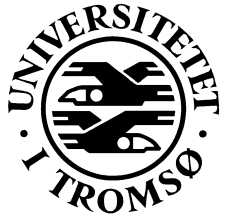

REPORT

Computer Science Technical Report: **94-19** **December 1994**

The StormCast API Specification of Software Interfaces in StormCast 2.1

G.Hartvigsen, D.Johansen, V.Farsi, W.Farstad, B.Høgtun, P.Knudsen

INSTITUTE OF MATHEMATICAL AND PHYSICAL SCIENCES
Department of Computer Science
University of Tromsø, N-9037 Tromsø, Norway, Telephone +47 77 64 40 41, Telefax +47 77 64 45 80

The StormCast API –

The Specification of Software Interfaces in StormCast 2.1

G. Hartvigsen, D. Johansen, V. Farsi, W. Farstad, B.Høgtun and P.Knudsen

Department of Computer Science,
Institute of Mathematical and Physical Sciences,
University of Tromsø,
N-9037 Tromsø, Norway

Abstract

This report presents the software architecture in the StormCast meteorological workbench. The focus is on the software interfaces in StormCast version 2.1. For each layer in the architecture the functionality, the call structure, the interfaces, the event diagrams and test data are described. In addition, further works are discussed. The identification of software interfaces are based on the StormCast prototype implementations and evaluations. The motivation is to derive an application program interface (API) to StormCast.

1 Introduction

The StormCast project attempts to increase the potential for distributed computing through the construction of realistic distributed applications within the weather and pollution domains. In our work on StormCast, we have developed prototypes of a new generation of distributed applications for the meteorology and pollution domains.

This document presents the software architecture layer interfaces between the modules in different layers of StormCast 2.1. The motivation has been to specify an application program interface (API) that can be used by StormCast programmers. In this way, we will construct a software standard for this kind of applications. The paper briefly gives a motivation for standard interfaces. Then each of the six layers in the StormCast architecture are presented. For each layer, we outline the functionality, the call structure, the interfaces, the event diagrams and test data.

2 Why standard interfaces?

Software engineering concerning long-lived systems recognizes the signification of standardization and software standards. Standards are needed in various areas of software engineering. Standards are developed at different levels of generality and by different actors. Some of the standards which have been developed are (Sommerville, 1992):

1. Programming language standards. Standards for Ada, COBOL, Pascal, C, Fortran, etc.
2. Operating system standards.
3. Networking standards.
4. Window system standards.

Our approach toward these issues is to build standardized interfaces for each component of the application, through which other components may communicate with this component. Software standardization is motivated by a number of factors. Compatibility, portability, adaptability, extensibility, scalability, reusability and maintainability of a software system are all enhanced by standardization. In the following, the relationship between standardization and these properties is discussed.

2.1 Software compatibility and portability

Compatibility is defined as the ability of two components to be in a relationship: usually to communicate or work together, or to be interchangeable (Morris, 1993). Usually the term software compatibility refers to a software system's ability to work on other hardware and software systems (virtual machine) than it was originally designed to work with. In particular software compatibility means the ability of a program to accept data and control information from another program without conversion of data or modification of the program code, or the ability of a new version of software to replace functions of the old version. Diversity of software systems makes the compatibility a serious issue in software engineering. The main way to achieve such a property is by developing standard interfaces through which different software components can communicate and exchange data and control information.

Software portability is an issue closely related to compatibility. A software system is adaptable if it can be easily and conveniently transported from one environment to another. Software based on standard interfaces should be able to adapt to new environments. Standards are portability interfaces a system needs to include in order to reduce the portability problems when the software is moved to another machine.

Because of the emergence of workable standards and their acceptance by large segments of the software development community, it is now easier to produce portable, reusable application systems and components than it was in 1970's and 1980's. If a standard is adopted by the implementors of an application system, that system should be portable, without change to any other systems who follows that standard (Sommerville, 1992).

2.2 Software scalability and extendability

Scalability and extensibility refer to a software system's ability to grow. A scalable software system is a system that can easily cope with the addition of users and sites, and whose growth involves minimal expense, performance degradation, and administrative complexity (Satyanarayamanan, 1993). By extensibility we mean the ability of a software system to easily include new services and functions.

Efforts made to enhance the scalability and extensibility of different systems, specially distributed systems and applications, conclude that modular, layered architecture, decentralized solutions and diversity of mechanisms are the basic means to enhance scalability and extendability in a software system. Having a modular, layered architecture, the focus will have to be on the interactions between the modules in the different layers of the architecture (i.e., the interface between different components in the architecture). Software systems offering standard interface reduce the necessary effort to scale and extend them. Ideally speaking, such a software system could integrate new sites, physical and software components by offering them the standard interfaces to be attached to.

2.3 Software maintainability and reusability

Maintainability and reusability of a software are both motivated first and foremost by the economical factors. Maintenance costs are, by far, the greatest cost incurred in developing and using a system. In general, these costs are dramatically underestimated when the system are designed and implemented. On average, maintenance costs seem to be between two and four times for large embedded software systems (Sommerville, 1992).

