On the construction of a robotic situation awareness support framework

Master's Thesis

Tom Schut
(s0362638)

Supervisors:
Dr. M. Saerbeck
Dr. I.G. Sprinkhuizen-Kuyper
Dr. I. van Rooij

Radboud University Nijmegen
Artificial Intelligence: cognitive science
Philips Research Eindhoven

Contents

Abstract

A field of study within human robot interaction is situation awareness, the study of making context information available to the system. When developing robotic systems, a lot of effort usually has to be made to integrate existing data perception algorithms, often developed in different languages, and use the data they provide. There is a need for a middleware that provides a way to reuse these algorithms more efficiently and offers an interface to the data. The "The Context Toolkit" situation awareness framework was adapted to investigate a solution to these problems. A configuration module was developed to allow modules to be connected automatically, and the overhead was separated from the data flow. Accordingly, the protocol was simplified to stimulate reuse and efficiency. A test case experiment was conducted with the iCat robotic framework to evaluate the system. We conclude with a discussion on considerations that have to be made when developing a situation awareness support system.

Acknowledgements

Writing a thesis proved not to be a trivial matter for me, so there are some people I would like to thank for helping me during writing this thesis. First of all I would like to thank Martin Saerbeck for supervising me as an intern at Philips Research. Also I would like to thank Ida Sprinkhuizen-Kuyper and Iris van Rooij for supervising me, as well as Gea Dreschler for tutoring me regarding the actual text of this thesis. I would like to thank Joris, Lina, Bram, Nina, Aljosja, Mark, Gokberk, Friedemann, Boris, and Berend for sharing a good time with me in Eindhoven. Lastly I would like to thank Jop, Ron, Saskia, Sanne, Anne, Marjolein, Loes, Josje, Jelmer, and of course my parents, for their moral support.

Chapter 1

Introduction

Humans are able to seamlessly interact with each other. Not only through speech, but also through other modalities as body language, touch, and even odours, a lot of information is transferred between persons. The reason for this success in communicating lies in the common ground of social interaction. Humans know how to interact with each other and what information about others is important to perceive. It is not hard for someone to grasp another persons intention by only perceiving what he is doing and the situation he's in. For a robot however, this is no trivial task.

One of the main objectives of human robot interaction is modelling natural interaction. A robot that interacts with a human can take advantage of the efficiency that can be reached through human-human interaction patterns. To be able to do this, it needs to be able to process the same data a human processes to build up a model of the world. This processing, and reasoning with the processed data, is called situation awareness. As pointed out humans are fairly good at this, but for robots, situation awareness is very hard to achieve.

Human robot interaction is the field of study concerned with the communication between humans and robots. Applications are amongst others: entertainment, education, and companion robotics. A robot's interaction with its human counterpart has to be adapted to its task, so different robots exist for different niches. Fong, Nourbakhsh, and Dautenhahn (2003) describe both morphological and behavioural design considerations for robots in different interactive environments. In our study, the iCat robot (Breemen, Yan, & Meerbeek, 2005), a caricatured social robot, is used.

A basic human robot interaction cycle consists of three steps (Murphy, 2000). First the robot is triggered by something in the world. This trigger can range from a change in the environment to an internal change of state like the passing of a certain amount of time, but can also be the start of the interaction cycle. In the second step the robot reasons with the new knowledge that it perceived. To be able to reason with input knowledge, the robot needs to construct an accurate world representation from the input it has received from its sensors and previous internal states. In the third step the robot generally produces output through an actuator, e.g. saying something to the user, but it can also

withhold from this, only changing its world representation.

The world representation has to come about by a series of algorithms that process input data and form or contribute to the representation of properties of the world (Figure 1.1). These properties do not only concern objects but they also concern sound data, environmental data and relations between objects. Everything that has to be represented on a higher level than the raw sensor data has to be done so by an algorithm that is devised to represent that single object or relation.

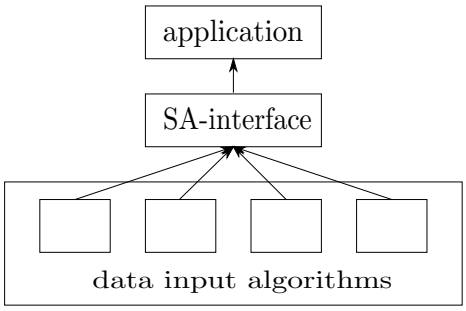

Figure 1.1: A depiction of stand alone input algorithms that have to be connected to the application by the SA-interface.

To represent data on various levels of abstraction, a system that provides a situation awareness interface will have to be equipped with a language that is able to describe these data types. The application developer also needs to use a descriptive language to address the situation awareness framework. This language, when assumed the same language is used for both actions, needs to be designed powerful enough to minimize unwanted and redundant data flow. In Chapter 2 we will describe further constraints on this language.

When a world representation is formed by using data from different algorithms and it is represented in a sufficiently expressive language, the data will have to be provided to other modules in the robotic system. To achieve this the system will have a need for an interface that unifies and provides the connection to all input algorithms.

The domain of robotics suffers from shortcomings in perception, reasoning and action (Dautenhahn, Ogden, & Quick, 2002). The perceptual problems are concerned with perceiving properties of the world humans take for granted as being perceived easily. For example, humans can easily identify what someone is doing, or what people are talking about. Robotic perception is nowhere near being similar to its human equivalent. Although there are working applications in computer vision that deal with face, body or gesture recognition (Wren, Azarbayejani, Darrell, & Pentland, 1997), these are not nearly as high levelled as human perception. The same problem arises in speech understanding (Jurafsky & Martin, 2000).

Over the years computer science researchers have developed numerous algorithms that focus on various forms and levels of perception. These algorithms are in general available for developers to use in their specific systems. However, every algorithm needs to be tailored to function inside a new host system. This

means that an algorithm needs to be adapted and sometimes even completely transcoded into another programming language. Usually, developers choose to reimplement the algorithm (Weihe, 1997).

A system that aims to reuse existing algorithms and use them in a constructive way is bound to run into problems regarding the language dependency of existing solutions as well as the lack of a suitable interface for the data produced by the reused algorithms. We also identify the need for an easy to use framework to use these algorithms in. Because once algorithms are made stand alone and language independent, a user should not have to configure them by hand every time they are used. With this comes a requirement for configuring and connecting co-dependent data-processing algorithms.

Solutions exist in providing an interface for data processing algorithms, but these are often intended as a situation awareness framework that is based on a single programming language (Hawes et al., 2007; Cote et al., 2004). Other - language independent - solutions exist, but these often use dedicated libraries the algorithms have to be linked against. Since we will focus on support frameworks for situation awareness, we will investigate the possibility of providing a framework which includes automatic configuration, a universal language which can handle diverse data types, and an interface component that is as concise as possible in its communication.

1.1 Research questions

In this work we address a main problem of robotic perception and reasoning research, the reusability of algorithms. As noted above, algorithms are being developed for various applications, but are often reimplemented instead of reused. We pose the following research questions:

1. Can we construct a situation awareness framework that facilitates the reuse of algorithms without restricting them to a single programming language or system?
2. Can we construct a situation awareness framework that provides an easy to use interface to the application developer but preserves performance efficiency?
3. Is it possible to handle all necessary data-types while preserving a clear communication protocol?

We have translated these research questions into requirements for a situation awareness framework which we have constructed. The framework facilitates the reuse of these algorithms, and provides a robotic application developer with a way to access the data produced by these algorithms. Our focus for this framework is on a robotic application, and we have tested the system with the iCat robotic research platform (Breemen et al., 2005).

The layout of this thesis will be as follows. In Chapter 2 we will give an introduction to the theory of situation awareness. In Chapter 3 we will list the

current research in the field of robotic SA. In Chapter 4 we will list our requirements for the implementation. In Chapter 5 we will describe the adaptations we have made to an existing situation awareness framework to meet our requirements. In Chapter 6 we will evaluate our implementation by the requirements of Chapter 4 and we will describe a case study that was conducted in the setting of a robotic experiment with the iCat (Breemen et al., 2005) robotic platform. Finally, in Chapter 7 we will present a number of applications of the implementation as suggestions for future research, and conclude on conceptual difficulties a SA framework poses.

Chapter 2

Background

In this chapter we will set out the definition of situation awareness (SA) we will use and the implications this has on the type of framework we will develop. The definition of situation awareness varies widely across fields of interest. In robotics SA is often defined as the perception of the context an agent (robot) is in (Drury, Scholtz, & Yanco, 2003). Behavioural researchers prefer a definition that includes the processing of perceptual and internal data (e.g. Endsley and Garland (2000a)).

Endsley and Garland (2000b) describe situation awareness [1] as a process consisting of three levels: perception, comprehension and projection. More generally put, they define situation awareness as:

> knowing what goes on around you. (p. 2)

Dey and Abowd (2000) define context in more operational terms:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. (p. 4)

Schilit and Theimer (1994) define context awareness as:

> The ability of an [agent] to discover and react to changes in the environment he is situated in. (p. 1)

It is obvious that this definition does not only describe the perceptual side of SA but also goes into the action that should be taken to prove situation awareness.

