK
{ RAD(-117),RAD(117) },{ RAD(-11),RAD(89) },{ RAD(-27),RAD(147) }, //left back REK
{ RAD(-117),RAD(117) },{ RAD(-11),RAD(89) },{ RAD(-27),RAD(147) }, //right back REK

{ RAD(-82),RAD(43) },{ RAD(-89.6),RAD(89.6) },{ RAD(-29),RAD(29) }, //neck TPR

{0,1},{0,1},{0,1},        // face left side LEDs x3
{0,1},{0,1},{0,1},        // face right side LEDs x3
{0,1},                    // head mode LED x1
{0,1},{0,1},{0,1},        // back left multi LEDs x3
{0,1},{0,1},{0,1},        // back right multi LEDs x3
{0,1},{0,1},{0,1},        // tail LEDs x3
{0,1},{0,1},{0,1},        // face front LEDs x3
{0,1}                     // retractable head light x1
    };

    //! This table holds the mechanical limits of each of the outputs
    const double mechanicalLimits[NumOutputs][2] =
    {
{ RAD(-120),RAD(120) },{ RAD(-14),RAD(92) },{ RAD(-30),RAD(150) }, //left front jsk
{ RAD(-120),RAD(120) },{ RAD(-14),RAD(92) },{ RAD(-30),RAD(150) }, //right front jsk
{ RAD(-120),RAD(120) },{ RAD(-14),RAD(92) },{ RAD(-30),RAD(150) }, //left back jsk
{ RAD(-120),RAD(120) },{ RAD(-14),RAD(92) },{ RAD(-30),RAD(150) }, //right back jsk

{ RAD(-85),RAD(46) },{ RAD(-92.6),RAD(92.6) },{ RAD(-32),RAD(32) }, //neck tpr
```

```
{0,1},{0,1},{0,1},        // face left side LEDs x3
{0,1},{0,1},{0,1},        // face right side LEDs x3
{0,1},                    // head mode LED x1
{0,1},{0,1},{0,1},        // back left multi LEDs x3
{0,1},{0,1},{0,1},        // back right multi LEDs x3
{0,1},{0,1},{0,1},        // tail LEDs x3
{0,1},{0,1},{0,1},        // face front LEDs x3
{0,1}                     // retractable head light x1
    };

#ifdef __RI_RAD_FLAG
#undef RAD
#undef __RI_RAD_FLAG
#endif

#endif //TGT_ERS2xx check

    /*! @name CPC IDs
     * values defined by OPEN-R, used to interface with lower level OPEN-R code to read
         sensors - DOESN'T correspond to ::PrimitiveName */
    static const int CPCJointNeckTilt           =  0; // PRM:/r1/c1-Joint2:j1
    static const int CPCJointNeckPan            =  1; // PRM:/r1/c1/c2-Joint2:j2
    static const int CPCJointNeckRoll           =  2; // PRM:/r1/c1/c2/c3-Joint2:j3
    static const int CPCSensorPSD               =  3; // PRM:/r1/c1/c2/c3/p1-Sensor:p1
    static const int CPCSensorHeadBackPressure  =  4; // PRM:/r1/c1/c2/c3/f1-Sensor:f1
    static const int CPCSensorHeadFrontPressure =  5; // PRM:/r1/c1/c2/c3/f2-Sensor:f2
    static const int CPCSensorChinSwitch        =  6; // PRM:/r1/c1/c2/c3/c4/s5-Sensor:s5
    static const int CPCJointLFRotator          =  7; // PRM:/r2/c1-Joint2:j1
    static const int CPCJointLFElevator         =  8; // PRM:/r2/c1/c2-Joint2:j2
    static const int CPCJointLFKnee             =  9; // PRM:/r2/c1/c2/c3-Joint2:j3
    static const int CPCSensorLFPaw             = 10; // PRM:/r2/c1/c2/c3/c4-Sensor:s4
    static const int CPCJointLHRotator          = 11; // PRM:/r3/c1-Joint2:j1
    static const int CPCJointLHElevator         = 12; // PRM:/r3/c1/c2-Joint2:j2
    static const int CPCJointLHKnee             = 13; // PRM:/r3/c1/c2/c3-Joint2:j3
    static const int CPCSensorLHPaw             = 14; // PRM:/r3/c1/c2/c3/c4-Sensor:s4
    static const int CPCJointRFRotator          = 15; // PRM:/r4/c1-Joint2:j1
    static const int CPCJointRFElevator         = 16; // PRM:/r4/c1/c2-Joint2:j2
    static const int CPCJointRFKnee             = 17; // PRM:/r4/c1/c2/c3-Joint2:j3
    static const int CPCSensorRFPaw             = 18; // PRM:/r4/c1/c2/c3/c4-Sensor:s4
    static const int CPCJointRHRotator          = 19; // PRM:/r5/c1-Joint2:j1
    static const int CPCJointRHElevator         = 20; // PRM:/r5/c1/c2-Joint2:j2
    static const int CPCJointRHKnee             = 21; // PRM:/r5/c1/c2/c3-Joint2:j3
    static const int CPCSensorRHPaw             = 22; // PRM:/r5/c1/c2/c3/c4-Sensor:s4
    static const int CPCSensorThermoSensor      = 23; // PRM:/r6/t1-Sensor:t1
    static const int CPCSensorBackSwitch        = 24; // PRM:/r6/s1-Sensor:s1
    static const int CPCSensorTailLeftSwitch    = 25; // PRM:/r6/s2-Sensor:s2  (ERS-220 only)
    static const int CPCSensorTailCenterSwitch  = 26; // PRM:/r6/s3-Sensor:s3  (ERS-220 only)
    static const int CPCSensorTailRightSwitch   = 27; // PRM:/r6/s4-Sensor:s4  (ERS-220 only)
    static const int CPCSensorAccelFB           = 28; // PRM:/a1-Sensor:a1
    static const int CPCSensorAccelLR           = 29; // PRM:/a2-Sensor:a2
    static const int CPCSensorAccelUD           = 30; // PRM:/a3-Sensor:a3
    //@}
```