Reusing software and software components can significantly reduce software costs and increase software quality. Reuse is an engineering discipline, based on maximum use of existing components. Clearly reusing the components can reduce the development cost and time. Components and systems which are used again tend to evolve and be more robust.

3 The StormCast architecture and interfaces

The StormCast application has been developed in several versions (Hartvigsen and Johansen, 1988, 1990, Johansen and Hartvigsen, 1991, Johansen 93). The architecture of StormCast 2.1 is illustrated in Figure 1.

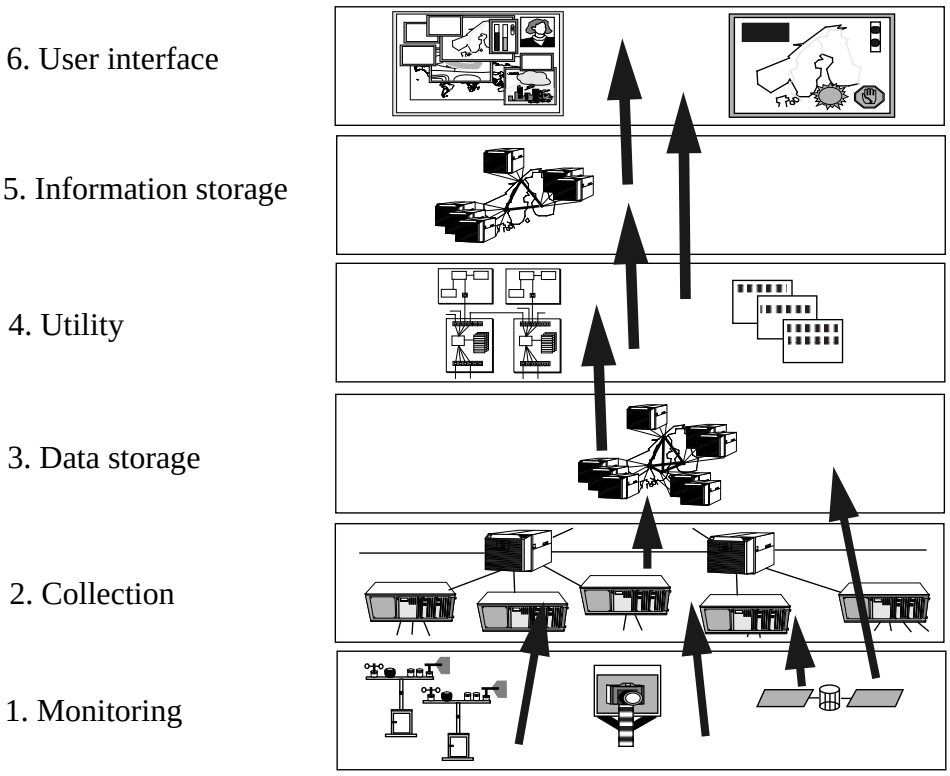

FIGURE 1. The StormCast 2.1 architecture.

In the StormCast 2.1 all interactions between the modules are based on a generalized RPC-semantic, using the ISIS distributed toolkit (Birman and van Renesse, 1994). By generalized RPC we mean that a client may send a request to one or a group of processes and waiting for replies from one or several servers. This is achieved by utilizing the group communication primitives `abcast`, `cbcast` or `gbcast` offered by ISIS. In the following chapters the layers are individually described.

4 The Monitoring Layer

4.1 Functionality

This layer consists of a set of loggers and monitoring stations. A logger is a dedicated computer which is connected to a few sensors. The logger converts analog input from the sensors to digital values and calibrates the data to physical quantities. The logger also stamps the data with the current location and time values.

4.2 Call structure

The services at the Monitoring Layer is called by the Collection Layer modules. See Section 5 on page 6 for a description.

4.3 Interface description

The current version does not implement a standard programming interface for this layer. (See the discussion below.)

4.4 Further work

StormCast 2.1 is based on the assumption that the Collection Layer consists of dumb data loggers and other data sources like video, audio or devices that offer satellite pictures. This means that the Collection Layer processes have to deal with different physical device interfaces. An enhancement of this strategy will be to introduce an additional software layer. This layer should offer a consistent interface (or API) to all collection processes, i.e., that each physical unit must be supplied with a corresponding driver that offer this API. This software layer, or *driver*, could be introduced either in the Collection Layer or in the Monitoring Layer. Figure 2a shows the driver in the Collection Layer. By introducing intelligence at the Monitoring layer the driver may be used in this layer (Figure 2b). The advantages of having a standard API that the collection processes use to connect to physical units are obvious. Physical units from different vendors could be changed without changing the collection code. The specification of this API must be a super-set of the functions and services that the physical units (data loggers) offer.

