Technical University of Crete, Greece
Department of Electronic and Computer Engineering

Player Behavior and Team Strategy in SimSpark Soccer Simulation 3D

Georgios Methenitis

Thesis Committee
Assistant Professor Michail G. Lagoudakis (ECE)
Assistant Professor Georgios Chalkiadakis (ECE)
Professor Miron Garofalakis (ECE)

Chania, August 2012

Πολυτεχνειο Κρητης

Τμημα Ηλεκτρονικων Μηχανικων και Μηχανικων Υπολογιστων

Συμπεριφορά Παικτών και Στρατηγική Ομάδας για το Πρωτάθλημα RoboCup 3D Simulation

Γεώργιος Μεθενίτης

Εξεταστική Επιτροπή

Επίκουρος Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)

Επίκουρος Καθηγητής Γεώργιος Χαλκιαδάκης (ΗΜΜΥ)

Καθηγητής Μίνως Γαροφαλάκης (ΗΜΜΥ)

Χανιά, Αύγουστος 2012

Abstract

Every team which participates in a game, requires both individual and team skills in order to be successful. We could define individual skills as the ability of each member of the team to do actions which are going to be productive and close to the team's goal. On the other hand, we could define team skills as the actions of individuals, brought together for a common purpose. In essence, each person on the team puts aside his or her individual needs to work towards the larger group objective. The interactions among the members and the work they complete is called teamwork. So, when we are talking about such a team sport like soccer both individual and team skills are in a big need. For human teams, these two skills exist and be improved over time, however, for robot teams these skills are completely in absence. This thesis describes, how we create a team's framework step by step, for the Robocup's soccer simulation league. This league includes all the classic Artificial Intelligence's and robot problems such as, perception, localization, movement and coordination. First of all, every player in the team should percept his environment and has a reliable imaging of his surroundings. If he does, then he should be able to locate his actual position in the field which is very important issue in robotic soccer games. Nothing could be accomplished by a soccer team if players have a poor movement in the field. There must be stable and fast movements by the players, this can be prove to be crucial in a soccer game. Last but not least, is the coordination, which is a major's importance factor in a multi-agent system like this. Agents, should be able to coordinate their actions through communication or other effectors in order to work as a team and towards the team's success. In this thesis, we are coming up with an approach in each one of these problems emphasizing in agents' coordination. The whole idea is based on agents' ability to communicate with each other. Using a simple communication protocol goalkeeper gathers information from all field players and then it his responsibility to make the coordination and send them in the end of the process the actions which have been calculated to be costless and worthy for the team.

Περίληψη

Κάθε ομάδα που συμμετέχει σε ένα ομαδικό παιχνίδι απαιτεί τις ατομικές ικανότητες κάθε παίκτη ξεχωριστά αλλά και την συνολική ικανότητα της σαν ομάδα ώστε να είναι πετυχημένη. Θα μπορούσαμε να χαρακτηρίσουμε τις ατομικές ικανότητες σαν ενέργειες ατόμων οι οποίες λαμβάνουν χώρα με σκοπό να γίνουν επικερδείς για την ομάδα. Από την άλλη μεριά, οι ομαδικές ικανότητες είναι ο συνδυασμός τον επιμέρους ενεργειών κάθε παίκτη που επιφέρει κέρδος στην εκάστοτε ομάδα. Οι ενέργειες αυτές γίνονται από την πλευρά κάθε παίκτη βάζοντας στην άκρη τις προσωπικές φιλοδοξίες για την επίτευξη ενός μεγαλύτερου σκοπού. Ειδικότερα όταν μιλάμε για ένα άθλημα όπως το ποδόσφαιρο, υπάρχει η απαίτηση και των δυο παραπάνω γνωρισμάτων από όλους τους παίκτες της ομάδας. Για τις ανθρώπινες ομάδες αυτό είναι κάτι τετριμμένο που υπήρχε πάντα και συνεχώς βελτιώνεται. Αλλά όταν μιλάμε για ρομποτικές ομάδες ποδόσφαιρου όλες αυτές οι ικανότητες δεν υφίσταται. Αυτή η διπλωματική εργασία περιγράφει την δημιουργία βήμα-βήμα ολοκλήρου του πλαισίου για την κάλυψη των αναγκών μιας ρομποτικής ομάδας ποδόσφαιρου για το επίσημο πρωτάθλημα προσομοίωσης Robocup. Αυτό το πρωτάθλημα περιέχει όλα τα κλασσικά προβλήματα της τεχνητής νοημοσύνης αλλά και των ρομποτικών συστημάτων όπως η αντίληψη, το πρόβλημα του εντοπισμού, η κίνηση και η συνεργασία. Αρχικά κάθε παίκτης πρέπει να είναι ικανός να αντιλαμβάνεται το περιβάλλον του και να έχει μια αξιοπρεπή απεικόνιση των πραγμάτων που βρίσκονται γύρω από αυτό. Αν είναι ικανός να το κάνει, τότε θα πρέπει να βρίσκει την θέση του στο γήπεδο, κάτι που είναι πολύ σημαντικό στο ρομποτικό ποδόσφαιρο. Τίποτα δεν θα μπορούσε να επιτευχτεί αν δεν υπήρχε η κίνηση, πρέπει να υπάρχουν σταθερές και γρήγορες κινήσεις που θα βοηθήσουν τα μέγιστα και είναι ιδιαίτερα σημαντικές σε τέτοιου τύπου αγώνες. Τέλος, είναι η συνεργασία που επιτυγχάνεται μέσω της επικοινωνίας η άλλων ενεργειών. Οι πράκτορες -ρομπότ- πρέπει να είναι σε θέση να συνεργάζονται μεταξύ τους ώστε να μπορούν να πετυχαίνουν το καλύτερο για την ομάδα τους. Σε αυτή την διπλωματική εργασία ερχόμαστε με μια προσέγγιση που δίνει απαντήσεις στα παραπάνω ερωτήματα και δίνει έμφαση στον τομέα της συνεργασίας. Η όλη ιδέα βασίζεται στην ικανότητα των παικτών να επικοινωνούν μεταξύ τους. Με ένα απλό και εύχρηστο πρωτόκολλο επικοινωνίας ο τερματοφύλακας μαζεύει πληροφορία από τους υπόλοιπους παίκτες της ομάδας και μετά είναι δική του ευθηνή ο καταμερισμός ενεργειών που θα προσδώσουν στην ομάδα το μέγιστο κέρδος με το μικρότερο δυνατό κόστος.