We will use the definition proposed by Dey and Abowd (2000), because an operational definition of situation awareness is more relevant for the development of an SA application. By this definition, situation awareness consists of the

[1] The terms "situation awareness" and "context awareness" will be used as synonyms in this thesis.

sensing of information that is relevant to the system. This indicates that it is necessary to decide what is relevant to the system. Because situation awareness is by definition very context dependent, and the context that applications are deployed in will vary, a situation awareness system should be able to deal with a wide scale of information sources within a wide scale of situations.

Situation awareness does not mean sensing everything that can be sensed. As will be indicated in our requirements the system requesting data from the SA framework will be able to exert control over the system by selecting data sources which are of interest. So not every input will be considered of importance to the system.

To model natural interaction, it may be necessary to model human behaviour. This includes the human ability to act in an unstructured environment, which is done by reasoning with high level representations of (objects and relations) in the environment. To be able to do this, a lot of common sense knowledge is necessary, a problem which is generally indicated as the frame problem (Haselager, 1997, p. 62). It is hard to represent everything that is necessary to make a good decision when reasoning in a high level manner. For example, to determine why somebody is scared by a big growling dog, one has to know that dogs can bite, and that big dogs bite harder, that animals growl when they are mad, and that dogs are animals. These relations have to be learned by or be built into an intelligent system, and thereby need to be represented as contextual information.

A general type of SA which is able to comprehend all possible information about the world is at this point not considered a possibility because firstly recognition algorithms are not able to acquire all desired data and secondly, as explained in the previous paragraph, an all encompassing reasoning system is not yet available. Constructing this combined system would therefore be outside the scope of this thesis. In this paper we will therefore focus on an SA support framework. We define an SA support framework as a middleware that facilitates the collecting of (relevant) context information. This means the system or individual components should be able to control data streams and some way decide what is relevant.

Figure 2.1 depicts the general setup of a system with a situation awareness support framework in place. We define the application as the program requesting context data from the *SA middleware*. The *application developer* is thus someone programming the robotic behaviour, and relying on the context data to do so. The *sensor algorithms* are algorithms that process the data in some way. This ranges from filtering noisy data to composing a data structure from several input sources.

Any SA framework will have to utilize a language which describes all data inside the framework. Designing this language is no easy task since every type of data is best described in its own format. it has to be possible to query modules for very specific data, which adds to the complexity of the descriptive language.

We will go into the range of data types and sources further in Chapter 4, and describe the system language itself in Chapter 5.

To summarize, our definition of SA, the definition of Dey and Abowd (2000), is situated between data perception and reasoning. Though in psychology SA

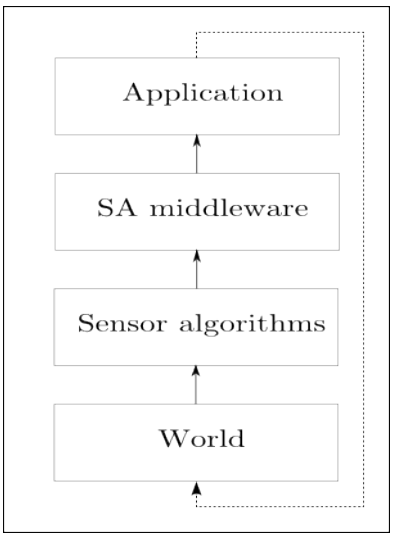

Figure 2.1: The place of an SA support framework in the perception-action cycle. The dashed line represents the indirect influence of the application on the environment through actuators.

includes reasoning, we aim to provide a framework that enables the application developer (reasoning programmer) to acquire data in an intelligent way, with as little effort as possible. This has to be done without losing data of interest. In the next chapter we will describe related work, and in Chapter 4 we will indicate to which features of SA we will contribute.

Chapter 3

Related work

In this chapter we will review existing situation awareness systems. We will first describe which different classes of context awareness systems are available in literature. After this we will list several systems that are related to our work.

Applications of context awareness are found in a wide range of fields. Fields of interest for context awareness are: location determination (Want, Hopper, & Gibbons, 1992), the sharing of information in groupware (Gutwin & Greenberg, 2002), but primarily applications are developed to make information sources accessible to the user in a discrete environment on e.g. mobile applications (Dey, 2000; Johanson, Fox, & Winograd, 2002a). We will however focus on robotic perception situation awareness.

There are a number of systems that work with situation awareness of which some are more general purpose while others are more specific to an area of development. There are two main dimensions systems can be distinguished on. The first dimension is the purpose of the system (see Section 3.1), which ranges from a general purpose intelligence framework to a context awareness facilitating middleware. The second dimension is the structure of the system (see Section 3.2), which ranges from a centralized to a distributed system. Centralized systems can also be called blackboard systems since they use one main data structure to store all data in. A third dimension, which we will view as a separate class, (Section 3.4), ranges from systems that are specialized on robotic applications to systems that are suitable for a more general application.

Purpose	**Structure**
General purpose intelligence	Distributed
Context awareness facilitory middleware	Centralized

Table 3.1: The dimensions systems will be classified on

3.1 Purpose

The first dimension is the purpose of the system. In this dimension the distinction can be made between architectures designed specifically for (robot) intelligence and architectures designed for context determination. On this dimension the first class of systems places many restrictions upon its individual components since all components are required to contribute to a specific goal. These systems thus have an extensive ontology to be able to exchange not only data but also instructions and plans with other modules. This kind of system is concerned with as well perception as reasoning and action, which means they are generally deployed as standalone systems for robotic applications or research environments. Furthermore, because a more general approach is adopted, individual modules are required to be written in a common programming language in constructs utilized by the entire system. Examples of these systems are Cosy (Hawes et al., 2007) and ACT-R (Anderson, 1993).

The second class of systems can be seen as middleware. Systems in this class leave computation inside the individual modules. The main restriction upon modules is on the way to communicate, not on how to reason or which programming language to use. Although many of these systems are distributed (e.g.(Brooks, Kaupp, Makarenko, Williams, & Orebaeck, 2007; Dey & Abowd, 2000)), some (e.g. (Johanson et al., 2002a)) have a centralized database.

3.2 Structure

In the second dimension the distinction can be made between centralized and distributed systems. Within this distinction one can also find two classes of distribution, data and message distribution. The first is concerned with the way data is stored, the second with the way communication is done within the network. For now we focus on data distribution.

Centralized systems (e.g. IROS (Johanson et al., 2002a)) are composed around a central component which stores all data and acts as a central hub for data distribution. This central component is also the main data-interpreter and reasoner. Reasoning is usually done through first order logic, so the format data is written into the blackboard is restricted to contain only first order logic statements. There are a number of disadvantages to this type of system. First, it is not scalable. When more rules are added, the time to go over all rules to check whether they apply increases. A second disadvantage is that it is not possible to apply more than one rule at the same time, because it is hard to estimate how many facts will be changed by the chain of rules that is invoked by triggering a rule.

Distributed systems (e.g. CTK (Dey, 2000)) are based on a different approach. The main assumption of distributed networks is that data processing can be separated and divided over different modules. Inside these modules the means of processing do not matter. Question however is whether, when going into higher representations, modules reasoning about situations or abstract concepts will not require so much data that they will become blackboards themselves because of the amount of information they have to interpret. But, at low and

	Intelligence	Context
Centralized	**Cosy**, ACT-R	**IROS**, CareLab
Distributed	**MARIE**, ORCA, YARP, OOA[1]	**CTK**

Table 3.2: The placement of the systems that were discussed above in the dimensions that were determined in the introduction of this chapter. Systems that will be elaborated upon are depicted in boldface.

medium level perception there is certainly a right of existence for recognition networks, and at higher levels they are still able to provide the infrastructure. From a neuropsychology point of view, a distributed network is a closer analogue to the brain.

3.3 System descriptions

In Table 3.3 a classification of systems in the dimension we have described above can be found. Below we are listing a system in each separate category, to give an overview of the

3.3.1 CoSy

CoSy (Hawes et al., 2007) is an architecture designed to be a general purpose cognitive systems architecture. It consists of a collection of structurally identical sub architectures that are 'loosely connected' to each other. Each sub architecture has these basic components (See Figure 3.1):

- A working memory to which all components have read/write access and all other sub architectures can read from.
- A task manager that keeps track of which tasks are handled and which components are allowed to run.
- Managed components which interact with each other. They can be assigned tasks by the task manager.
- Unmanaged components which behave like basic sensors. They do not interact, they only write into the working memory.

The task manager and managed components interact with each other by writing goals and instructions into the working memory and adapting their own goals toward the current goal. The task manager determines which components get priority and also handles coordination and goal-mediation. There is a privileged kind of sub architecture that is able to write into the memory of other components. This allows for top-down coordination of other sub architectures.

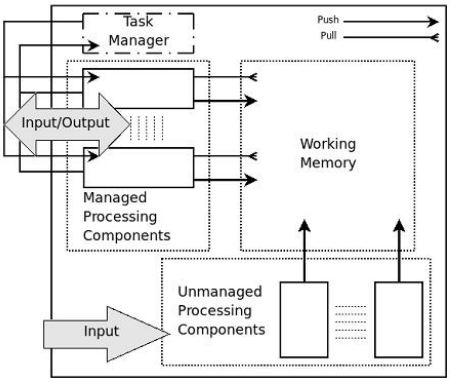

Figure 3.1: A basic component in CoSy

The advantages of CoSy are that the system is entirely modular, so can be run in parallel. Its ontology is well defined and its structure is basically designed to be hierarchical, so top down goal design can exist next to low-level (reactive) processing of events. Its drawbacks are that it is dependent on the programming language of the framework, currently C++ or Java and that its communication protocol is very specific to higher level problem solving. For example, it contains learning instructions, goal-setters, etc.