There are several arguments for introducing intelligence at the Monitoring Layer. One reason is when logger stations are available only trough (unreliable) wireless communication (radio/satellite). Examples of this are logger stations placed on sea vessels and at distant places. A data logger on a sea vessel could send its data at preset intervals via a radio or satellite link. However, a more robust and flexible solution is to connect the logger station to a computer and let the computer take care of unreliable communication (through building a more reliable protocol). The advantage of this is that the communication becomes more reliable and that unmanned distant logging stations can be controlled from a central site.

Farsi (1994) suggests to use a name server in each StormCast layer. A name server for the Monitoring Layer maps logical monitoring device names to physical addresses. Collection process programmers then use location transparent logical names instead of physical addresses. This name server may also keep track of which monitoring servers are available in each collection modules area. For instance mobile monitoring devices could update this server with location information.

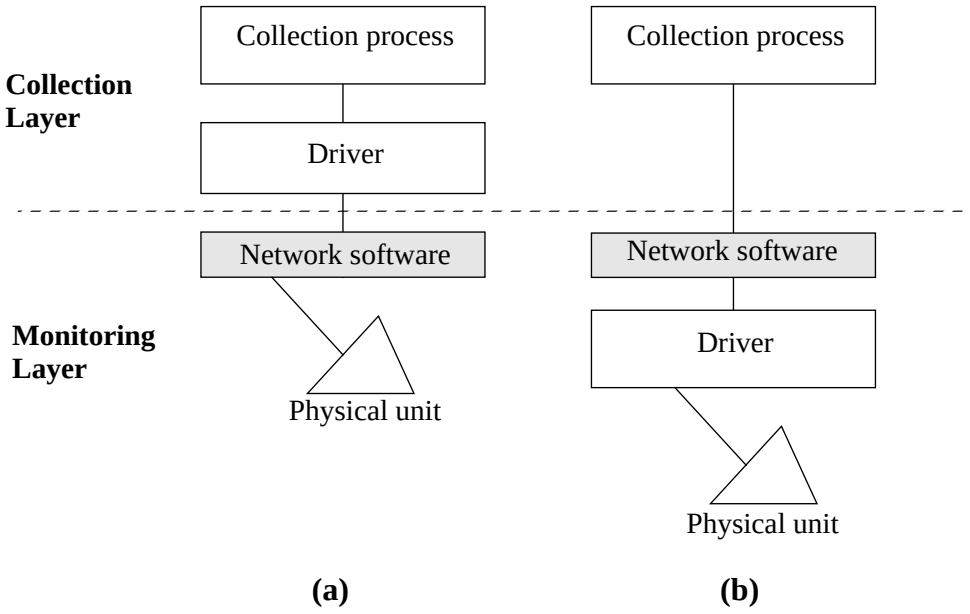

FIGURE 2. Drivers in the monitoring layer. Figure a) shows a dumb driver located at the Collection Layer. Figure b) illustrates an intelligent driver at the Monitoring Layer.

5 The Collection Layer

5.1 Functionality

The Collection Layer collects the monitored data from the bottom layer and filters out data to be transmitted higher up in the architecture. This layer consists of a set of replicated collection modules. Each replicated module is responsible for collecting of raw data from one or several geographical areas. Higher layers can request data from a certain area and the Collection Layer is responsible for carrying out this request.

For availability, the requested data will also be cached locally in this layer. If current data is impossible to obtain from the Monitoring Layer, cached data will be returned as response on requests from higher layers. The higher layers can validate the data based on its timestamp.

The Collection Layer also makes StormCast more transparent to the users, because the users do not have to worry about the format of the data, how they actually are retrieved etc. The layer converts raw data according to specific rules, and make sure they are time- and date-stamped. It is possible to get further information regarding logger stations and their sensors. This is partly based on the concept of a Management Information Base (MIB) (Robertsen, 1992).

5.2 Implementation description

The implementation of the Collection Layer is realized using the ISIS toolkit (Birman and van Renesse, 1994). The Collection Layer has four entry-points. Other layers may send requests to the Collection Layer to any of these entry-points. The information and functionality found in the Collection Layer is accessible through these entry-points.

There are three entry-points which may be used to retrieve information from the MIB. The final entry-point provides weather data collected from the loggers. The data received are explicitly assumed to be correct. No checking is done with respect to the meaning of the data.