```
}

/*! @file
 * @brief Defines RobotInfo namespace for ERS-220 models, gives some information about
       the robot's capabilities, such as joint counts, offsets, names and PID values
 * @author Daishi MORI (Creator)
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.7 $
 * $State: Rel $
 * $Date: 2003/09/25 15:31:52 $
 */

#endif
```

D.10 SoundPlay/SoundManager.h

```
//-*-c++-*-
#ifndef INCLUDED_SoundManager_h_
#define INCLUDED_SoundManager_h_

#include <OPENR/RCRegion.h>
#include <OPENR/ODataFormats.h>

#include <string>
#include <vector>
#include "Shared/ListMemBuf.h"
#include "Shared/MutexLock.h"
#include "SoundManagerMsg.h"
#include "Shared/ProcessID.h"

class OSubject;
class ONotifyEvent;

//! Provides sound effects and caching services, as well as mixing buffers for the SoundPlay process
/*! Provides easy methods for playing back sounds, either from files on the memory stick,
 *  or from dynamically generated buffers.  You can chain playback commands so that when
 *  one sound finishes, another picks up automatically.  This might be handy if, say, someone
 *  wants to write an MP3 player ;)  The sounds would be too large to load into memory all at
 *  once, but you could load a block at a time and chain them so it seamlessly moves from one
 *  to the other.
 *
 *  All functions will attempt to lock the SoundManager.
 *
 *  @todo Volume control, variable playback speed, support more wav file formats (all go together)
 *
 *  @todo Add functions to hand out regions to be filled out to avoid copying into the buffer.
 */
class SoundManager {
public:
    //!constructor
```

```
    SoundManager();

    //!This is used for referring to sound data so you can start playing it or release it
    typedef SoundManagerMsg::Snd_ID Snd_ID;
    static const Snd_ID invalid_Snd_ID=(Snd_ID)-1; //!< for reporting errors
    static const Snd_ID MAX_SND=50; //!< the number of sounds that can be loaded at any given time

    //!This is for referring to instances of the play command so you can stop, pause, or
         monitor progress (later versions will send events upon completion)
    typedef unsigned short Play_ID;
    static const Play_ID invalid_Play_ID=(Play_ID)-1; //!< for reporting errors
    static const Play_ID MAX_PLAY=256; //!< the number of sounds that can be enqueued at
          the same time (see MixMode_t)

    static const unsigned int MAX_NAME_LEN=64;   //!<maximum length of a path

    //!Used to set the mode for mixing multiple sound channels
    /*!Feel free to add a higher quality mixer if you're an audiophile - I'm pretty new
            to sound processing*/
    enum MixMode_t {
Fast,    //!< uses bit shifting trick, but can result in reduced volume with
            more active channels, best if you set max_channels to a power of 2
Quality  //!< uses real division to maintain volume level, although still a
             rather naive (but relatively fast) algorithm
    };

    enum QueueMode_t {
Enqueue,  //!< newer sounds are played when a channel opens up (when old sound finishes)
Pause,   //!< newer sounds pause oldest sound, which continues when a channel opens
Stop,           //!< newer sounds stop oldest sound
Override,//!< older sounds have play heads advanced, but don't get mixed until a channel opens
    };