3.3.2 MARIE

Marie (Cote et al., 2004) is a framework that aims for the same goals as are set in this project: to facilitate the reuse of robot modules. A strong point of MARIE is the proposed encapsulation and communication mechanism. Figure 3.2 shows the basic structure of MARIE. Its main component is a centralized communication component called mediator. The mediator is a collection of communication-adapters and application-adapters. Since applications (the basic data processors) are supposed to be supplied independently, these adapters encapsulate them. Communication adapters are responsible for the communication to an application, application adapters are responsible for the control of an application. The protocol of communication between adapters is standardized through an internal proxy, but communication between the application and adapters is left open. So this communication could take place by shared memory or sockets. Internally however, data gets formatted to standard MARIE-datatypes. MARIE comes with a variety of adapters to include communication with CARMEN / Player (Gerkey, Vaughan, & Howard, 2003), RobotFlow, etc. The main drawback of MARIE is the unnecessary centralization of the communication. All adapters (not applications) have to be run on a single computer, and communication is done through the communication layer of MARIE which is slower than direct coupling of the components.

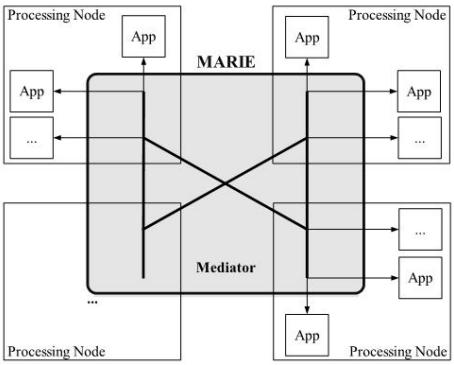

Figure 3.2: The architecture of MARIE, centered around the mediator component

3.3.3 IROS

Interactive Room Operating System (IROS) (Johanson, Fox, & Winograd, 2002b) is a blackboard type situation awareness system. It works with a central heap, which processes every incoming message. The internal structure of this central component is logical. It transforms incoming messages into facts which are ran through the rulebase.

This approach has a number of strong points. The centralized reasoning is easily monitored and controlled, and can be extended with new rules easily. The reasoning algorithm can also be varied when using a single database. Another strong point is the separation between abstraction and reasoning which allows for independent development of sensors although the use of more complex sensors in a hierarchy is not supported by this approach since all reasoning is done inside the database.

One of the major weaknesses of this approach is the scalability. The use of a single non-decomposable database prohibits multiple algorithms to be run parallel because of inconsistencies. Furthermore it is very difficult to maintain one huge database of rules because it becomes impossible to oversee all effects that a change to the set of rules has.

3.3.4 CTK

The context toolkit (CTK) (Dey & Abowd, 2000) is a context awareness system developed to acquire context information in pervasive computing applications like smart rooms. CTK has a control system that is partly distributed, and partly centralized. The centralization is due to a currently undocumented module called the Discoverer, which is a component that registers and keeps track of all running modules in the framework. Components can subscribe to the discoverer which can notify other components of their existence, and query it for the location of other components that are of interest.

The older architecture of CTK (Figure 3.3) however allowed modules to run on their own, since it did not provide a discoverer component, and it allowed

components to register to one another. In this setup there is bookkeeping needed by the components themselves, because they have to know where their "sources" are. With the discoverer they can query it for this information, but still need to connect to their sources themselves.

The basic module in CTK is the widget. It is a standalone unit that either does the processing necessary for a certain datatype, or wraps around an algorithm that does. There are a number of actions that can be performed by a widget, which either focus on another widget, or on the discoverer.

CTK modules send XML messages by HTTP over TCP. This means each module is listening on a unique port for incoming messages. This does not mean it cannot use any other ports for e.g. data communication.

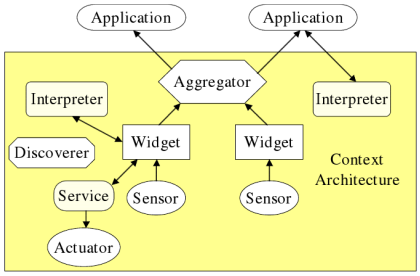

Figure 3.3: The CTK architecture

3.4 Robotic frameworks

A different class of systems is the robotic support framework. This category includes ORCA (Brooks et al., 2007), Player 2.0 (Collett, MacDonald, & Gerkey, 2005), MARIE and RobotFlow (Cote et al., 2004), and YARP (Metta, Fitzpatrick, & Natale, 2006). These systems can be placed on the dimension that was introduced above, but have the advantage that they were specifically built for robotic systems. These systems will also be discussed in the next sections. Although they have options that are at some points more tailored to robotic support, we will evaluate them in the same manner as context awareness systems.

3.5 Communication

In most intelligence systems communication is done through shared memory (Hawes et al., 2007; Anderson, 1993), while context awareness systems are mainly utilizing network protocols for communication (Johanson et al., 2002a; Dey & Abowd, 2000). This distinction can of course be viewed in light of the task that is performed by the network. Intelligence systems need to be more integrated than context awareness systems.

Another part of communication, which is only relevant to systems that use network communication, is the protocol messages are transmitted in. There is a tendency to use protocol middleware like CORBA (Orfali & Harkey, 1998; Nelson, 1998), ACE (Schmidt & Huston, 2002; Cote et al., 2004) or JINI (Arnold, Scheifler, Waldo, O'Sullivan, & Wollrath, 1999; Collett et al., 2005). As we will make clear in our requirements (Chapter 4) we choose for another direction for communication.

3.6 Summary

All discussed systems have their advantages. A big advantage of reusing a framework is that it allows for rapid prototyping and thus offers a lot of functionality from the start on. To determine which systems are interesting for use, adaptation, or feature-reuse, a selection will first be made regarding the independence of protocol, the distributedness, and the complexity of the system. Table 3.3 lists the systems that were considered, and their features. We have not included resource discovery and configuration. These requirements will be explained in Chapter 4. It suffices to say that only CTK fulfills resource discovery, and to our knowledge none of the systems features configuration.

	Structure	Communication
CoSy	distributed	Memory
ACT-R	centralized	Memory
MARIE	distributed	Memory
YARP	distributed	TCP
IROS	centralized	TCP
Player	distributed	TCP (dedicated library)
ORCA	distributed	Multiple (dedicated libraries)
CTK	distributed	TCP - HTTP/XML

Table 3.3: System specification of several context awareness systems

When considering Tables 3.3 and 3.3 we conclude CTK, Player, and ORCA score highest on our requirements, with Player and ORCA having the downside of having to use a dedicated library. With our requirements we will further rate the systems described in this chapter and decide whether it is necessary to implement a new system or if we can reuse an old system.

In the next chapter, Chapter 4 we will list the requirements that we will use for the development of a new system and conclude whether we can use or adapt an existing system, or whether we have to design a new system.

Chapter 4

Requirements

Having considered the state of the art on context awareness systems, we will now construct a list of requirements that have to be fulfilled by a robotic context awareness system. The requirements are constructed from the evaluation of systems in the previous chapter and an evaluation of our research questions (Section 1.1). Our requirements are aimed primarily at our preset goal of reusability. The other requirements are either related to reusability or list what in our view is necessary for a context awareness system. From these requirements we will decide on an implementation for our framework.

We distinguish two categories of requirements: user centered and system centered. The first category (Section 4.1) contains requirements that are constructed to the benefit of the user, the latter (Section 4.2) contains requirements that are necessary for the performance of the system. In this view we loosen the definition of user, to include both the application developer and the developer of individual modules, since both have to be able to work with the system.

4.1 System centered requirements

4.1.1 Modularity

We define a module as an independent specialized data processor (algorithm) that only specifies incoming and outgoing data(types). During operation it is a stand alone program that receives its input, processes it, and sends out its output. This output can be a higher levelled representation or a subset of the input, an aggregate of several inputs, or a conclusion that was made triggered by the input.

4.1.2 Distributability

This also opens the possibility of running components on different computers to be able to compute in parallel to ensure responsiveness during interaction.

Physically distributed algorithms also have the advantage of speed and reliability, because run separately more resources are available and it is less likely multiple modules will fail when a computer fails.

4.1.3 Implications of modularity and distributability

Conflict management

When multiple modules are used to output the same information, like using a face identifier and an RFID reader to identify a person, conflicting data can be entered into the framework. There has to be some way to resolve these conflicts. In this resolving a decision has to be made based on the quality of service a module provides: e.g. whether it provides more information than another, or more reliable information. In some cases no distinction can be made and the modules will have to be viewed as redundant. It is then up to the application developer to make a choice, or up to the randomness of the configuration module.