Contents

List of Figures

LIST OF FIGURES

Chapter 1

Introduction

What will happen if a robot be put into a soccer field? It is obvious for everyone to understand that nothing is going to happen. This occurs due to the fact that a machine such as a robot should be programmed to percept its surroundings and act just like a human soccer player. Therefore, everything in the robot's world, start from the absolute zero.

Communication is a much need skill, which is met everywhere around us from a human cell and micro-organisms to vastly advanced animals as humans. Even the nature itself reinvents itself if a communication channel is broken down. Humans since the advent of their history form all kinds of groups striving to achieve a common goal, especially , in our time and age , for teams participating in games, where success can only be achieved through collaborative and coordinated efforts. Teams lacking means of communication are doomed to act simply as a collection of individuals with no added benefit, other than the multiplicity of individual skills. For human teams, communication is second nature; it is used not only for teamwork in games, but also in all aspects of life, and takes a multitude of different forms (oral, written, gestural, visual, auditory). Robots can hardly replicate all human communication means, since these would require extremely accurate and robust perceptual and action abilities on each robot. Fortunately, most modern robots are capable of communicating over data networks, an ability which is not available to their human counterparts. However, exploiting such networking means for team communication purposes in an efficient manner that

does not drain the underlying resources and provides transparent exchange of information in real time is a rather challenging problem.

This thesis describes a distributed communication framework for robotic teams, which was originally developed for the Standard Platform League (SPL) of the RoboCup (robotic soccer) competition, but can easily serve any other domain with similar communication needs. During a RoboCup game, robot players occasionally need to share perceptual, strategic, and other team-related information with their team-mates in order to coordinate their efforts. In addition, during development and debugging, human researchers need to be in direct contact with their robots for monitoring and modification purposes. Typically, these kinds of communication take place over a wireless network. The naive solution of maintaining a synchronized copy of each robot's data on each node on the network is both inefficient and unrealistic for such real-time systems.

Our proposal suggests a distributed and transparent communication framework, called *Narukom*, whereby any node of the network (robot or remote computer) can access on demand any data available on some other node in a natural and straightforward way. The framework is based on the publish/subscribe paradigm and provides maximal decoupling not only between nodes, but also between threads running on the same node. As a result, *Narukom* offers a uniform communication mechanism between different threads of execution on the same or on different machines. The data shared between the nodes of the team are stored on local blackboards which are transparently accessible from all nodes. In real-time systems, such as robotic teams, data come in streams and are being refreshed regularly, therefore it is important to communicate the latest data or data with a particular time stamp among the robots. To address such synchronization needs, we have integrated temporal information into the meta-data of the underlying messages exchanged over the network. *Narukom*'s distributed nature and platform independence make it an ideal base for the development of complex team strategies that require tight coordination, but also for the distribution of resource-intensive computations, such as learning experiments, over several (robot and non-robot) nodes.

1.1 Thesis Outline

Chapter 2 provides some background information on the RoboCup Competition and the underlying technologies used to developed *Narukom*. In Chapter 3 we demonstrate some important aspects of the problem of communication across robots. Continuing to chapter **??**, where the core ideas and an outline of the architecture of our proposal is discussed. Moving on to chapter 5, a thorough discussion about optimization, implementation design and decisions taken during the development of *Narukom*. In Chapter 6 a discussion on the results is taking place by providing several experiments in order to evaluate our work. The following chapter 7, presents similar systems developed by other robocup teams including a brief comparison between those systems and ours. Future work and proposals on extending and improving our framework are the subject of the chapter 8. The final chapter 9 serves as an epilogue to this thesis, including a small overview of the system and some long terms plans about *Narukom*.