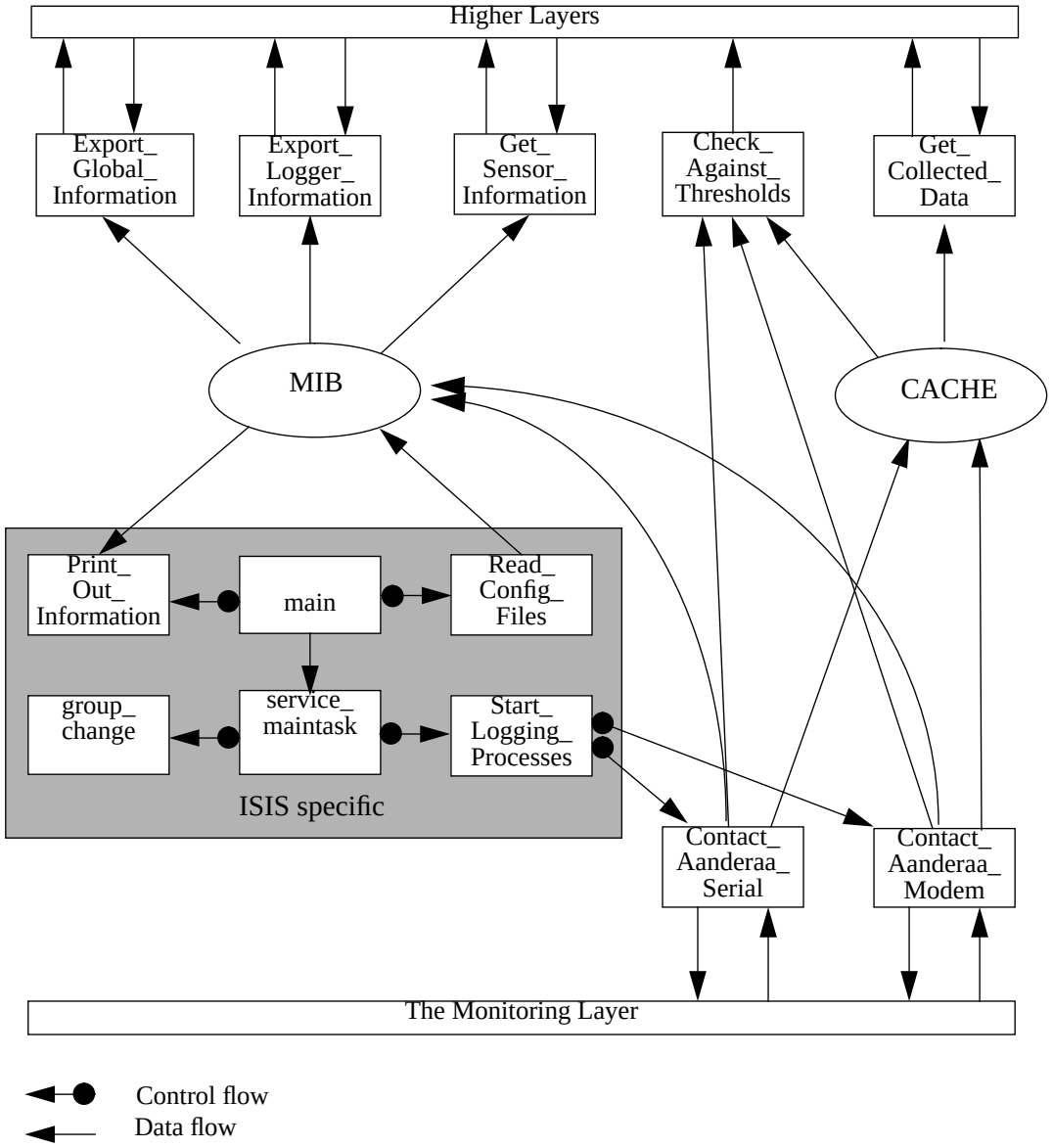

FIGURE 3. The design of the Collection Layer.

The design of the actual program which makes up the Collection Layer is shown in Figure 3. As illustrated the interface with other layers is realized by the procedures located at the top- and bottom of the figure. Information about the monitoring stations is found in a configuration file. This file is read at start-up time.

The Collection Layer uses a local cache from which requests are answered when logger stations are unavailable. This cache acts like a FIFO-queue, and may be either a disk file or some allocated memory.

Incoming messages will be received from any of the higher layers in the model. These messages will all request some kind of data. The message will contain a specification explaining which logger to contact. This will be one of the logger stations connected to this particular instance of the Collection Layer. The Collection Layer will now act like a server, and be responsible for getting data from the appropriate logging stations.

Data received from the Monitoring Layer will, if necessary, be transformed into a more appropriate format. This data is also time-stamped at once they arrive at the Collection Layer. The next step is to check the values in the collected data against the threshold values. The data from the Monitoring Layer may be received through a special constructed library for either serial connection or modem connection.

5.3 Interface description

Get_Collected_Data (name_specification, environment_specification)

Purpose: Receives a request for weather data, reads the contents of the cache and sends a reply to the calling process.

Called by: The Data Storage Layer, The Utility Layer.

Calls: None.

name_specification

The geographical name of the logger the calling process wants to retrieve data from.

environment_specification

The name of the environment, i.e. the sensor, the calling process wants to retrieve data from. Specified as “ALL” if the calling process wants data from all sensors at the same time.

Export_Global_Information (name_specification)

Purpose: Retrieves information like geographical name, coordinates and logger type concerning a logger connected to this instance of the Collection Layer. This information is found in the MIB.

Called by: The Utility Layer.