Control & Reverse connections

Network modules have to be controlled in some way. They have to be turned on, paused, connected to each other, and at some time turned off. The communication protocol should be consistent for these control messages. Control can also be used to establish feedback connections in a network. Feedback connections can be used to establish some control of modules that are later in the network over modules that are before them. A classic example of this type of control is top-down inhibition as can be seen in human attention (Klein, 2000) and architectures like IA (McClelland & Rumelhart, 1981).

The third control application is of a more practical nature. Attention and control can have an interesting interaction to determine which modules are not necessary at the moment. When it is dark there is no use in running all face recognizer algorithms.

4.1.4 Communication

The modularity requirement of Section 4.1.1 defines the scope of a software component. The communication requirement is involved with the clear definition of communication patterns so that modules can operate independently. It is necessary to specify these formats in advance, so it is clear what modules can be connected without problems.

The main reason for a context awareness middleware is data communication. The application developer is mainly interested in data that is perceived by the sensors, and in some cases data that is processed into an abstract or filtered format. This means the communication protocol needs to be able to deal with various kinds of data type. From raw sensory data, to high-level constructs like relations or structures. A vision preprocessor could return (part of) an image as output, while a more abstract module like a gesture recognizer will return another kind of representation. A means of representation would for example be

"talking(jane)" or "facing(john, mary)". It is clear that both types are different, but have to be supported the same by the protocol. We will provide a list of data type constructs that can be supported in Chapter 5.

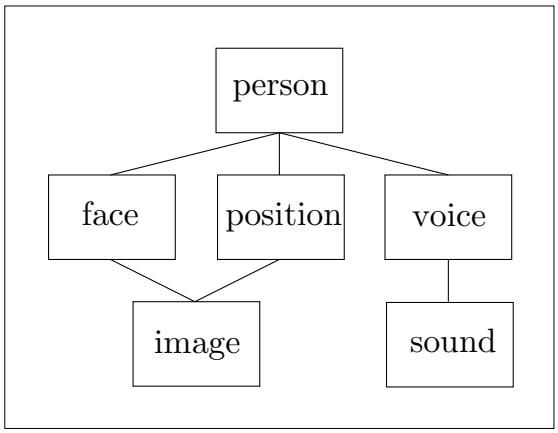

Figure 4.1: An example of data abstraction. Different datatypes need to be represented, but have to be communicated within the same protocol.

There are two (non-exclusive) types of communication models that can be implemented: event based (e.g. (Van Cutsem, Mostinckx, Boix, Dedecker, & De Meuter, 2007) and remote procedure call based (Birrell & Nelson, 1984). These models can be expressed in terms of each other so they are equally expressive. Event based communication is based on sending data when it comes available, so every module would send every piece of information another module is interested in as soon as it gets it. Remote procedure call based communication is based on requests or queries. A module queries another module for information, which can be historical, current, or future. In the last case the providing module sends the information as soon as it comes available, in the other cases it sends it immediately.

As described in Section 3.2 communication can also be done by blackboard systems. This is a combination of event driven, and remote procedure call communication, since all information is written to a blackboard (event driven) and interested modules can query the blackboard for information regarding their input needs. As explained in Section 3.2 this blackboard can become a bottleneck for the system so we will not use this paradigm.

We will make use of both models: event driven for standard communication purposes as well as administrative, and remote procedure call for the querying of data when modules do not want a constant stream of data, but respond to a user, or the environment.

4.1.5 Time and performance

There are two aspects of time that need to be considered, the representation of time and timing of processing. Time representation is needed for processes like memory retrieval and pattern recognition. The issue of memory retrieval is

practical. In social interaction the time some information is processed carries information about the interaction itself.

The other time aspect has to do with performance. In order to develop real-time applications it is very important to be able to depend on the responsiveness of modules. A later module would not only want to know the time its predecessor was done computing, but also the time of perception of the specific percept to know the time delay introduced by the chain of computation. For this comparison it is also necessary to have the same time representation for all modules. So there is a need for synchronization through a system like a network time protocol server (Mills, 1991).

4.1.6 Storage

Memory, in this case representational memory, is important for stable recognition. To be able to track a person across a room or over time historical percepts are necessary. This makes it clear that memory needs to be represented inside the network.

There are two possibilities for memory storage: distributed or locally. We aim not to use a central storage unit because it would form a bottleneck in the network. This leaves distributed storage. So each individual module needs to have some storage mechanism, either internal or external. For externally developed modules this means storage is already included and has to be wrapped or it needs to be added either inside the wrapper or as an independent module.

Storage of data can in theory be without time restriction, but due to space restrictions it is not possible to save all data that is acquired. This problem is correlated with the size of the output that has to be saved. Modules that are closer to the sensors usually have larger data structures which require more memory. A possible solution to this problem would be an exact specification of the memory requirements (in terms of history length) of modules on their providers. This way a module that has no requirements on its memory could then just turn it off and a memory with a requirement of only 7 time steps could release older information.

A different solution to the memory problem that perhaps simplifies the basic structure and protocol of the network would be having modules maintain the input they need themselves. So a motion detector would save a previous output of the camera itself. The advantages of such a storage model are that there does not have to be any query protocol included in the language and data only needs to be sent once. Also externally developed modules often do not rely on a query mechanism since they are not developed to be included into a network. So these modules already have a built in memory for previous input. The disadvantage is that there can be redundant storage.

For the SA component to be able to answer queries of the user there has to be a reliable way to access the latest data of modules. This could be done by storing the last data, or by querying.

4.2 User centered requirements

4.2.1 Reusability

Our context awareness middleware should enable the integration of externally developed modules, because a single developer cannot deal with the complexity of various input modalities. This means that a developer should be able to develop his own module in his programming language of choice without having to worry about much more than basic communication protocol (see also Section 4.1.4). It also means that when a third party algorithm is obtained one has to be able to incorporate it easily. In Section 4.1 structural requirements will be considered that enhance reusability.

Regarding this reusability requirement, it is important to have a compact protocol that enables the application developer to do as precise communication as possible while keeping the total size of the protocol as small as possible. Of course there need to be a limited number of overhead messages, but we strive to keep the ratio overhead/data communication very low. In Section 4.1.4 several communication requirements which stimulate ease of use will be discussed.

4.2.2 Configurability and maintainability

Configuration can be divided into two main tasks: construction of a network of modules and allocation of resources for individual modules.

Because perception, the main task of a context awareness middleware, can be thought of as a hierarchy of processing steps, construction of a network can be done relatively simple by backward chaining the dependency of the modules that provide output that is of interest. For example, a person recognition module can specify its input as needing a face recognition and voice identification module. Automatic configuration relieves the application programmer from the task of finding out what modules are necessary. However, configuration becomes more interesting when perception is viewed as a bidirectional flow of information (e.g. (Bullier, 2001)) because information flow could in theory become circular. We require our framework to be able to construct this type of networks as well.

Allocation on the other hand is more hard to achieve. Allocation deals with dividing resources (machines) over a network of potentially computationally demanding modules. Metta et al. (2006) deliberately exclude automatic allocation from their system: YARP because “the link between hardware and corresponding control software is subject to constraints understood by the developer, but cumbersome to encode, particularly in a continually changing research environment”(p. 2). We will not require our framework to be able to allocate resources over processors. This is beyond the scope of this thesis. We will however strive to make the system distributable, but automatic allocation will not be done.

4.2.3 Wrappers

The use of externally developed modules has one main drawback: in praxis it is often hard adjust the module to comply with the protocol because the source

code is unavailable. It is inevitable an externally developed module will not use the communication protocol required by the network.

A likely solution to this problem is to encapsulate modules within wrappers that translate input from the required protocol to the correct format for the externally developed module and do the reverse for its output. For the framework this would mean each module knows about only the content and not the format of its predecessors output.

4.2.4 Interface to the application developer

The application developer will have to be able to interact with the network in some way. We propose to use a dedicated module for this, which we will call the situation awareness (SA) module. This relieves the application developer of the task of communicating with individual data providers, and thus makes the system easier to use. To know which information is available the application programmer will be provided with an API listing the different representations that can be provided by individual modules.

4.3 Evaluation of existing systems by requirements

With these requirements we will try to implement a robotic context awareness system. As said in Chapter 3 we prefer to reuse and adapt an existing system because of the scope of this thesis. The most important requirements on which we finally will judge each system are distributability, configurability, protocol, interface, and storage. Table 4.3 features the systems that scored highest on these requirements. In addition the remaining requirements stated above will be considered, but these can be added if an existing system is selected for reuse.

	Distributability	Configurability	Protocol	Interface	Storage
CTK	+	-	+	+	+-
CoSy	-	-	+	-	+-
IROS	-	-	+	+	+
MARIE	+-	-	-	-	+

Table 4.1: This table summarizes the score of each system on our requirements. We have chosen to exclude the systems that scored low in Table 3.3. ORCA was also excluded from this table because it was not found during our literature research and thus not considered.

Although lacking configurability and having a somewhat difficult interface, CTK has the advantage of being distributed open source, we conclude CTK has the most suitable specification and documentation. In Chapter 5 we will describe the adaptations we have made to it in order to fulfill the requirements that were set in this chapter. The requirements will be used in Chapter 6 to evaluate the implementation.