Chapter 2

Background

2.1 RoboCup Competition

The RoboCup Competition, in its short history, has grown to a well-established annual event bringing together the best robotics researchers from all over the world. The initial conception by Hiroaki Kitano [1] in 1993 led to the formation of the RoboCup Federation with a bold vision: "By the year 2050, to develop a team of fully autonomous humanoid robots that can win against the human world soccer champions".The uniqueness of RoboCup stems from the real-world challenge it poses, whereby the core problems of robotics (perception, cognition, action, coordination) must be addressed simultaneously under real-time constraints. The proposed solutions are tested on a common benchmark environment through soccer games in various leagues,with the goal of promoting the best approaches, and ultimately advancing the state-of-the-art in the area.

2.2 RoboCup Leagues

Beyond soccer, RoboCup now includes also competitions in search-and-rescue missions (RoboRescue), home-keeping tasks (RoboCup@Home), robotic performances (RoboDance), and simplified soccer leagues for K-12 students (RoboCup Junior). Broadening the research areas where RoboCup focuses, was a very interesting and clever addition, which enables all the more scientists and researchers

combine their expertise in order to solve real world problems. A lot of progress has been made so far in many disciplines of robotics and RoboCup has been established in one of the most important events around the world.

2.3 RoboRescue

RoboRescue initiated from the need of people to create robots capable of operating in hostile or inaccessible environments by humans. This area is very motivating in terms of helping the humanity while promoting robotics. Multi-agent team work coordination, physical robotic agents and rescue strategies are all being tested in real or simulated environments (Figure 2.1), using sensors including cameras, temperature sensors, CO_2 sensors and other that allow for fast and efficient human body localization.

Figure 2.1: RoboRescue environments can become extremely hostile for robots.

2.4 RoboCup@Home

The RoboCup@Home league aims to develop service and assistive robot technology with high relevance for future personal domestic applications. It is the largest international annual competition for autonomous service robots and is part of the

RoboCup initiative. A set of benchmark tests is used to evaluate the robots' abilities and performance in a realistic non-standardized home environment setting (Figure 2.2).

Most of the research lies in many domains including Human-Robot-Interaction and Cooperation, Navigation and Mapping in dynamic environments, Computer Vision and Object Recognition under natural light conditions, Object Manipulation, Adaptive Behaviors, Behavior Integration, Ambient Intelligence, Standardization and System Integration.

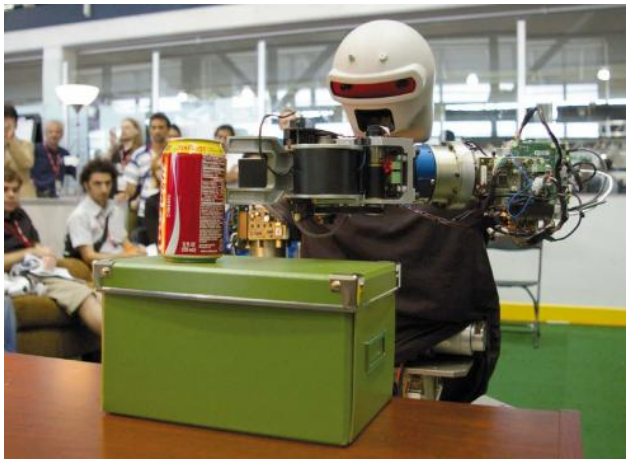

Figure 2.2: Robocup@Home represents a social aspect of robotics interacting with people.

2.5 RoboCup Junior

Robocup Junior is one of the smaller and less competitive, in terms of antagonism sub-domain of RoboCup, which is mainly intended in the preparation of gifted children interested in robotics. It is very important to understand that this competition has nothing to be jealous of the other leagues, but in fact share the same vision and the required dedication to excel.

2.5.1 Dance

RoboDance might one say to be the most amusing competition of RoboCup, where robots are performing in front of their audience. Dances, most of times, are innovative and unique. Developers are being judged on the novelty of motions, the cooperation of the robots, and finally the synchronization of the robot motion according to the music.

2.5.2 Rescue

Rescue in the Junior league, is a simplified version of RoboRescue and is limited in the line-following problem. Each team has to qualify from a number of tracks whose difficulty gradually increases. The trials are judged on the duration of the track completion and on the agent's behavior in misleading parts of the track, such as line intersections, or line gaps.

2.5.3 Soccer

The soccer competition is played by two cylindrical robots which share the same features and play soccer in a box or in a table surrounded by low walls. This is by far the most exciting competition in Robocup Junior and the most demanding.

2.6 RoboCup Soccer League

The RoboCup Soccer League, is the domain with the most fans. In this league researchers combine their technical knowledge in order to prepare the best robotic soccer team among other universities.

2.6.1 The Standard Platform League

Standard Platform League (SPL) of the RoboCup competition is the most popular league, featuring two to four humanoid Aldebaran Nao robot players in each team. This league was formerly known as the Four-Legged League with Sony Aibo robots, which were replaced in 2008 by Aldebaran Nao (Figure 2.3). Games take place in a $4m \times 6m$ field marked with thick white lines on a green carpet. The

two colored goals (sky-blue and yellow) also serve as landmarks for localizing the robots in the field. Each game consists of two 10-minute halves and teams switch colors and sides at halftime. There are several rules enforced by human referees during the game. For example, a player is punished with a 30-seconds removal from the field if he performs an illegal action, such as pushing an opponent for more than three seconds, grabbing the ball between his legs for more than three seconds, or entering his own goal area as a defender.