Calls: None.

name_specification

The geographical name of the logger the calling process wants to retrieve information from.

Export_Logger_Information (name_specification)

Purpose: Retrieves the environment name of all sensors of a specified logger connected to this instance of the Collection Layer. This information is found in the MIB.

Called by: The Utility Layer.

Calls: None.

name_specification

The geographical name of the logger the calling process wants to retrieve information from.

Get_Sensor_Information (name_specification, environment_specification)

Purpose: Used for retrieval of information about one of the sensors at a specified logger connected to this instance of the Collection Layer. This information is found in the MIB.

Called by: The Utility Layer.

Calls: None.

name_specification

The geographical name of the logger the calling process wants to retrieve information from.

environment_specification

The name of the sensor the calling process wants to retrieve data from.

Contact_Aanderaa_Serial (i)

Purpose: Gets weather data from an Aanderaa station connected to the Collection Layer through a serial line.

Called by: Start_Logging_Processes.

Calls: The Monitoring Layer.

i

The index number in the list of loggers connected to this particular instance of the Collection Layer.

Contact_Aanderaa_Modem (i)

Purpose: Gets weather data from an Aanderaa station connected to the Collection Layer by a modem.

Called by: Start_Logging_Processes.

Calls: The Monitoring Layer.

i

The index number in the list of loggers connected to this particular instance of the Collection Layer.

5.4 Event diagrams

The interaction pattern in StormCast does not depend of some kind of ordering of events. This means that interaction between modules (layers) theoretically could be based on any type of data communications. At present StormCast is based on the ISIS distributed toolkit. The motivation for the use of ISIS is its group-communication utility. In StormCast, process groups are a vital mean of increasing the overall fault-tolerance.

There are also various requirements for reliability of the communication between modules. This depends on both the modules and the layer. If all modules are placed in a local area network the reliability is higher than if some modules were placed i.e. on sea vessels, communicating via a satellite or a radio link.

Figure 4 shows a typical event diagram for a connection between the Data Storage Layer and the Collection Layer where the communication between the layers is based on an unreliable satellite multicast RPC protocol (Farstad, Johansen and Hartvigsen, 1994). Here, the monitoring devices may be servers placed on mobile sea vessels, which are contacted via satellite. The communication is of such type that messages could be lost in both directions. (It is not always convenient or desirable to build a reliable communication protocol on top of an unreliable satellite link because of the system delays).

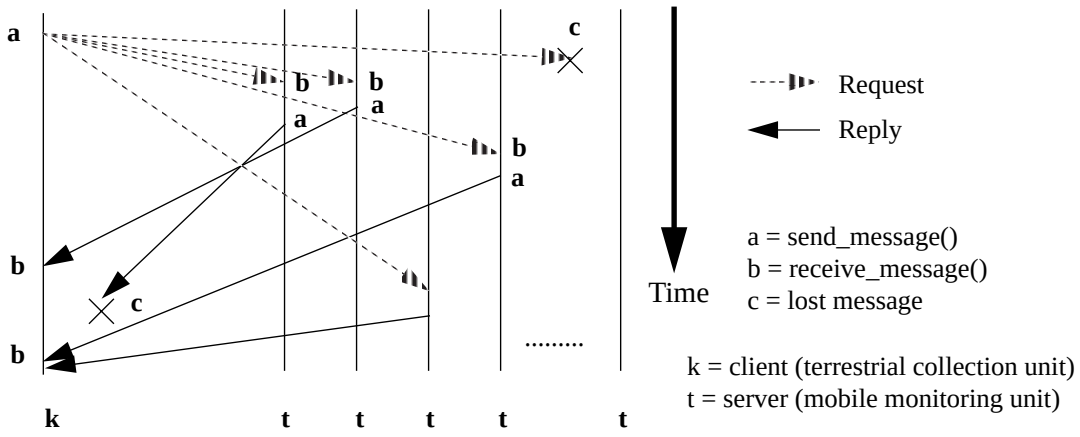

FIGURE 4. Data Storage Layer – Collection Layer connection events.

5.5 Further work

During the implementation and test phase, we have only used logger stations from Aanderaa, connected through a serial line or a modem. Adding more routines for communication with other kind of collection-modules enables the Collection Layer to receive data from other possible sources.

The StormCast 2.1 does not contain any name servers. The application is based on letting the Collection Layer map data requests from an area to the monitoring devices in the requested areas. For mobile monitoring devices (i.e. logger stations placed on sea vessels) this means that the Collection Layer continuously must be updated with new location information. As suggested in (Farsi, 1994) there should be a name server in each layer in the StormCast architecture to avoid scaling problems. The collection name server must therefore do the mapping between high-level area name / monitoring device name to physical address. The servers offering monitoring services must be named in a location transparent way. But, requests for data are based on geographically locations, meaning that the request contains location information. This location information must therefore be mapped to the physical address to the collection module that handles the requested area. In this way, location transparency is a matter of naming the predefined servers in a transparent way.