    //!Needed to send sounds to the SoundPlay process
    void InitAccess(OSubject* subj);

    //!loads a wav file (if it matches Config::sound_config settings - can't do resampling yet)
    /*!Since the SoundManager does the loading, if the same file is being played more than once, o
     * @param name can be either a full path, or a partial path relative to Config::sound_config::
     * @return ID number for future references (can also use name)
     * The sound data will be cached until ReleaseFile() or Release() is called a
         matching number of times*/
    Snd_ID LoadFile(const char* name);

    //!loads raw samples from a buffer (assumes matches Config::sound_config settings)
    /*!The sound data will be cached until Release() is called a matching number of times.\n
     * This function is useful for dynamic sound sources.  A copy will be made. */
    Snd_ID LoadBuffer(const char buf[], unsigned int len);

    //!Marks the sound buffer to be released after the last play command completes
           (or right now if not being played)
    void ReleaseFile(const char* name);
```

```
//!Marks the sound buffer to be released after the last play command completes
      (or right now if not being played)
void Release(Snd_ID id);

//!play a wav file (if it matches Config::sound_config settings -
       can't do resampling yet)
/*!Will do a call to LoadFile() first, and then automatically release the sound again when complete
 * @param name can be either a full path, or a partial path relative to Config::sound_config::root
 * @return ID number for future references
 * The sound data will not be cached after done playing unless a call to LoadFile is made*/
Play_ID PlayFile(const char* name);

//!loads raw samples from a buffer (assumes buffer matches Config::sound_config settings)
/*!The sound data will be released after done playing*/
Play_ID PlayBuffer(const char buf[], unsigned int len);

//!plays a previously loaded buffer or file
Play_ID Play(Snd_ID id);

//!allows automatic queuing of sounds - good for dynamic sound sources!
/*!if you chain more than once to the same base, the new buffers are appended
 * to the end of the chain - the new buffer doesn't replace the current chain
 * @return @a base - just for convenience of multiple calls*/
Play_ID ChainFile(Play_ID base, const char* next);

//!allows automatic queuing of sounds - good for dynamic sound sources!
/*!if you chain more than once to the same base, the new buffers are appended
 * to the end of the chain - the new buffer doesn't replace the current chain
 * @return @a base - just for convenience of multiple calls*/
Play_ID ChainBuffer(Play_ID base, const char buf[], unsigned int len);

//!allows automatic queuing of sounds - good for dynamic sound sources!
/*!if you chain more than once to the same base, the new buffers are appended
 * to the end of the chain - the new buffer doesn't replace the current chain
 * @return @a base - just for convenience of multiple calls*/
Play_ID Chain(Play_ID base, Snd_ID next);

//!Lets you stop playback of all sounds
void StopPlay();

//!Lets you stop playback of a sound
void StopPlay(Play_ID id);

//!Lets you pause playback
void PausePlay(Play_ID id);

//!Lets you resume playback
void ResumePlay(Play_ID id);

//!Lets you control the maximum number of channels (currently playing sounds), method
   for mixing (applies when max_chan>1), and queuing method (for when overflow channels)
void SetMode(unsigned int max_channels, MixMode_t mixer_mode, QueueMode_t queuing_mode);
```

```
    //!Gives the time until the sound finishes, in milliseconds.  Subtract 32 to get
              guarranteed valid time for this ID.
    /*!You should be passing the beginning of a chain to get proper results...\n
     * May be slightly conservative (will report too small a time) because this
     * does not account for delay until SoundPlay picks up the message that a
     * sound has been added.\n
     * However, it is slighly optimistic (will report too large a time) because
     * it processes a buffer all at one go, so it could mark the sound as finished
     * (and cause the ID to go invalid) up to RobotInfo::SoundBufferTime (32 ms)
     * before the sound finishes.  So subtract SoundBufferTime if you want to be
     * absolutely sure the ID will still valid. */
    unsigned int GetRemainTime(Play_ID id) const;