Figure 2.3: Standard Platform League game in Robocup 2008(Opponents should be in different colors, but there was a lack of Nao robots in that event due to malfunctions)

The main characteristic of the Standard Platform League is that no hardware changes are allowed; all teams use the exact same robotic platform and differ only in terms of their software. This convention results to the league's enrichment of a unique set of features: autonomous player operation, vision-based perception, legged locomotion and action. Given that the underlying robotic hardware is common for all competing teams, research effort has been focused on the development of more efficient algorithms and techniques for visual perception, active localization, omni-directional motion, skill learning, and coordination strategies.

During the course of the years, one could easily notice a clear progress in all research directions.

2.6.2 Simulation League

Every year there is a number of simulation games taking place in RoboCup competitions. These include 2D soccer games, where teams consist of 11 agents providing to developers with a perfect multi-agent environment to tune and benchmark their solutions. 3D simulation games exist as well; usage of physics engines demand more realistic approaches.

Simulators offer the ability to control the amount of "negative" realism added in these environments; thus, it is a great way to allow researchers work focusing in multi-agent cooperation approaches and other state-of-the-art algorithms, abstracting from real-world problems (gravity, forces etc).

In RoboCup 2008, three of the most important simulation leagues, were the official RoboCup Simulation League run in an open source simulator, the Microsoft Robotics Studio competition, and the Webots RoboStadium.

2.6.3 Small Size League

A Small Size robot soccer game takes place between two teams of five robots each. Each robot must fit within an 180mm diameter circle and must be no higher than 15cm, unless they use on-board vision. Robots play soccer on a 6.05m long by 4.05m wide, green carpeted field with an orange golf ball. Vision information is either processed on-board the robot or is transmitted back to the off-field PC. Another off-field PC is being used to communicate referee commands and position information to the robots, when an extra camera mounted on top of the field serves as the vision sensor. Typically, off-field PCs are used in the coordination and control of the robots. Communication is wireless and typically use dedicated commercial FM transmitter/receiver units[1].

[1] http://small-size.informatik.uni-bremen.de/rules:main

2.6.4 Middle Size League

Middle Size is more competitive and demanding, having the largest field dimensions among other leagues (Figure 2.4). Two teams of mid-sized robots consist of 5 players each, with all sensors on-board play soccer on a field of $18m \times 12m$, whereas relevant objects are distinguished by colors only. Communication among robots (if any) is supported on wireless communications. Once again, no external intervention by humans is allowed, except to insert or remove robots in/from the field. These robots are the best players far among other leagues. The robots' bodies are heavy enough having powerful motors, heavy batteries, omni-directional camera, and a full laptop computer running in every robot; characteristics that make this league a great domain for research.

Figure 2.4: Middle size league game in Robocup 2008.

2.6.5 Humanoid League

The Humanoid League is one of the most dynamically progressing leagues and the one closest to the 2050's goal. In this league, autonomous robots with a human-like body plan and human-like senses play soccer against each other. In addition

to soccer competitions, technical challenges take place. The robots are divided into two size classes: KidSize (30-60cm height) and TeenSize (100-160cm height). Dynamic walking, running, and kicking the ball while maintaining balance, visual perception of the ball, other players, and the field, self-localization, and team play are among the many research issues investigated in this league. Several of the best autonomous humanoid robots in the world compete in the RoboCup Humanoid League[1].

[1]http://www.tzi.de/humanoid/bin/view/Website/WebHome

Chapter 3

SimSpark

3.1 Soccer simulation

3.2 Server

3.3 Simulation Update Loop

3.4 Network Protocol

3.5 Monitor

3.6 Perceptors

3.7 Effectors

3.8 Agents

3.9 Model

Chapter 4

Agent components

4.1 Connection

Here will be placed a general statement about connection

4.1.1 Incoming messages

Here will be placed a detailed explanation

4.1.2 Outgoing messages

Here will be placed a detailed explanation

4.2 Motions

Here will be placed a general statement about motions in simspark

4.2.1 .motion

Here will be placed a detailed explanation

4.2.1.1 files

Here will be placed a detailed explanation

4.2.1.2 Controller

Here will be placed a detailed explanation

4.2.2 .xml

Here will be placed a detailed explanation

4.2.2.1 files

Here will be placed a detailed explanation

4.2.2.2 Controller

Here will be placed a detailed explanation

4.3 Localization

Here will be placed a detailed explanation

4.3.1 Filtering

Here will be placed a detailed explanation

4.4 Actions

Here will be placed a general explanation

4.4.1 Simple

Here will be placed a detailed explanation

4.4.2 Complex

Here will be placed a detailed explanation

4.4.3 Vision

Here will be placed a detailed explanation

4.4.4 Head movement

Here will be placed a detailed explanation

4.4.5 Moving objects

Here will be placed a detailed explanation

4.4.6 Obstacle perceptor

Here will be placed a detailed explanation

4.4.7 Obstacle avoidance

Here will be placed a detailed explanation

Chapter 5

From Theory To Practice

This section describes all the optimizations , implementation decisions and details of Narukom. The primary goal of Narukom is to be used as a communication framework in robotic environments, so most of our choices were in favour of real-time embedding systems and specifically for the Standard Platform League competition. Consequently, Narukom does not fit in all problem domains.