Chapter 5

Implementation

As described in Chapter 1 we aim at developing a robotic context awareness framework. In Section 4.3 we have concluded CTK comes closest to a system which could be adapted to our needs. CTK is a distributed context awareness system that was intended for use with mobile devices. Because speed is less of a requirement for this domain, less emphasis is given to the size of individual messages and the extend of the protocol. Our general aims for the adaptation of CTK ware speed and ease of use, to which the additions described in Chapter 4: feedback, an easier interface, a clearer querying protocol, and configurability will be added.

In this chapter we will describe the efforts that were made to adapt the CTK framework. We will describe in depth the protocol and structure of the previous version of CTK and the changes that were implemented.

5.1 Situation Awareness module

The current version of CTK (Figure 3.3) has a module, the discoverer, that serves as a supporting mechanism for other modules. It keeps track of the modules in the network, and the datatypes they can provide. However, most of the administrative communication is still done by individual modules. Registration to the discoverer, querying for datasources, subscribing to them, and maintaining the connections have to be done by the individual modules. Though the discoverer is supposed to be notified of all changes that occur in the network, which requires a lot of unneeded overhead for the other modules, the benefits of this system are rarely utilized. We have centralized the overhead messages while keeping a strictly separated data flow.

The discoverer has been evolved into a situation awareness module, for it will act as a mediator between the modules and the behaviour. This means it will take care that both relevant sensor information and requested information reaches the behaviour, without forming a bottleneck (since no data is sent through the situation awareness module). The other task set out for this module is that it needs to configure the hierarchy of sensors, so that this administration and

information is centralized. At startup it receives a list of data the behaviour is interested in, and constructs a module hierarchy to be able to provide this data to the behaviour.

5.1.1 Time

The SA component is the authority considering time in the network. It sends time information to individual modules regularly to ensure all percepts are provided with the correct timestamp. This is done by adding information to the `<ping>` message. The ping message is a message the SA component sends to check whether modules are still alive (Listing 5.1).

```
<ping>
    <time></time>
</ping>
```

Listing 5.1: ping and time provision combined

5.1.2 Communicating with the SA component

There are two ways for the application developer to obtain data from a component that is managed by the SA component: it can subscribe to its data and/or it can query for it. We will give an exact specification of the messages that are associated with these requests. All components have to initially subscribe to the SA component to notify it of their incoming requests and their outgoing attributes. The SA component will then search for components providing these attributes and subscribing the component to them. The registration message is depicted in Listing 5.2 It consists of three structures. A `<client>` structure, as depicted in Listing 5.8 represents the details of the registrar. The `<inQueries>` structure is a list of queries (Listing 5.13) that state the attributes the registering component is interested in and the conditions that are set on these attributes.

```
<register>
    <client></client>
    <inQueries></inQueries>
    <outAttributes></outAttributes>
</register>
```

Listing 5.2: The message that is sent to the SA component when a component registers itself.

The SA module then constructs a dependency tree, based on which it will send subscription messages to components that have been selected to provide data for requesting modules. Conflict management however is still not implemented in this version. The SA module simply picks the (alphabetically) first occurrence of a data provider and connects it to the demanding module.

The add subscriber message (Listing 5.3) will consist of the client the data has to be sent to and a query specifying the data (attributes and conditions). The

query stated in the message will be evaluated only for newly acquired data, so no conditions can be stated on past data.

```
<addSubscriber>
    <client></client>
    <query></query>
</addSubscriber>
```

Listing 5.3: Add subscriber message

During operation the SA module, same as the Discoverer, will ensure all active modules are alive by a ping message (Listing 5.1). When a module is not responding the SA module tries to find another source of data for the modules that were subscribed to and replace the old module with it.

Querying the SA module

Because our aim is to serve as an interface between the network and the behaviour the SA module will serve as a facilitator for sensing queries to individual modules. So the behaviour will be able to query the SA for information as were it as if the SA module produced this information itself. It will translate the query and send it to the relevant module, with the instruction to send it to the behaviour. This is done by specifying the behaviour as client to the query instead of the SA module. This way not all information has to go through the SA.

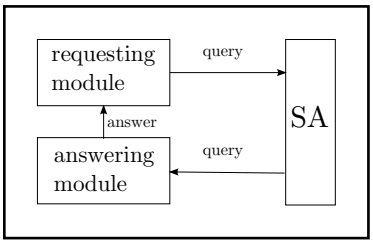

Figure 5.1: The process of answering a query. The SA module only sends through the query message, relieving the requesting module of searching for a provider. The subscription process is analog to this.

The retrieve data message (Listing 5.4) is the basic query mechanism. It replaces the old retrieve data, and the old poll widget message. The idea is that the query specified in this message can, in contrast to the query in the add subscriber message, place a constraint on the time, so percepts of multiple time steps can be retrieved. To facilitate the functionality of polling for the last data in a widget we have introduced a placeholder for the latest timestamp that was processed. This would be done by adding `<equal><timestamp>LATEST</timestamp></equal>` to the query.

```
1 <retrieveData>
2    <client></client>
3    <query></query>
4 </retrieveData>
```

Listing 5.4: Retrieve data message

API

A module can query the SA module for a list of all current datastructures held by the network with a getApi message depicted in Listing 5.5. The API that is returned (Listing 5.6) is an attribute list as will be explained in Section 5.2.2. This attribute list differs in that it indicates the types of the attributes. These types can be a structure, plain text, but also a proprietary type like a binary structure provided by a specific programming language.

```
1 <getApi></getApi>
```

Listing 5.5: The message that is sent to obtain the API of the active modules

```
1 <api>
2   <attributes>
3     <sensortime></sensortime>
4     <face></face>
5   </attributes>
6 </api>
```

Listing 5.6: An example api return message

5.2 Communication

5.2.1 Protocol

CTK modules send XML messages by HTTP over TCP. This means each module is listening on a unique port for incoming messages. This does not mean it cannot use any other ports for e.g. data communication. We will first explain the CTK protocol by an example message, pictured in Listing 5.7.

Basic protocol

HTTP Lines 1 through 6 represent the HTTP protocol. Line 1 lists the type of message, in this case a POST message, which usually represents a data input to a module. A GET message is used to ask for information from a module. Line 2 lists the version of HTTP. Line 3 describes the client the message originates from, within CTK this is always "'Context Client,"' line 4 describes the computer

```
1  POST ping
2  HTTP/1.0
3  User-Agent: Context Client
4  Host: nlvehvres2dt6a8
5  Content-Type: text/xml
6  Content-Length: 206
7  <?xml version=1.0?>
8  <addSubscriber>
9      <id></id>
10     <subscriber>
11         <id></id>
12         <hostname></hostname>
13         <port></port>
14         <callbackName></callbackName>
15         <attributes></attributes>
16         <conditions></conditions>
17     </subscriber>
18 </addSubscriber>
```

Listing 5.7: An example CTK message

the message comes from, and lines four and five describe respectively that the message is in the XML format, and the length of the message starting after line 5.

XML Line 5 indicates the message is written in the XML 1.0 format. This means every piece of information is built as a (sometimes nested) xml tuple. The name of the message or data is therefore always in the tagname, and the information is between tags.

Message Lines seven through seventeen are the actual message that is transmitted. Line seven indicates this particular message is an "'addSubscriber"' message, which means the sending module wants to subscribe to data provided by the receiving module. There are a number of these messages, which will be listed in remainder of this chapter.

Changes to the protocol

We have stripped the protocol of most of its recurring data, without losing the protocol requirements. This has been done to minimize the amount of overhead data that is sent with each transmission, increasing potential speed. Lines 3 and 4 have been removed. Line 3 is not necessary with each transfer, since only messages from context clients are sent, and other messages will probably not be interpreted correctly. Line 4 is the hostname, it is already represented in the data structure.

The message itself has also been altered. The xml definition of line 7 has been removed, since all modules communicate in xml version 1.0. We have discarded

the `<id>` tag (line 9) that precedes every message, because there are basically no fallback options when a message is sent to the wrong module. By centralizing control the checking for valid subscriptions will be done in the situation awareness module. The `<subscriber>` or in other messages `<client>` tag will be standardized into the structure in Listing 5.8. It is sent at subscription, query (retrieveData), and registration/unregistration to the discoverer.

```
1 <client>
2     <name></name>
3     <hostAddress></hostAddress>
4     <port></port>
5 </client>
```

Listing 5.8: New client structure

5.2.2 Datatypes

In this section we will address the features of the original CTK and subsequently describe the adaptations that were made to these features. The original CTK has four distinct high-level data types: attributes, conditions, callbacks, and errorcodes. The common basic datatype is the attributeNameValue tuple.

Attributes

The basic data structure in the original CTK is an attributeNameValue tuple. Attributes are grouped by the `<attributes>` tag which serves as a marker for the start and end of attributes in the message. So a single attribute would also be indicated with the `<attributes>` tag. An example of these structures are given in Section 5.9.

```
1 <attributeNameValue>
2     <attributeName>speech</attributeName>
3     <attributeValue>good eye cat</attributeValue>
4 </attributeNameValue>
```

Listing 5.9: Basic attribute in CTK

An attribute can also be be a structure. In a structure multiple attributes can be grouped under a name. A structure is indicated by the `attributeType="struct"` tag. It consists of a name, which is not surrounded by xml tags, followed by a list of attributes structured as described in the previous paragraph.