    //!Copies the sound data to the OPENR buffer, ready to be passed to the system,
         only called by SoundPlay
    /*!@return the number of active sounds */
    unsigned int CopyTo(OSoundVectorData* data);

    //!updates internal data structures on the SoundPlay side - you shouldn't be calling this
    void ReceivedMsg(const ONotifyEvent& event);

    //!@return number of sounds currently playing
    unsigned int GetNumPlaying() { return chanlist.size(); }

protected:
    //!Sets up a shared region to hold a sound - rounds to nearest page size
    static RCRegion* initRegion(unsigned int size);

    //!Looks to see if @a name matches any of the sounds in sndlist
    Snd_ID lookup(const char* name) const;
    //!Looks to see if @a name matches any of the sounds in sndlist (assumes is absolute path)
    Snd_ID lookupPath(const char* path) const;

    //!prepends config.sound.root to the name if necessary
    static const char* makePath(const char* name, char tmp[MAX_NAME_LEN]);

    //!selects which of the channels are actually to be mixed together, depending on queue_mode
    void selectChannels(std::vector<Play_ID>& mix);

    //!update the offsets of sounds which weren't mixed (when needed depending on queue_mode)
    void updateChannels(const std::vector<Play_ID>& mixs,size_t used);

    //!called when a buffer end is reached, may reset buffer to next in chain, or just StopPlay()
    bool endPlay(Play_ID id);

    //!Holds data about the loaded sounds
    struct SoundData {
SoundData();                            //!<constructor
RCRegion * rcr;                         //!<shared region - don't need to share
              among processes, just collect in SoundPlay
byte* data;    //!<point to data in region (for convenience, only valid in SoundPlay)
unsigned int len;                       //!<size of the sound
unsigned int ref;                       //!<reference counter
```

```
char name[SoundManager::MAX_NAME_LEN];//!<stores the path to the file, empty if from a buffer
    private:
SoundData(const SoundData&);             //!< don't call
SoundData operator=(const SoundData&);   //!< don't call
    };
    //!For convenience
    typedef ListMemBuf<SoundData,MAX_SND,Snd_ID> sndlist_t;
    //!Holds a list of all currently loaded sounds
    sndlist_t sndlist;

//!Holds data about sounds currently being played
    struct PlayState {
PlayState();            //!<constructor
Snd_ID snd_id;          //!<index of sound
unsigned int offset;    //!<position in the sound
unsigned int cumulative;//!<total time of playing (over queued sounds)
Play_ID next_id;        //!<lets you queue for continuous sound, or loop
    };
    //!For convenience
    typedef ListMemBuf<PlayState,MAX_PLAY,Play_ID> playlist_t;
    //!Holds a list of all sounds currently enqueued
    playlist_t playlist;
    //!For convenience
    typedef ListMemBuf<Play_ID,MAX_PLAY,Play_ID> chanlist_t;
    //!Holds a list of all currently playing sounds, ordered newest (front) to oldest(back)
    chanlist_t chanlist;

    //!Current mixing mode, set by SetMode();
    MixMode_t mix_mode;

    //!Current queuing mode, set by SetMode();
    QueueMode_t queue_mode;

    //!Current maximum number of sounds to mix together
    unsigned int max_chan;

    //!Prevents multiple processes from accessing at the same time
    mutable MutexLock<ProcessID::NumProcesses> lock;

    //!For automatic transmission of shared regions to SoundPlay
    OSubject * subjs[ProcessID::NumProcesses];

    SoundManager(const SoundManager&);           //!< don't call
    SoundManager operator=(const SoundManager&); //!< don't call
};

//! lets you play a sound from anywhere in your code - just a one liner!
extern SoundManager * sndman;

/*! @file
 * @brief Describes SoundManager, which provides sound effects and caching services,
         as well as mixing buffers for the SoundPlay process
 * @author ejt (Creator)
```

```
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.9 $
 * $State: Rel $
 * $Date: 2003/09/25 15:32:08 $
 */

#endif
```

D.11 Events/EventRouter.h

```
//-*-c++-*-
#ifndef INCLUDED_EventRouter_h
#define INCLUDED_EventRouter_h

#include <vector>
#include <list>
#include <map>
#include <algorithm>
#include "EventListener.h"
#include "EventTrapper.h"
#include "Shared/get_time.h"
#include "Shared/debuget.h"
#include <iostream>