5.1 Understanding the basics

5.1.1 Messages

The exchanged data between the threads are protocol buffers messages, defined by the user. Narukom requires to include in each message definition some attributes that are of outmost importance in order for Narukom to work properly and handle every message. As metioned before, every message definition in Narukom takes the following form with five mandatory attributes in its preamble:

```
message xxxMessage{
    required string host      = 1 [default = "localhost"];
    required string publisher = 2 [default = "" ];
    required string topic     = 3 [default = "global"];
    required int32  timeout   = 4 [default = 0];
    required string  timestamp = 5 [default = ""];
    < attributes specific to the xxxMessage type >
```

```
}
```

Narukom includes a definition of a basic structure called BasicMessage in order to deal with all the messages. Each attribute serves a purpose,

- `host,publisher` attributes are used in order to distinguish messages from the between network nodes,
- `topic` is the topic in which each message should be published and
- `timeout,timestamp` are the temporal information needed by *Narukom* for synchronization purposes.

5.1.2 MessageBuffer

This structure is a thread safe and as its name suggests, is a buffer in which protocol buffer messages are stored. Apart from the simple vector to hold the messages, the structure has a string attribute named `owner` ,which denotes the name of the owner of the buffer. It must be noted that message buffer is implemented as monitor.

5.2 Publish/Subscribe system in *Narukom*

Publish/Subscribe implementation is shown in figure 5.1.

Both publishers and subscribers have one message buffer through which the exchange of data takes place. Message queue, periodically checks all the publishers' buffers and copies them to the buffers of all interested subscribers. This approach minimizes the race conditions when accessing messages , on the cost of high decoupling between the publishers and subscribers , which in real-time systems could be source of undesirable behavior. In the next section , it is described how this problem is addressed. Continuing with the publish/subscribe system, in order for classes use the infrastructure provided by Narukom , should inherit from either `Publisher` or `Subscriber` class (or both). Although , there is a default implementation of the interfaces for both classes, it is recommended

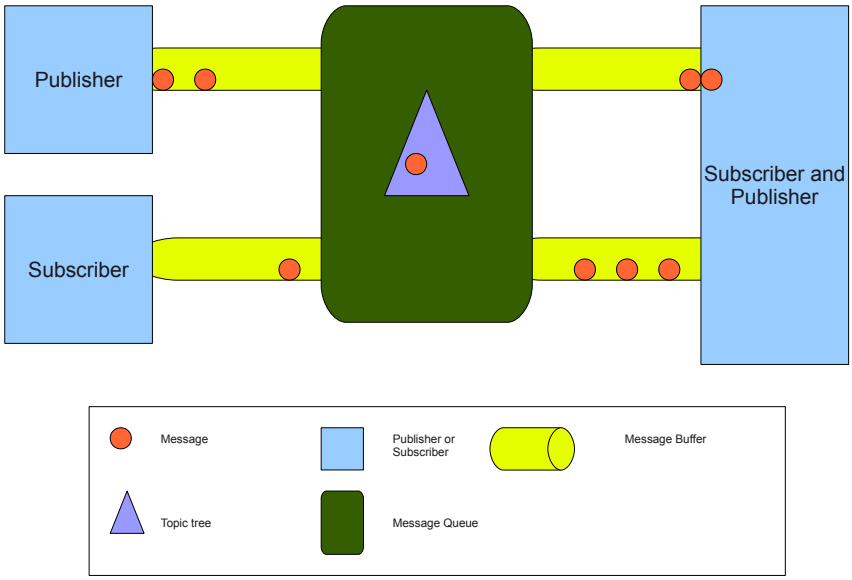

Figure 5.1: Narukom's publish/subscribe implementation.

derived classes not to rely on the default implementation as it might have undesirable effects on the behavior of their system. The `Publisher` class requires the implementation of the method:

```
void publish( google::protobuf::Message* msg)
```

The default implementation of this method just adds the *msg* message to the buffer, which in the majority of the scenarios would be a sufficient behavior for sending a message. On the other hand , classes derived from `Subscriber` class is strongly encouraged to provide their own implementation of the method: [1]

```
void process_messages()
```

5.2.1 Message Queue

Message queue performance affects directly the overall performance of our system, thus several optimizations have been made to this class. The most important

[1] The default implementation of the process_messages() is provided just as a reference.

optimization is that the internal topic tree is used as a reference structure, while the actual data structure used is a hash table on all the topics the tree included. The configuration of the tree is done via an XML file on each node, on the initialization of the message queue the file is parsed and the tree is created. The hash table gives us fast retrieval on the list of subscribers to which the arrived message must be delivered. Furthermore, when it comes to reading the message's meta-data, a cast to `BasicMessage` type is used, overriding the normal way of accessing data in protocol buffers. It must be stressed that this approach is used ONLY for reading and ONLY in **Blackboard and Message Queue** classes. **under no circumstances is recommended as the normal way for accessing the meta-data required by Narukom**. This is considered a heavy optimization which could stop working after a new release of protocol buffers. Last but not least, instead of keeping pointers to the publisher or subscriber , a message buffer pointer is used in order to avoid the small overhead of calling an extra method every time `Message Queue` needs to access a buffer. As insignificant as these optimizations may seem , in (SPL) ,where software makes the difference, they could decide the winner of a game.