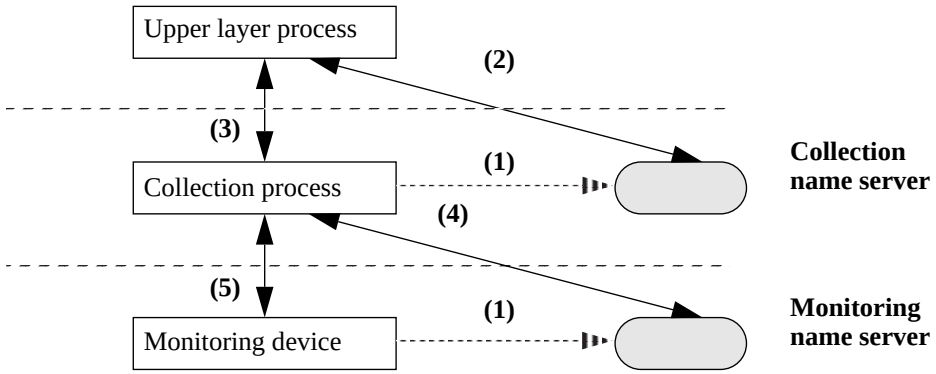

FIGURE 5. Naming in StormCast (based on Farsi, 1994)

Figure 5 illustrates a possible interaction pattern (Farsi, 1994). First the modules are registered in the name servers (1). The upper layer process maps an area to an address (or addresses) which again is mapped to a collection server that handles this area by contacting the collection name server (2). Then the given server is contacted (3), which in turn uses the monitoring name server to check which monitoring devices are available (4). The returned physical address(es) are used to contact the monitoring device (5).

6 The Data Storage Layer

6.1 Functionality

The Data Storage Layer is the first layer where data is permanently stored. Also this layer consists of a set of replicated modules. These modules send request about data from certain geographical areas at predefined frequencies to the Collection Layer modules. These modules store the obtained data and reply with retrieved data when requested from higher levels.

The Data Storage Layer provides a high degree of transparency to the other modules in the architecture. A request to the Data Storage Layer needs only to contain a short specification of the start and stop time and the location. The Data Storage Layer accomplishes the request according to the specifications given. The data retrieved is mostly used by the modules found in the Utility Layer, which use them for services as statistics and weather prognoses.

6.2 Implementation description

The communication with both the Collection Layer and the higher layer takes place by the broadcast mechanisms in ISIS only. These broadcasts will be the only way to communicate with the Data Storage Layer. As illustrated in Figure 6, there are two ISIS entry-points which the higher layers may access when old data is retrieved (`Retreive_Old_Data`) or miscellaneous data (`Store_Misc_Data`) is stored.

The processes in this layer send a request to the Collection Layer at predefined frequencies to get data to be stored permanently. This version of StormCast appends new data from a logger station to a sequential file dedicated to data from this logger station. There is one separate file for each date.