//! This class will handle distribution of events as well as management of timers
/*! Classes must inherit from EventListener and/or EventTrapper in order to
 *  receive events.\n
 *  Use the global @c ::erouter EventRouter to post and subscribe to events\n
 *
 *  When multiple listeners are subscribed, the order in which an event is
 *  distributed among them looks like this:
 *  -# "Specific" listeners: any listener which specifies a particular source id.
 *     (It doesn't matter if they specify a type id or not.)
 *    - older listeners get events before younger listeners
 *  -# "General" listeners: those that subscribe to an entire generator
 *    - older listeners get events before younger listeners
 *
 *  ...but if you're relying on that ordering, there should be a cleaner
 *  way to do whatever you're doing.
 *
 *  If one behaviors unsubscribes another one during a processEvent(), that
 *  behavior will still get the "current" event before the unsubscription
 *  takes place.
 *
 *  Buffering events has not been tested thoroughly...
 *
 *  @see EventBase::EventGeneratorID_t for a complete listing of all generators,
 *  as well as instructions on how to add new generators.
 */
class EventRouter : public EventListener {
public:
```

```
EventRouter(); //!< Constructs the router, #buffertime defaults to 1
virtual ~EventRouter() { reset(); removeAllTimers(); } //!< just calls reset and removeAllTimers()
void reset() { listeners.clear(); trappers.clear(); removeAllTimers(); }
     //!< erases all listeners, trappers and timers, resets EventRouter

void setBufferTime(unsigned int t) { buffertime=t; } /*!< @brief sets the time to wait
   between buffer clears, see EventRouter::buffertime. @param t time to wait between
       buffer clears @see buffertime */
unsigned int getBufferTime() { return buffertime; } /*!< @brief returns the time to
      wait between buffer clears, see EventRouter::buffertime. @return time to wait
     between buffer clears @see buffertime */

//!@name Posting/Processing Events
/*!@brief recommended to create and post an event using current buffer setting */
void postEvent(EventBase::EventGeneratorID_t egid, unsigned int sid,
 EventBase::EventTypeID_t etid, unsigned int dur) { if(buffertime>0) events.push_back(new
 EventBase(egid,sid,etid,dur)); else processEvent(EventBase(egid,sid,etid,dur)); }
void postEvent(EventBase::EventGeneratorID_t egid, unsigned int sid,
  EventBase::EventTypeID_t etid, unsigned int dur, const std::string& n, float m)
   { if(buffertime>0) events.push_back(new EventBase(egid,sid,etid,dur,n,m));
      else processEvent(EventBase(egid,sid,etid,dur,n,m)); }
void postEvent(EventBase* e) { if(buffertime>0) events.push_back(e); else
     { processEvent(*e); delete e; } }

//! determines if timers need to be posted, and posts them if so.
/*! Call this often to ensure accurate timers.  Also, will clear event buffer before sending
  timer events in order to ensure correct ordering of event reception */
void processTimers();
void processEventBuffer(); //!< clears the event buffer, deletes events as it does so.
void processEvent(const EventBase& e); //!< forces unbuffered - sends event *now*.
      Will clear the buffer first if needed to ensure proper event ordering
//@}

//!@name Listener Detection
/*!@brief <b>counts both listeners and trappers</b>, so stuff can tell if it even
      needs to bother generating an event...*/
/* ... if a tree falls in a forest, and there's no one around to see it, does it make a sound?\n
   ... if Vision sees a ball in an image, and there's no listeners, does it make an event? ;) */
bool hasListeners(EventBase::EventGeneratorID_t egid) { return trappers.hasMapping
      (egid) || listeners.hasMapping(egid); }
bool hasListeners(EventBase::EventGeneratorID_t egid, unsigned int sid)
    { return trappers.hasMapping(egid,sid) || listeners.hasMapping(egid,sid); }
bool hasListeners(EventBase::EventGeneratorID_t egid, unsigned int sid,
    EventBase::EventTypeID_t etid) { return trappers.hasMapping(egid,sid,etid) ||
     listeners.hasMapping(egid,sid,etid); }
//@}

//!@name Timer Management
void addTimer(EventListener* el, unsigned int sid, unsigned int delay, bool repeat=true);
   //!< adds a timer or sets a timer if it doesn't already exist.
void addTimer(EventListener* el, const EventBase& e, bool repeat=true){ addTimer(el,
   e.getSourceID(),e.getDuration(),repeat); } //!< calls the other addTimer() with
  the event's source id and duration, doesn't check to see if the generator is timerEGID
```