5.3 Blackboard and Synchronization

The implemented publish/subscribe system maximizes the decoupling between publishers and subscribers, thus a lot synchronization issues might arise between threads running locally on a node. this problem is dealt, by adding two attributes in the meta-data of each message . Narukom introduces some conventions for each message, when a message is published a timestamp is assigned (at least in the default implementation of publish), at the time the subscriber receives a message should check the validity of the data by ensuring that those data are not expired, on data expiration subscribers are advised to dispose those messages in order to limit the excess and unusable information on a node. It should be noted that those are convention of Narukom that are used internally and cannot be imposed on subscribers, each subscriber is free to comply or not with them. Narukom's blackboard utilizes the time information to minimize memory allocation and provide synchronization services to the higher level system. Blackboard

implementation creates two indexes over the stored messages, one to provide fast access for data requests and another to provide fast cleanup time. The latter index is an ordered list in terms of absolute timeout calculated when new messages are stored in the blackboard. Frequently, the blackboard cleans up all the expired data.

5.4 Communication across nodes

Communication between the nodes of Narukom is achieved through channels, which are well-defined pairings between actuators (motors, speakers, network adapter) affecting the robot's environment, whereas sensors (camera, microphones,network adapters) sense the robots environment. Narukom was designed in order to be able to utilize all the individual channels, to tackle with the wide range of features and integrate those channels in the framework catalog module introduced.

5.5 Catalog Module

Catalog module could be characterized as the address book of each node in Narukom , since it holds all the contact information of all the reachable peers by the node itself. Furthermore, Catalog is responsible for creating and destroying the various channels that the node is able to use in order to communicate with other nodes. The idea behind this module is that in distributed environments a node must be aware of all the available nodes and what type of service they provide. In detail, the information stored in a catalog module are names of all reachable peers, through which channels are these nodes available and which kind of data are interested in. Moreover, catalog holds a list of all the usable channels on the node and has the ability to start, stop or filter the type of messages that are sent through each channel. The filtering is done by sending commands that a channel can interpret into one of the above functions. These commands are a protocol buffer message with the following attributes

```
message ChannelCommand{
required string type = 1 [default = ""];
```

```
required string element = 2 [ default = "none"];
}
```

There are four different types of commands

- `subscribe`, when this type of message is received, a channel must subscribe on a certain topic.
- `unsubscribe`, a channel should unsubscribe from a topic
- `enable`, this command is used to enable a specific type of message to be sent through it, by default after the subscription of a channel all the incoming messages are enabled and the
- `disable`, obviously this is the opposite of the enable command.

The attribute element is in accordance with the command type , in case of (un)subscribe , it should hold the topic , whereas when sending an enable or disable command it must be a type of message. Channels should implement a simple interface, which is:

- `bool start()`, with this the catalog modules starts a channel,
- `bool stop()` fucntion is used to stop a channel ,
- `list_of_hosts available_peers()` catalog asks for all the information about the reachable peers continuing to,
- `list_of_strings prefered_message()`, with which the catalog is informed about what channels could be produced from this channel and
- `bool process_command(const ChannelCommand cmd)` used to send a command.

These four functions are adequate for Catalog to control a channel and provide the system with all the information for a channel. Before closing this section it should be noted that all these are conventions used by Narukom to provide users a unified way of controlling the behavior of a node in terms of communication

with other reachable nodes. Narukom does not require this functionality but should a programmer decides to extend Narukom's capabilities by creating a customized channel , this channel to be treated as an integrated part of the rest of the system. However, there could be a channel which does not implement the required interface, this does not raise any problem to the rest of the system as it would be treated as any other module on the system.

5.6 Network Communication

Network communication is the first and most important way for exchanging data between nodes. Protocol buffers messages are serialized and if their size makes it impossible to be sent on one message then they are chopped to network packets. The definition of the NetworkPacket structure follows:

```
message NetworkHeader
{
required  uint32 message_num = 1 ;
required uint32 packet_num =2  [default = 1];
required uint32 number_of_packets = 3[default = 1];
required string timestamp = 4 [default = ""];
required uint32 timeout = 5 [default = 1000];
required string host = 6;
required string type =7;
}

message NetworkPacket{
required NetworkHeader header = 1;
required bytes byte_buffer =2;
}
```

A network message acts as buffer for NetworkPackets. On the other end , network messages are stored to network message buffers.

5.6.1 NetworkChannel

NetworkChannel class is a class which encapsulates all the communication functionality provided by any network network channel (i.e. TCP based,UDP based, etc.). This class is a derived class of Channel and provides implementation for all the methods of its base class. In addition, introduces new functionality provided mainly in network such as, resolve host or notification on events such as sent or receive timeout, etc.