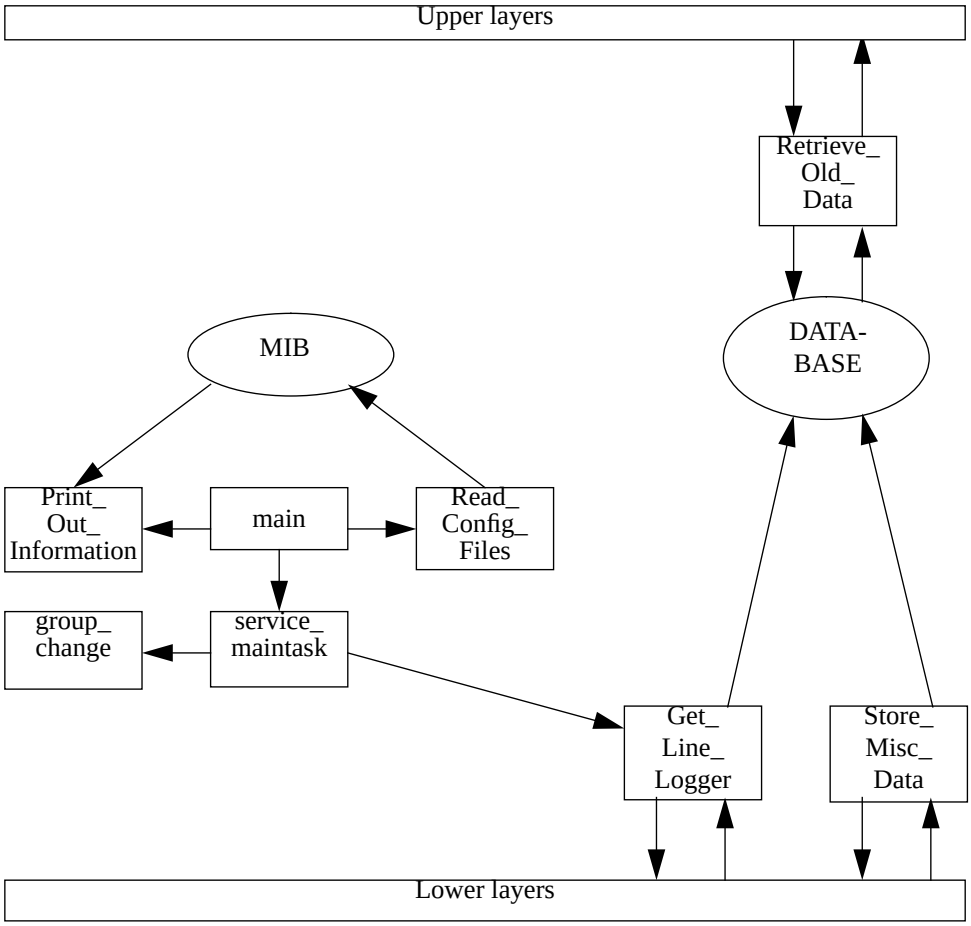

FIGURE 6. The design of the Data Storage Layer.

6.3 Interface description

Retreive_Old_Data (name_specification, start_date, stop_date, start_time, stop_time)

Purpose: Receives a request for old data, reads the contents of the database and sends a reply to the calling process.

Called by: The Utility Layer, The Interface Layer.

Calls: None.

name_specification

The geographical name of the logger the calling process wants to retrieve data from.

start_date

The first date of the requested data.

stop_date

The last date of the requested data.

start_time

The start time on the start_date.

stop_time

The stop time on the stop_date.

Get_Line_Logger (name_specification)

Purpose: Sends a request to the Collection Layer at predefined intervals to get recent weather data. These data are later to be stored in the database.

Called by: service_maintask

Calls: Get_Collected_Data (in The Collection Layer).

name_specification

Name of the logger the Data Storage Layer wants to retrieve weather data from.

Store_Misc_Data (name_specification, datestamp, timestamp, exceeded_values)

Purpose: Receives a request to store weather data in the database. The request contains further specifications about the data to be stored.

Called by: The Collection Layer, The Utility Layer.

Calls: None.

name_specification

The geographical name of the logger related to the data.

datestamp

The date the data are collected.

timestamp

The time the data are collected.

exceeded_values

A list of the actual values of the observation.

6.4 Event diagrams

See the event diagram for the Collection Layer in Figure 4 on page 11.

6.5 Further work

Permanently stored data, especially multimedia data and satellite pictures lead to a huge amount of data. Therefore it is reasonable to consider some compression techniques.

In the StormCast 2.1 all data are stored in sequential text files. This is a simple and for most purposes an adequate solution. However, when searching in huge amounts of data, indexing should be used for efficiency. Different DBMSs therefore have to be considered. As mentioned a multimedia data storage, including a multimedia database, will be addressed in the project.

7 The Utility Layer

7.1 Functionality

The Utility Layer contains different types of services (applications) which convert raw data into information. The modules in this layer request data from the Collection Layer, the Data Storage Layer or other applications.

7.2 Implementation description

Due to the functionality of the applications, we have called them tools. The design of each existing application is not a task of this document, however the design of the applications connection to the upper and lower layers are described here.