```
void removeTimer(EventListener* el); //!< clears all pending timers for listener @a el
void removeTimer(EventListener* el, unsigned int sid); //!< clears any pending timers
    with source id @a sid for listener @a el
void removeAllTimers(); //!< clears all timers for all listeners
//@}

//!@name Listener Management
void addListener(EventListener* el, const EventBase& e); //!< Adds a listener for a
  specific source id and type from a given event generator, adding a Timer event will
  invoke addTimer(@a el, @a e.getSourceID(), @a e.getDuration(), @c true)
void addListener(EventListener* el, EventBase::EventGeneratorID_t egid);
      //!< Adds a listener for all events from a given event generator
void addListener(EventListener* el, EventBase::EventGeneratorID_t egid, unsigned int
    sid); //!< Adds a listener for all types from a specific source and generator
void addListener(EventListener* el, EventBase::EventGeneratorID_t egid, unsigned int
   sid, EventBase::EventTypeID_t etid); //!< Adds a listener for a specific source
      id and type from a given event generator

//! stops sending specified events from the generator to the listener.  If a timer
      is passed it will invoke removeTimer(@a el, @a e.getSourceID())
void removeListener(EventListener* el, const EventBase& e);
void removeListener(EventListener* el, EventBase::EventGeneratorID_t egid) {
      listeners.removeMapping(el,egid); listeners.clean(); }
     //!< stops sending specified events from the generator to the listener.
void removeListener(EventListener* el, EventBase::EventGeneratorID_t egid, unsigned
 int sid) { for(unsigned int et=0; et<EventBase::numETIDs; et++)
 listeners.removeMapping(el,egid,sid,(EventBase::EventTypeID_t)et); listeners.clean();
   } //!< stops sending specified events from the generator to the listener.
void removeListener(EventListener* el, EventBase::EventGeneratorID_t egid, unsigned
  int sid, EventBase::EventTypeID_t etid) { listeners.removeMapping(el,egid,sid,etid);
   listeners.clean(); } //!< stops sending specified events from the generator to the listener

void removeListener(EventListener* el) { for(unsigned int eg=0; eg<EventBase::numEGIDs;
    eg++) listeners.removeMapping(el,(EventBase::EventGeneratorID_t)eg); listeners.clean(); }
    //!< stops sending ALL events to the listener
void forgetListener(EventListener* el) { removeTimer(el); removeListener(el); }
    //!< clears timers and removes listener from all events
//@}

//!@name Trapper Management
void addTrapper(EventTrapper* el, const EventBase& e) { trappers.addMapping(el,
   e.getGeneratorID(),e.getSourceID(),e.getTypeID()); }
      //!< Adds a trapper for a specific source id and type from a given event generator
void addTrapper(EventTrapper* el, EventBase::EventGeneratorID_t egid)
  { trappers.addMapping(el,egid); } //!< Adds a trapper for all events from a given event gene
void addTrapper(EventTrapper* el, EventBase::EventGeneratorID_t egid, unsigned int sid)
  { for(unsigned int et=0; et<EventBase::numETIDs; et++) trappers.addMapping(el,egid,
    sid,(EventBase::EventTypeID_t)et); }
      //!< Adds a trapper for all types from a specific source and generator
void addTrapper(EventTrapper* el, EventBase::EventGeneratorID_t egid, unsigned int sid,
     EventBase::EventTypeID_t etid) { trappers.addMapping(el,egid,sid,etid); }
    //!< Adds a trapper for a specific source id and type from a given event generator
```

```
    void addTrapper(EventTrapper* el) { for(unsigned int eg=0; eg<EventBase::numEGIDs; eg++)
    trappers.addMapping(el,(EventBase::EventGeneratorID_t)eg); } //!< adds a trapper for ALL events

    //! stops sending specified events from the generator to the trapper.
    void removeTrapper(EventTrapper* el, const EventBase& e) { trappers.removeMapping(el,
       e.getGeneratorID(),e.getSourceID(),e.getTypeID()); trappers.clean(); }
    void removeTrapper(EventTrapper* el, EventBase::EventGeneratorID_t egid) {trappers.removeMapping(el
      trappers.clean(); } //!< stops sending specified events from the generator to the trapper.
    void removeTrapper(EventTrapper* el, EventBase::EventGeneratorID_t egid, unsigned int sid)
     { for(unsigned int et=0; et<EventBase::numETIDs; et++) trappers.removeMapping(el,egid,sid,
       (EventBase::EventTypeID_t)et); trappers.clean(); }
        //!< stops sending specified events from the generator to the trapper.
    void removeTrapper(EventTrapper* el, EventBase::EventGeneratorID_t egid,
       unsigned int sid, EventBase::EventTypeID_t etid) { trappers.removeMapping(el,egid,
        sid,etid); trappers.clean(); }
           //!< stops sending specified events from the generator to the trapper.