5.6.2 UpdMulticastChannel

The only channel that is so far implemented in Narukom is a simple multicast channel over udp. A sharp reader could easily devise that UdpMulticastChannel class is a child class of NetworkChannel. The implemented channel is not reliable, meaning that there is no guarantee that sending a message through this channel it would reach its destination, but provides a partial guarantee that if a part of the message is arrived on one node then the whole message should be delivered. The algorithm that ensures that a whole message will be delivered or not is:

```
receive packet from the network =>
add packet to the appropriate network message =>
Is the whole network message?
a) yes: then recreate the original message and publish it then remove it fro
b) no: if the packet is out of order send NACK Messages for all the pending
```

At fist site , it might be wrong decision to use an unreliable channel for our transmissions, but in real-time systems the described behavior is preferred than a bandwidth consuming approach (i.e. TCP). The UdpMulticastChannel subscribes by default on global topic where important information interesting for all the nodes on the network are published. Finally, the catalog module under certain circumstances (during a robocup game) initializes another UdpMulticastChannel in order to listen to the game controller.

Chapter 6

Results

The evaluation of the proposed approach was made on the following subjects:

- `Validation`: to validate that Narukom works properly we carried out experiments testing both inter-process and intra-process communication,
- `Metrics`: A variety of tests to measure the performance of Narukom,
- `Synchronization`: In order to test the synchronization capabilities of Narukom a simple application was developed, last but not least,
- `User friendliness`: A very small group of developers was asked to create a simple chat application to assess the user-friendliness of our framework.

6.1 Validation

Narukom has been tested successfully for both inter-process communication on a single node and multiple nodes using a simple threaded application which simulates a ping-pong (table tennis) game. There are two threads for the two players and two message types (`Ping` and `Pong`), one for each player. These two threads communicate through Narukom's publish/subscribe infrastructure, which is used to deliver the `Ping` and `Pong` messages between them. Apart from the two players there is a scorekeeper, another thread which is responsible for keeping the score, as its name suggests. The player who has the ball picks randomly a side (left or right) to which the ball should be sent and publishes a message to declare the

completion of his move. On the other side of the table, the other player picks the side where his racket will be placed (left or right) and listens for the opponent's move (he subscribes to opponent messages). If the side he chose for the racket matches the side to which the opponent sent the ball, the game continues with the next move. If the sides of the racket and the ball do not match, a point is awarded to the player who sent the ball and the game continues with the next move. The scorekeeper subscribes to both types of messages and continuously prints out the current score.

6.2 Metrics

Narukom to the extreme, in order to measure the performance of Narukom various tests were carried out such an approximate calculation of average time for message delivery, measuring network throughput under high traffic, framework's reliability over udp multicast.

6.3 Synchronization

In order to ensures that Narukom's synchronization capabilities are adequate to used in SPL we developed an application where two threads running on different frequencies demand previous published data from each other. In more detail, there is one thread (Fast) running approximately every 10 milliseconds and another one (Slow), which is run every second and asks for data that has already been created.

6.4 User Friendliness

Nowadays, ease of use plays an important role of in the adoption of a product, thus we decided to conduct a limited (in terms of the number of people) survey to assess the level of user friendliness of our framework. After a short presentation of Narukom , we asked a small group of developers to create a simple chat application. Afterwards , we asked them to write a brief description of their thoughts working with Narukom.

Chapter 7

Related Work

In this section we give a short overview of what other RoboCup teams have published to our knowledge about their communication system and compare their work to ours. There are various approaches for sharing information among processes on a robot and among processes on different robots. These approaches can be summarized into the following main ideas:

7.1 Shared Memory

The main idea here is to define a shared memory area on each node, which holds information about the current world model. Each module can write to and/or read data from this shared location. Under this scheme, different nodes use a simple UDP protocol to exchange important data. This is quite similar to our approach, however communication is mainly point-to-point and lacks the organization and efficiency imposed by the hierarchical topic-based publish/subscribe scheme. Furthermore, depending on the implementation, problems with corrupted data could be appear, if no lock and safety mechanisms are implemented. This idea with variations has been adopted by teams as Austin Villa [6],BreDoBrothers [7]

7.1.1 Blackboard/Shared World Model

This is a similar idea with the previous, however different components access the central storage data via well defined interfaces, providing with the mechanisms that might be absent from the approach above. Team robots share the information of their blackboard creating a world model composed of the individual robots' perceptions. There is one variation from GTCMU [**?**] where validity flags are added to the objects stored in order to declare "use with caution" when the objects are not updated for a long period of time. For intra-robot or debugging purposes custom communication systems are used over UDP or TCP. The purpose of data exchange suggests the used protocol, for example for debugging purposes TCP is used, while when in game UDP transmission is preferred. The shared model is close to Narukom's implementation of distributed blackboard, but the exchange of newly created data requires from the programmer writing code for sending and receiving those data, whereas on Narukom newly declared Messages are immediately ready for network transfer. The vast majority of robocup teams use a slight variation of this approach such as Zadeat [8], rUNSWift [**?**].

7.1.2 Message queue and Streams

This idea is advocated recently by team B-Human [9] and earlier by the German Team [10]. It is based on using message queues for constructing three different types of cross-platform communication streams: (a) inter-process communication between the cognition and the motion module, (b) debug communication between robots and remote computers for debugging purposes, and (c) information exchange between the robots. The first type is similar to Narukom's inter-process communication. The other two types are implemented differently in Narukom, but they serve the same purposes. Again, this idea lacks the organization and efficiency of Narukom. It must be noted , however that a lot of effort has been put into creating a full featured Stream library to support the required functionality, which is not present in Narukom.