We have replaced all attributeNameValue structures by tuples of `<name>value</name>`. Structures will be supported implicitly by the structure of xml, so a structure would look like Listing 5.10. These are then combined in a larger structure, which is kept from the original CTK framework, the `<attributes` tag (Listing 5.11).

```
<face>
    <x></x>
    <y></y>
    <size>
        <height></height>
        <width></width>
    </size>
</face>
```

Listing 5.10: The new structure note that structures can be nested

```
<attributes>
    <sensortime></sensortime>
    <face></face>
</attributes>
```

Listing 5.11: The new percept message

Queries

To receive information modules can be queried by all other modules. In these queries (Listing 5.13), conditions can be specified so subsets of a widgets history can be returned. Multiple conditions are marked and grouped by a `<conditions>` environment. CTK components then take the conjunct (AND) of these conditions. There is no available syntax for disjunction (OR) or negation (NOT). See also Listing 5.12. When a condition is placed on a member of a structure, the member has to be noted as `structurename_membername`. CTK then returns the entire structure.

```
<condition>
    <name>myStruct_speech</name>
    <compare>4</compare>
    <value>0</value>
</condition>
```

Listing 5.12: Condition. In the compare tag 0: equal; 1:less than equal; 2:less than; 3:greater than equal; 4:greater than.

A callback is, as stated above, a name for a collection of data. A callback is answered when new data that falls within the definition of the callback is perceived. In theory different callbacks could be stated for different combinations of perceived attributes, but in practice only a single callback is defined most for most of the widgets: the update callback. The update callback used to indicate new data is perceived. For specialized sensors there is usually a single structure to be perceived so there is only a single callback specified. Callbacks are mainly a way of grouping data into a structure and giving this structure a callback name. Because of this, we have removed this data structure.

We have grouped the two concepts conditions and callbacks into one structure

called a query. A query consists of two elements, a list of attributes that are to be returned when the condition is satisfied and the condition. The list of attributes is made up of individual attribute tuples.

A basic condition has the structure that is depicted in Listing 5.13. Its outer tags represent the operator, which (at the moment) has to be either “greaterThan”, “equal”, or “smallerThan”. The two enclosed structures are respectively the first and second argument to the operator. The first argument necessarily has to consist of a variable, the second of a value. There is also a syntax available for not, which encloses the value of an attribute. Conditions can be composite by surrounding two conditions by `<AND>` or `OR` tags, or singular.

```
1 <query>
2   <attributes><face></face></attributes>
3   <greaterThan><timestamp>3</timestamp></greaterThan>
4 </query>
```

Listing 5.13: query

Error codes

Error codes are sent to indicate the success of a communiqué. In CTK they are standardized, but there are a lot of them. Every error has its own errorcode. A full list will not be given here because we have simplified the list. It suffices to know that the standard errorcode is ”noError”, which is always used to describe success of the transaction, and that other errorcodes are largely understandable by their form.

5.3 Rapid prototyping

The rapid prototyping software development kit that was included in the CTK package has been updated with a standardized HTTP server, and every change that has been made to the protocol has been implemented in the SDK as well. Next to this we have removed all components that were excluded from the new design. With the adapted SDK it is thus still possible to easily write modules in the new system.

5.4 Conclusion

Although we have preserved the basic structure and communication protocol of CTK, we have simplified both by removing several components and lines of protocol. We have also introduced a clearer interface to both module developers and application developers by separating data flow and communication flow. This results in having to querying the situation awareness module for data. Next to this we have introduced configuration so that modules by sacrificing some freedom do not have to connect themselves to their data sources. Finally

a unified structure was presented for both querying and indicating data types of interest. We will evaluate these modifications by our requirements, and by a test case experiment in the next chapter.

Chapter 6

Evaluation

In this chapter we will evaluate the adapted version of the CTK framework. We will start by evaluating it by the requirements that were set in Chapter 4. Subsequently we will describe a test case experiment we performed with the iCat research platform (Breemen et al., 2005).

6.1 Evaluation by the requirements

The requirements that were set in Chapter 4 were in general fulfilled. The sections below will explain in which ways requirements were met, and how they could be improved in case there are some weak points to our solution.

6.1.1 User centered requirements

Reusability and ease of use

The reusability of individual modules, and also the adaptation of third party algorithms in the framework, can be estimated by the amount of coding that has to be performed to use a module multiple times in different network set-ups. The reusability of the framework was already high due to its *modularity* and *distributability* our adaptations have not changed this.

The ease of use of the framework is defined by the amount of coding that has to be performed to set up the network and to introduce a new module into the system (cf.: *wrapping*). To facilitate this we made a few adjustments to minimize the protocol, described in Chapter 5. The removal of redundant lines and the standardization of default structures like the `<client>` structure improved both the *incorporation* and the ease of use of the framework.

Configurability

Configurability was absent in CTK. With the SA module we have introduced automatic configuration and reduced the total amount of messages and overhead

by individual modules contributing to *ease of use* and the overall *performance*. A downside to this approach however is that the SA component could become a bottleneck when communication increases. This can be solved by a distributed form of SA but this would immediately increase overhead for individual modules. It has not been tested yet what size the network has to have for this problem to appear, but we expect that individual modules will find a good balance between subscriptions and queries, which will result in minimized traffic. For example, a module that continuously sends queries over the network should better switch to a subscription to this data, as subscription is only once run through the SA module.

Conflict management

As indicated in Chapter 5 conflict management is still absent in this version of the system. We expect better conflict management would be done when a meta variable is introduced to describe the quality of the data on different levels of interest. For example, a visual module could list its framerate and resolution, a recognizer could list its accuracy, and types of error. It can be seen that this approach needs some extra adaptations for the protocol since these levels of interest needs to be standardized. A simpler form of this could be an objective quality of data that would be introduced by the designer of the system, or even a manual configuration by changing the datatype-names.

Control

We have sacrificed the control that can be asserted over modules largely to the ease of use and configurability of the system. In our view it is more important to have a simple and clear communication protocol than a large means of controlling individual modules. We will elaborate on this stance in Chapter 7.

Speed and performance

As indicated above we have implemented a minimized approach to the protocol that strips away a large part of the recurrent and redundant information. This increases the performance and overall speed of message passing by reducing the header/message ratio.

Interface to the Application developer

The interface to the application developer has been simplified and extended. The automatic *configuration* and the API allows for more effective communication, and since all communication is done through the SA module, an application developer only has to keep open one line of communication for his requests, as well as having to open at least one for the reception of data. In the original setup this would be achieved by querying the Discoverer for potential data sources, selecting one, and subscribing to that source.

6.1.2 System centered requirements

Communication model

The communication model was simplified. The number, size, and complexity of messages was reduced. This was done by changing the basic data structures to tuples instead of the composite attributeNameValue structures they were in the previous version. The standardized and extended query message results in one standard mechanism for both querying and subscribing which increases the *ease of use* as well as making retrieval from *storage* more efficient.

Time

In order not to depend on an NTP server (Mills, 2003), we have introduced timekeeping to the ping message. Having a time baseline system on a central module included into a message that is sent for the *maintenance* of the network is preferred over a separate module.

Architecture

Traditionally, context awareness systems are categorized as blackboard or distributed. As remarked in Chapter 3 each separate system has its advantages and drawbacks. Put concisely, the main advantage of blackboard is its limited overhead, peripheral modules only need to communicate with the central blackboard. In this centrality however lies also its disadvantage: a large amount of data in one place can cause network traffic jams. The distributed system resolves this problem by storing data distributed, so that individual sub-datasources are relatively less busy than one main data source. The drawback of distributed systems is the large overhead, modules need to find out where their data of interest is stored, and request it from that source.

Our approach tries to take advantageous parts from both other types of system, by combining the blackboard communication style with the distributed storage. This means a lesser amount of communication needs to be done, without the threat of individual modules becoming a bottleneck, provided data is distributed properly.

Basic protocol

The protocol of CTK, XML over HTTP, was not changed. There is however a disadvantage to using XML over HTTP; it has a high header to information ratio. For the original purpose of CTK, a discrete environment, this was not a problem. However, for the continuous robotic perception environment one can think of protocol with a lower header to information ratio. The HTTP itself was already stripped down, but an alternative to the XML could be comma separated values. The overhead (subscription, querying) could still be sent in XML, since it provides a good means of describing a `<query>`, but data could be sent in CSV, after specifying the order of elements. Or a more radical solution

could be specifying the number of bytes each element consumes and sending the byte stream without any separators. An alternative to HTTP could also be considered, this would mean a direct connection would be established between modules, without using a header to indicate the length of messages. We have not implemented these solutions so they can be seen as suggestions for future research.

6.2 Case study

To evaluate the adapted situation awareness system we have performed an experiment with the iCat research platform. We will evaluate the efforts that were done to link the system to this platform (Open Platform for Personal Robotics (OPPR)(Van Breemen, 2005)), and the operation of the combined system during the experiment. First we will introduce the iCat and OPPR.

6.2.1 iCat

iCat is a robotic research platform designed specifically for human robot interaction (HRI). As shown in Figure 6.1 it consists of a cat shaped head, which moves on an immobile base. iCat has 4 touch sensors, two in its feet, two in its ears, a camera in its nose, two microphones on both sides, and a distance sensor in its left foot. It can move its entire head, its eyebrows, eyes, eyelids, and lips. It is possible to interact through lights on the feet and ears, and there is a speaker on the front of the base.