    void removeTrapper(EventTrapper* el) { for(unsigned int eg=0; eg<EventBase::numEGIDs;
     eg++) trappers.removeMapping(el,(EventBase::EventGeneratorID_t)eg); trappers.clean(); }
      //!< stops sending ALL events to the trapper
    //@}

protected:
    void doSendBuffer(); //!< does the work of clearing the buffer (calls doSendEvent() )

    //! does the work of sending an event
    /*! Be aware this is an O(n^2) where n is the number of listeners for a particular event.
     *  This is because after each call to processEvent, the list of listeners could have changed
     *  so each listener much be verified before it is sent an event.  New listeners won't get the
     *  current event, but neither should listeners which have just be removed */
    void doSendEvent(const EventBase& e);

    //! Contains all the information needed to maintain a timer by the EventRouter
    struct TimerEntry {
    //! constructors an entry using the given value for next - useful for with TimerEntryPtrCmp
    explicit TimerEntry(unsigned int nxt) : el(NULL), sid(0), delay(0), next(nxt), repeat(false) {}
    //! constructs with the given values, sets next field automatically @see next
    TimerEntry(EventListener* e, unsigned int s, unsigned int d, bool r) : el(e), sid(s), delay(d), next
   //! just does the default, i'm just being explicit since there's a pointer (no deep copy!)
   TimerEntry(const TimerEntry& t) : el(t.el), sid(t.sid), delay(t.delay), next(t.next), repeat(t.repeat
   //! just does the default, i'm just being explicit since there's a pointer (no deep copy!)
   TimerEntry operator=(const TimerEntry& t) { el=t.el; sid=t.sid; delay=t.delay; next=t.next; repeat=t.
   //! will reset timer
   /*! @param d the time from now when the timer should go off (in milliseconds)
    *  @param r true if the timer should automatically repeat */
   void Set(unsigned int d, bool r) { delay=d; repeat=r; next=get_time()+delay; }
   EventListener* el;  //!< the listener to fire at
   unsigned int sid;   //!< the source id to fire with
   unsigned int delay; //!< the delay until firing
   unsigned int next;  //!< the time at which this timer will go off next
   bool repeat;        //!< if true, will reset after firing, else will be deleted
    };
    /*! @brief Used by STL to sort the timer list in order of activation time
```

```
     *  @see EventRouter::timers */
    class TimerEntryPtrCmp {
    public:
   //! Used by STL to sort the timer list in order of activation time @see timers
   /*! Since we remove NULLs before sorting, shouldn't need to check here (and I want to know if i
      *  @return (a->next<b->next) */
  bool operator()(const TimerEntry* const a, const TimerEntry* const b) const
       { return (a->next<b->next); }
    };
    typedef std::vector<TimerEntry*>::iterator timer_it_t; //!< makes code more readable
    std::vector<TimerEntry*> timers;
     //!< the list of timer entries being maintained, kept sorted by time they go active

    //! just for debugging
    void chkTimers() {
unsigned int last=0;
for(timer_it_t it=timers.begin(); it!=timers.end(); it++) {
    if(last>(*it)->next) {
dispTimers();
return;
    }
    last=(*it)->next;
}
    }
    //! just for debugging
    void dispTimers() {
std::cout << "out of order timers " << get_time() << " :\t";
unsigned int last=0;
for(timer_it_t it=timers.begin(); it!=timers.end(); it++) {
    if(last>(*it)->next)
std::cout << "##";
    std::cout << (last=(*it)->next) << '\t';
}
std::cout << std::endl;
    }

    std::vector<EventBase*> events;     //!< used to store buffered events
    bool doSendBufferLock; //!< in case of recursive calls to processEventBuffer()/doSendBuffer()
    unsigned int lastBufClear;          //!< time of last event buffer clear
    unsigned int buffertime;            //!< The time between clearings of the buffer
    /*!< <li>0 will not use the buffer, events are routed upon posting
     *   <li>1 will clear the buffer at next call to processTimers() or processEventBuffer()
     *   <li>a larger value will cause a delay of that number of milliseconds since the last clear