7.1.3 Publisher/Subscribe

This idea is based on the publish/subscribe paradigm and has been adopted only the Austrian Kangaroos [11] team. The idea is to use a publish/subscribe architecture for exchanging data between the threads on the same machine by triggering signals in all connected modules if a subscribed memory location is altered. This design allows an IRQ-like implementation of service calls at a high system level. Network communication is done via SOAP wrapped functions calls, which generally are considered heavy compared to Narukom's approach. On the other hand, in this approach there each module a more direct access to remote data by subscribing to remote memory locations.

7.2 Overview

Generally speaking, Narukom integrates features that support any communication need in the SPL; these features are not found collectively in any other existing communication system. One may argue that Narukom's functionality could be implemented using the infrastructure provided by the proprietary NaoQi middleware offered by Aldebaran Robotics with the Nao robots. This is true, however Narukom's purpose is to provide a communication system that does not depend on proprietary technology and can be used widely on many different robotic teams, apart from Nao robots, and other real-time distributed systems. Narukom also represents our efforts in departing from the dependence on the NaoQi framework towards platform-independent customizable software architecture for robotic teams.

Chapter 8

Future Work

This is just a small , but well founded step towards our notion of the ideal communication framework for robotic teams and other similar distributed systems. Thus, there are plans for extending, adding and optimizing further the proposed framework.

8.1 Alternative Channels

8.2 Narukom outside robocup

8.3 Optimization and Debugging

Every project, no matter how well it is tested optimized during the development phase after it is released bugs, bottlenecks are found. Our intention and desire is to continue to actively develop Narukom in order to meet the needs of as many people as possible. As a result a constant procedure of optimizing , debgugin and testing is needed to ensure that Narukom is a robust, easy to use framework.

Chapter 9

Conclusion

Narukom is an attempt to address a lot of communication problems our team had during the last years in Robocup (SPL), while at the same time is a first step towards a cross-platform architecture used in robotic agents. Our journey does not end here our Narukom has, hopefully, a long way, yet to go. The first real world test is the Robocup 2010 in Singapore where both our inter-process and team communication would be based on our framework.

9.1 Out To The Real World

Narukom is based on open source projects, so the least we could do is to open source it in the near future. We believe that Narukom can contribute to open source community as a cross-platform, distributed, transparent communication framework for everyone willing to use it. I, personally, would be more than happy to assist anyone dare to use our framework. Additionally, any suggestion , comment, critiscism, imporvement and modification is at least welcomed. By open sourcing Narukom we try to achieve two goals take Narukom to the first step and encourage new coders to make available to the community of open source their code in order to advance the collaboration between programmers and avoid reinventing the wheel every now and then. Finally, I would like to thank those who spent some time using Narukom.

References

[1] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: Robocup: A challenge problem for AI. AI Magazine **18**(1) (1997) 73–85 5

[2] Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys **35**(2) (2003) 114–131

[3] Google Inc.: Protocol Buffers. (2010) code.google.com/apis/protocolbuffers.

[4] Hayes-Roth, B.: A blackboard architecture for control. Artificial Intelligence **26**(3) (1985) 251–321

[5] Petters, S., Meriçli, T.: RoboCup Game Controller Open-Source Software. (2010) sourceforge.net/projects/robocupgc.

[6] Hester, T., Quinlan, M., Stone, P.: UT Austin Villa 2008: Standing on two legs (2008) Only available online: www.cs.utexas.edu/~AustinVilla. 29

[7] Czarnetzki, S., Hauschildt, D., Kerner, S., Urbann, O.: BreDoBrothers team report for RoboCup 2008 (2008) Only available online: www.bredobrothers.de. 29

[8] Ferrein, A., Steinbauer, G., McPhillips, G., Niemüller, T., Potgieter, A.: Team Zadeat - team report (2009) Only available online: www.zadeat.org. 30

[9] Röfer, T., Laue, T., Müller, J., Bösche, O., Burchardt, A., Damrose, E., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Rieskamp, A., Schreck, A., Sieverdingbeck, I., Worch, J.H.: B-Human team report and code release

2009 (2009) Only available online: www.b-human.de/download.php?file=coderelease09_doc. 30

[10] Rofer, T., Laue, T., Weber, M., Burkhard, H.D., Jungel, M., Gohring, D., Hoffmann, J., Krause, T., Spranger, M., von Stryk, O., Brunn, R., Dassler, M., Kunz, M., Oberlies, T., Risler, M., Schwiegelshohn, U., Hebbel, M., Nistico, W., Czarnetzki, S., Kerkhof, T., Meyer, M., Rohde, C., Schmitz, B., Wachter, M., Wegner, T., Zarges, C.: GermanTeam 2005 (2005) Only available online: www.germanteam.org/GT2005.pdf. 30

[11] Bader, M., Hofmann, A., Schreiner, D.: Austrian Kangaroos team description paper (2009) Only available online: www.austrian-kangaroos.com. 31