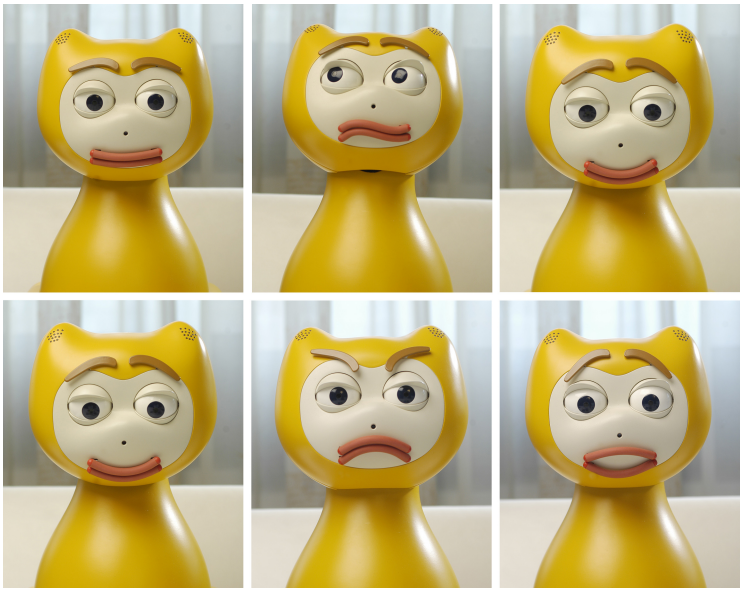

Figure 6.1: The iCat robot showing six facial expressions.

6.2.2 OPPR

The Open Platform for Personal Robotics is a platform that forms the backbone of any iCat experiment. It would be too much to describe its architecture here, so we will give a short description, and refer to Van Breemen (2005) for a full description.

OPPR consists of five main software packages: Architecture, Workbench, Believability, Intelligence, and Connectivity, handling respectively the communication, construction of animations and behaviour, the rendering of animations, the rendering of behaviour, and the connection to outside environments. We will go into the intelligence module, since this module is important in describing Section 6.2.3. The Intelligence module runs RIBML scripts, that are easy to code, even for designers with little programming experience. RIBML scripts run the behaviour of the iCat robot by running animations and saying utterances. An example of RIBML, and a more thorough explanation can be found in Saerbeck, Schut, Bartneck, and Janse (2009).

6.2.3 Linking OPPR to CTK

To test whether our requirements regarding a.o. *ease of use*, *interface*, and *incorporation* were practically met, we have linked the CTK system to OPPR. For this we constructed a library inside the Intelligence module of OPPR that connects to the CTK network as a regular module, and is able to subscribe to a data provider or query it, resp. Listings 6.1 and 6.2.

```
1 <subscribe typeName="card"/>
```

Listing 6.1: The subscribe message in RIBML

```
1 <query typeName="speech"/>
```

Listing 6.2: The query message in RIBML

Linking OPPR to CTK posed some challenges. The linking library was written entirely in Lua, as this is the scripting language run by the interaction module of OPPR. This meant sockets needed to be opened to communicate through the TCP/IP protocol. Because of the potentially parallel remote procedure calls that would be done during which the script had to wait for information to return, this posed a challenge to the architecture.

6.2.4 Experiment

We will briefly describe the iCat experiment here, but since the scope of this thesis is the situation awareness architecture we will concentrate on its operation during the experiment. The experiment we conducted with the iCat in combination with the adapted CTK framework focused on the use of a robotic character as a teacher. We chose the domain of language acquisition as the test

domain, specifically the Toki Pona artificial language (Kisa, 2009). In our experimental setting the subjects (grade school students) were exposed to in one condition a supportive and in the other condition an non-supportive robotic teacher (iCat). For a further description and results see Saerbeck et al. (2009).

We have ran pilot tests with an automatic speech recognizer (Walker et al., 2004), a speaker identifier, and three vision modules: an emotion recognizer, a face position recognizer, and a recognizer to determine where the subject is looking at. However, we did not include these in the final experiment, since recognition rates were not high enough for deployment in an experiment. Eventually, only two modules were used, both were run as Wizard of Oz (Kelley, 1984) input modules. Figure 6.2 depicts the modules linked to the SA module and the behaviour. The first module consists of an interface to input the card shown to iCat, the second module communicates the users utterances to the behaviour. Next to these modules, the aforementioned iCat input modalities were used.

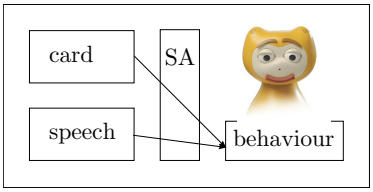

Figure 6.2: The experimental setup. Card and speech represent the (wizard) card- and voice recognizer.

During operation both the architecture and the link between CTK and OPPR functioned properly. Although we did not encounter serious problems, the feature of restarting a module when it fails would however be desired, because when a module halts, the entire system does not function anymore. This functionality could easily be included when the system is not distributed over multiple machines, but when it is, it poses quite a challenge since it introduces allocation to the SA module. Allocation would mean the SA module needs a table of available resources, and is able to control these resources and run programs on individual machines.

The communication model of routing each request through the SA component was very clear and easy to use, and for the application developer the RIBML script was also easy to use. Although, as said before, the effort for linking CTK and OPPR was quite large, especially with a scripting language. We expect this effort to be lighter with a programming language like Java or C++.

From the experiment and the evaluation of the requirements we can conclude there is a need for more control over individual modules and automatic allocation of resources. Also, a better conflict management policy could be adopted in a later version.

6.3 Conclusion

From Section 6.1 we can conclude that configuration is a valuable addition to a situation awareness support framework because it limits the amount of non-automatic messaging that needs to be done. Data-type conflicts are not resolved in the system yet, this can be addressed in future research. The test-case experiment showed that the protocol poses a small challenge when developing modules, but that the only way to overcome this problem is to develop binaries for each programming language to handle communication or to incorporate a third party communication management system.

In the next Chapter the research questions that were presented in Chapter 1 will be discussed and future research topics will be suggested.

Chapter 7

Discussion

In this chapter we will present a general discussion on the situation awareness support framework by evaluating our research questions and provide suggestions for further research.

7.1 Discussion of the research questions

In Chapter 1 we have set the following research questions:

1. Can we construct a situation awareness framework that facilitates the reuse of algorithms without restricting them to a single programming language or system?
2. Can we construct a situation awareness system that provides an easy to use interface to the application developer but preserves performance efficiency?
3. Is it possible to handle all necessary data-types while preserving a clear communication protocol?

The first research question composes a deliberation between programing language independence and reusability. Introducing a set programming language holds several benefits, including high efficiency and the possibility of using features of this language. However, the obvious downside to this is the exclusion of all other programming languages, which obligates users of these languages to use workarounds to be able to work with the system.

The approach described in this thesis is independent of programming languages, which introduces a steeper introduction-curve. A developer needs to script the communication protocol of CTK into his software to make it able to interact with the framework. Although we have reduced both the size and quantity of the protocol, this does not provide the ease of use a ready implemented framework in a single language does. This can be prevented by using the provided rapid

prototyping development kit which allows one to circumvent the communication protocol.

The second research question describes the needs for a language that is able to describe every possible data type. We have found that without committing to a typing system like CORBA (Orfali & Harkey, 1998) or JINI (Arnold et al., 1999), it is very difficult to describe every data type in a single language. As described in Chapter 5 we have adopted the CTK ASCII approach. This approach means every data type needs to be transcribed into a plain text representation, which can in some cases, e.g. for binary data, not be desirable. Since we wanted to make the transition to the developed system as easy as possible we chose not to use systems like CORBA. We conclude that a system that is able to utilize standardized communication back-ends has a large advantage over the ASCII approach, given the back-end is available to all programming languages, which reverts the problem back to the discussion on the first research question.

Our third research question focuses on providing an easy to use interface to the application developer while preserving performance efficiency. In a discussion on this configuration and control have to be included, since they are very important in providing respectively ease of use and performance efficiency. Configuration has the main advantage of relieving modules of their task of finding their data providers. However, by this relieving, the individual modules become dependent on the configuration module, which prevents an application to exert his own control over modules. This can easily be solved by preserving the original communication structure of CTK and overlaying it with the new structure. This, although easily implemented, would mean the SA module would have to be exempted from its configuration task by notifying it connections have already been made.

We have analyzed the need for an additional message that starts and stops modules, or makes them run idle, but decided against it. A module that is running idle will have to be listening for either a startup message or a subscription or query message. In the first case a module would first have to be started up, and could then be queried, while it had to be listening for a message all along, which means the startup message is redundant. The idle message would have to be sent when no other modules are interested in the output of a providing module, but again this is redundant since a module can turn idle until it receives a request.

There is however the possibility of a module being interested in querying a providing module for its data at times, which would have to prevent the providing module from going idle. In our case study 6.2 we have circumvented this problem by not letting a module turn idle, but it would be desirable to have an addition to the protocol indicating to a module it cannot turn idle. This addition would also require a mechanism for modules to indicate their interest in querying a module at some point in the future.