    //! Does the actual storage of the mapping between EventBase's and the EventListeners/EventTra
    /*! Actually only stores void*'s, so it's more general than just Listeners or Trappers */
    class EventMapper {
    public:
//! constructor
EventMapper();

void addMapping(void* el, EventBase::EventGeneratorID_t egid) {
         allevents[egid].push_back(el); } //!< Adds a listener for all events from a given event g
```

```
void addMapping(void* el, EventBase::EventGeneratorID_t egid, unsigned int sid,
          EventBase::EventTypeID_t etid); //!< Adds a listener for a specific source id
             and type from a given event generator

//! Removes a listener for all events from a given event generator
/*! Doesn't necessarily remove the vector or mapping if this was the last
               listener, use clean() to do that */
void removeMapping(void* el, EventBase::EventGeneratorID_t egid);

//! Removes a listener for a specific source id and type from a given event generator
/*! Doesn't necessarily remove the vector or mapping if this was the last
             listener, use clean() to do that */
void removeMapping(void* el, EventBase::EventGeneratorID_t egid, unsigned int sid,
              EventBase::EventTypeID_t etid);

void clean(); //!<compresses empty data structures
void clear(); //!<Resets the mapping

   //@{
   //! so stuff can tell if it even needs to bother generating an event...
   /*! ... if a tree falls in a forest, and there's no one around to see it, does it make a sound?\n
  ... if Vision sees a ball in an image, and there's no listeners, does it make an event? ;) \n
  @return true if it has any listeners, false otherwise */
   bool hasMapping(EventBase::EventGeneratorID_t egid);
   bool hasMapping(EventBase::EventGeneratorID_t egid, unsigned int sid);
  bool hasMapping(EventBase::EventGeneratorID_t egid, unsigned int sid, EventBase::EventTypeID_t etid);
//@}

   //! builds a list of all listeners which should receive the event, templated to typecast the pointers

   /*! @param e the key event
    *  @param listeners upon return, the resulting list of listeners @a e maps to\n
    *  @a listeners is not cleared prior to building, new listeners are pushed on end\n
    *  Results are in the order: all specific matches first, all generator listeners second,
         in order they were added to the EventMapper.*/
   template<class T>
   void getMapping(const EventBase& e, std::vector<T*>& listeners);

   //! Used to make sure that the specified listener exists for the given event
   /*! This is needed because after we call processEvent on a lister, we can't assume
    *  that no other listeners have been modified - one listener could cause another
    *  to turn off.  If that has happened, we shouldn't send the event, even if it
    *  was in the queue originally. */
   bool verifyMapping(void * listener, const EventBase& e) { return verifyMapping(listener,
         e.getGeneratorID(),e.getSourceID(),e.getTypeID()); }
bool verifyMapping(void * listener, EventBase::EventGeneratorID_t egid,
              unsigned int sid, EventBase::EventTypeID_t etid);

    protected:
//! a mapping from source IDs (unsigned ints), each to a vector of pointers to listeners
/*! main use in filteredevents @see filteredevents */
typedef std::map<unsigned int,std::vector<void*>,std::less<unsigned int> > SIDtoListenerVectorMap_t;
```

```
//! an array of vectors of pointers to listeners... in other words, a vector of
               listener pointers for each generator
std::vector<void*> allevents[EventBase::numEGIDs];
//! not for the faint of heart: a matrix of mappings to vectors of pointers to listeners
SIDtoListenerVectorMap_t* filteredevents[EventBase::numEGIDs][EventBase::numETIDs];

    private:
EventMapper(const EventMapper&);           //!< this shouldn't be called...
EventMapper operator=(const EventMapper&); //!< this shouldn't be called...
    };

   EventMapper trappers;//!< A mapping of which EventTrapper's should get a chance to trap the eve
   EventMapper listeners; //!< A mapping of which EventListener's should receive events

};

//! a global router for cross communication, probably the most common usage, although
 perhaps there may be times you'd rather have multiple event routers for smaller sections
extern EventRouter * erouter;

/*! @file
 * @brief Describes EventRouter class, for distribution and trapping of events to listeners
 * @author ejt (Creator)
 *
 * $Author: ejt $
 * $Name: tekkotsu-1_5 $
 * $Revision: 1.12 $
 * $State: Rel $
 * $Date: 2003/10/03 03:41:01 $
 */

#endif
```