Our test case experiment showed that configuration is very desirable, since it limits the number of messages that have to be sent to other modules, and thus contributes to the usability and efficiency of the entire system.

7.2 Suggestions for future research

7.2.1 Suggestions for module applications

In the previous chapters we viewed modules as basic data interpreters, only specifying an in and output, without suggesting the kind of computation that could be done inside a module. In this section we will center on the computation inside the module, suggesting several applications.

In the design we have considered the possibility of constructing a module which runs a logical engine. With the rapid prototyping environment it would be possible to link the Java code to Prolog through JPL (Singleton, Dushin, & Wielemaker, 2006). The returned types would then be registered as outputs, and truth-values as their values. Although it sounds counter-intuitive it is easy to see an advantage to distributing a logical system when it is possible to separate variables. This prevents the aforementioned bottleneck, as all rules need to be checked for potential triggering when a variable changes.

As a part of the context awareness systems work with either first order logic or fuzzy logic, it would be interesting to make this coupling and evaluate its use. Also, the querying and history feature of the system makes it possible to log historical variables, and reason with them. This could improve reasoning by inducing relations, and making statements about historical data.

7.2.2 Stable representations

When interacting in an ever changing complex world there is a need to determine whether a something in a percept is new, or whether it already occurred in the previous percept. With this knowledge it becomes easier for a robot to determine whether someone in its field of vision is the same person that was there before or the same person that it is hearing, but it can also enable it to develop a historical profile of conversational partners or other objects in the world. Furthermore this system can be used to direct top-down attention to a stable representation, see also Section 7.2.3. The protocol can be extended to facilitate this stable representations. The SA module would have to be extended with a system to distribute or lease identification tags for the stable constructs that would be generated by the modules. These id's could then be leased by the modules to assign to data structures in order to identify them as being the same. The application developer could use the tags to identify structures and perhaps assign a name to them, although implementing the latter would require a larger adaptation to the protocol.

7.2.3 Attention

When stable representations are assumed, directing attention to these representations becomes a lighter task, since there are identifiable object to direct the attention to. Of course the process of deciding which representations demand attention is still as hard, directing attention could be simplified to the focus on a single concept in the perceptive space. This directing attention only

relates to the top-down approach to attention, the bottom-up approach would insist on more conscious perception inside individual modules, but could also be regulated in a top-down fashion by instructing modules to send data when it is unexpected. The implementation of unexpected could be a NOT-condition stating all facts that are expected, so that unexpected new data is sent through. The unexpected-condition would then provide a context for the bottom-up attention.

7.3 Conclusion

It can be concluded that every step that is taken to increase the ease of use will decrease the freedom of individual modules. Configuration conflicts with control, since when configuring modules automatically freedom is taken from individual modules. This also applies to control, and can be overcome by restoring parts of the original protocol. There are some downsides to our choice for a system with a universally usable language instead of a system with an underlying communication manager regarding the easy of use, but these only concern a one time effort and do not affect performance. Finally we conclude that although our approach presented some new technical issues, we believe they are not unsolvable and that it is worth investigating whether a system can be developed that provides both control and configuration, and is both universally usable and easy to incorporate.

References

Anderson, J. (1993). *Rules of the Mind*. Lawrence Erlbaum Associates.

Arnold, K., Scheifler, R., Waldo, J., O'Sullivan, B., & Wollrath, A. (1999). *Jini Specification*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Birrell, A., & Nelson, B. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, *2*(1), 59.

Breemen, A. van, Yan, X., & Meerbeek, B. (2005). iCat: an animated user-interface robot with personality. In *Proceedings of the fourth international joint conference on autonomous agents and multiagent systems* (pp. 143–144).

Brooks, A., Kaupp, T., Makarenko, A., Williams, S., & Orebaeck, A. (2007). Orca: A component model and repository. *Software Engineering for Experimental Robotics*, 231.

Bullier, J. (2001). Feedback connections and conscious vision. *Trends in Cognitive Sciences*, *5*(9), 369–370.

Collett, T., MacDonald, B., & Gerkey, B. (2005). Player 2.0: Toward a practical robot programming framework. In *Proceedings of the australasian conference on robotics and automation (acra 2005)*.

Cote, C., Letourneau, D., Michaud, F., Valin, J., Brosseau, Y., Raievsky, C., et al. (2004). Code reusability tools for programming mobile robots. *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, *2*.

Dautenhahn, K., Ogden, B., & Quick, T. (2002). From embodied to socially embedded agents–implications for interaction-aware robots. *Cognitive Systems Research*, *3*(3), 397–428.

Dey, A. K. (2000). *Providing architectural support for building context-aware applications*. Unpublished doctoral dissertation, Atlanta, GA, USA. (Director-Gregory D. Abowd)

Dey, A. K., & Abowd, G. (2000). Towards a Better Understanding of Context and Context-Awareness. *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*.

Drury, J., Scholtz, J., & Yanco, H. (2003). Awareness in human-robot interactions. In *Proceedings of the ieee conference on systems, man and cybernetics* (pp. 912–918).

Endsley, M., & Garland, D. (2000a). *Situation awareness: analysis and measurement*. CRC Press.

Endsley, M., & Garland, D. (2000b). Theoretical underpinnings of situation awareness: A critical review. *Situation Awareness*.

Fong, T., Nourbakhsh, I., & Dautenhahn, K. (2003). A survey of socially interactive robots. *Robotics and autonomous systems*, *42*(3-4), 143–166.

Gerkey, B., Vaughan, R., & Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. *Proceedings of the 11th International Conference on Advanced Robotics*, 317–323.

Gutwin, C., & Greenberg, S. (2002). A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work (CSCW)*, *11*(3), 411–446.

Haselager, W. (1997). *Cognitive Science and Folk Psychology: The Right Frame of Mind*. Sage.

Hawes, N., Sloman, A., Wyatt, J., Zillich, M., Jacobsson, H., Kruijff, G., et al. (2007). Towards an integrated robot with multiple cognitive functions. *Proc. AAAI*, *7*.

Johanson, B., Fox, A., & Winograd, T. (2002a). The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms.

Johanson, B., Fox, A., & Winograd, T. (2002b). The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE pervasive computing*, 67–74.

Jurafsky, D., & Martin, J. (2000). *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. MIT Press.

Kelley, J. (1984). An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems (TOIS)*, *2*(1), 26–41.

Kisa, S. E. (2009, March). *The toki pona language.* `http://en.tokipona.org/`.

Klein, R. (2000). Inhibition of return. *Trends in Cognitive Sciences*, *4*(4), 138–147.

Martin, D., Cheyer, A., & Moran, D. (1999). The open agent architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, *13*(1), 91–128.

McClelland, J., & Rumelhart, D. (1981). An Interactive Activation Model of Context Effects in Letter Perception: Part 1. An Account of Basic Findings. *Psychological Review*, *88*(5), 375–407.

Metta, G., Fitzpatrick, P., & Natale, L. (2006). YARP: yet another robot platform. *International Journal on Advanced Robotics Systems*, *3*(1), 43–48.

Mills, D. (1991). Internet time synchronization: The network time protocol. *Communications*, *39*, 1482–1493.

Mills, D. (2003). A brief history of NTP time: Memoirs of an Internet timekeeper. *ACM SIGCOMM Computer Communication Review*, *33*(2), 21.

Murphy, R. (2000). *Introduction to AI robotics*. The MIT Press.

Nelson, G. (1998). Context-aware and location systems. *Unpublished PhD dissertation, University of Cambridge*.

Orfali, R., & Harkey, D. (1998). *Client/server programming with Java and CORBA*. Wiley NewYork.

Saerbeck, M., Schut, T., Bartneck, C., & Janse, M. (2009). *Expressive robots in education: Varying the degree of social supportive behavior of a robotic tutor.* (In preparation)

Schilit, B., & Theimer, M. (1994). Disseminating active map information to mobile hosts. *Network, IEEE*, *8*(5), 22–32.

Schmidt, D., & Huston, S. (2002). *C++ Network Programming: Systematic Reuse with ACE and Frameworks, Vol. 2*. Pearson Education.

Singleton, P., Dushin, F., & Wielemaker, J. (2006). *JPL: A bidirectional Prolog/Java interface.*

Van Breemen, A. (2005). iCat: Experimenting with animabotics. In *Proceedings, aisb 2005 creative robotics symposium.*

Van Cutsem, T., Mostinckx, S., Boix, E., Dedecker, J., & De Meuter, W. (2007). Ambienttalk: Object-oriented event-driven programming in mobile ad hoc

networks. In *Proceedings of the xxvi international conference of the chilean society of computer science* (pp. 3–12).

Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., et al. (2004). Sphinx-4: A flexible open source framework for speech recognition. *Sphinx Whitepaper, Sun Microsystems INC*.

Want, R., Hopper, A., & Gibbons, V. (1992). The active badge location system. *ACM Transactions on Information Systems*, *10*(1), 91–102.

Weihe, K. (1997). Reuse of algorithms: Still a challenge to object-oriented programming. *ACM SIGPLAN Notices*, *32*(10), 34–48.

Wren, C., Azarbayejani, A., Darrell, T., & Pentland, A. (1997). Pfinder: Real-Time Tracking of the Human Body.