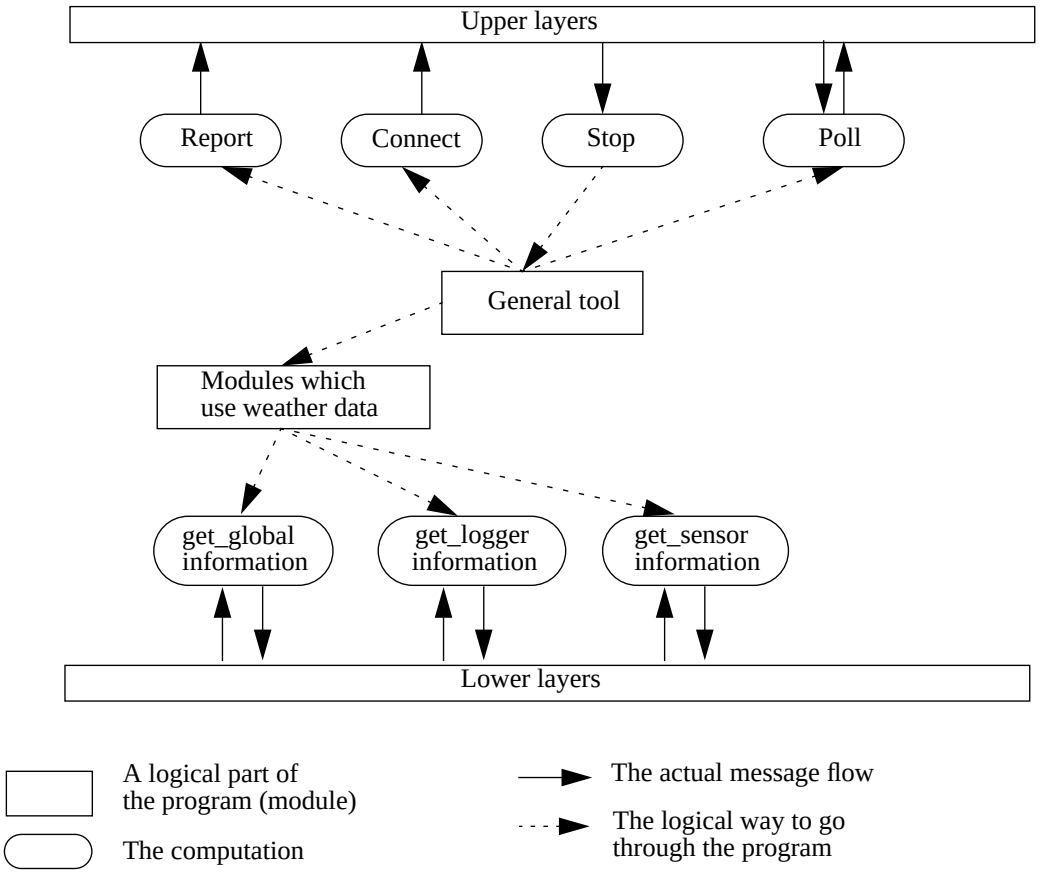

FIGURE 7. The design of the Utility Layer.

7.3 Interface description

The following are actions that any tool can perform.

Report (text)

Purpose: Sends an error message to the ToolManager. (See Section 9.1 on page 19).

Called by: Any application.

Calls: The Interface Layer.

text

The text to be displayed in the ToolManager.

Example located in: LoggerManagerIF.c

ToolManagerConnect (id)

Purpose: Reports to the ToolManager that we are up and running.

Called by: Any application.

Calls: The Interface Layer.

id

Identifies of the tool which performs the connection.

Example located in: ToolManagerConnect(), LoggerManagerIF.c

Poll(id)

Purpose: Receives a poll message and sends the identification as an ack.

Called by: The Interface Layer.

Calls: None

id

Identifies the tool which wants to check the status of the ToolManager.

Example located in: ReceivePoll(), LoggerManagerIF.c

Stop(id)

Purpose: Receives a stop message and terminates if the tool's id equals the one in question.

Called by: The Interface Layer.

Calls: None

id

Identifies the tool to stop.

Example located in: ReceiveStop(), LoggerManagerIF.c

get_global_information(name_spesification)

Purpose: Gets the information about a logger. The information is the geographical name of the logger, the location coordinates and the logger type.

Called by: Any application.

Calls: The Collection Layer.

name_specification

Name of the wanted logger.

Example located in: doQuery(), TreeIF.c (The Locator application)

get_logger_information(name_specification)

Purpose: Gets a list of names of all sensors attached to the specified logger.

Called by: Any application.

Calls: The Collection Layer.

name_specification

Name of the wanted logger.

Example located in: FillInTree(), LoggerManagerIF.c

get_sensor_information(name_specification, environment_specification)

Purpose: Gets sensor values from one or all sensors in the specified logger. Values will be the last updated.

Called by: Any application

Calls: The Collection Layer

name_specification

Name of the wanted logger.

environment_specification

The name of the environment, i.e. the sensor, the calling process wants to retrieve data from. Specified as “ALL” if the calling process wants data from all sensors at the same time.

Example located in: doQuery(), QueryObject.c (The LoggerManager application)

8 The Information Storage Layer

8.1 Functionality

In the StormCast application a distinction is made between raw data and information (processed data). This layer provides the same set of services as the Data Storage Layer. Thus, the Data Storage Layer and the Information Storage Layer are quite similar. The only difference between these to layers is the concept of stored entities. Because of this similarity, the Information Storage Layer is not discussed here.

9 The User Interface Layer

9.1 Functionality

This layer creates and manages the user dialogues. The User Interface Layer has mainly to tasks:

1. to collect data and information requested by the user, and
2. to present the collected data or information to the user.

Typically, a client module is activated for each requested service. This client module interacts with one or several services, logically found in lower layers of the architecture to get the information requested. The presentation of data is done by using weather maps and other visualization techniques.

The management of user dialogues includes the following tasks:

1. to hold a list of the tool states, and to update it every time any tool changes its state.
2. to offer some environment to manage to start new tools and stop running tools.
3. to display to the user messages from any tool (error messages or state messages).

The managing process is called the ToolManager. In order to start a new tool the ToolManager performs a command which contains the tool's name and a few user parameters set by the user. Communication between the ToolManager and the tools is performed through Isis. The ToolManager also uses Isis to monitor the state of the tools. Every time a tool fails the Isis system will trigger a polling action from the ToolManager.

9.2 Implementation description

The ToolManager interface is build with the use of the XWindows system. The design of the ToolManager can logically be divided into two parts, the interface part and the execution part (see Figure 8). The latter consists of callback actions activated from the interface part or Isis tasks activated by messages from lower layers.

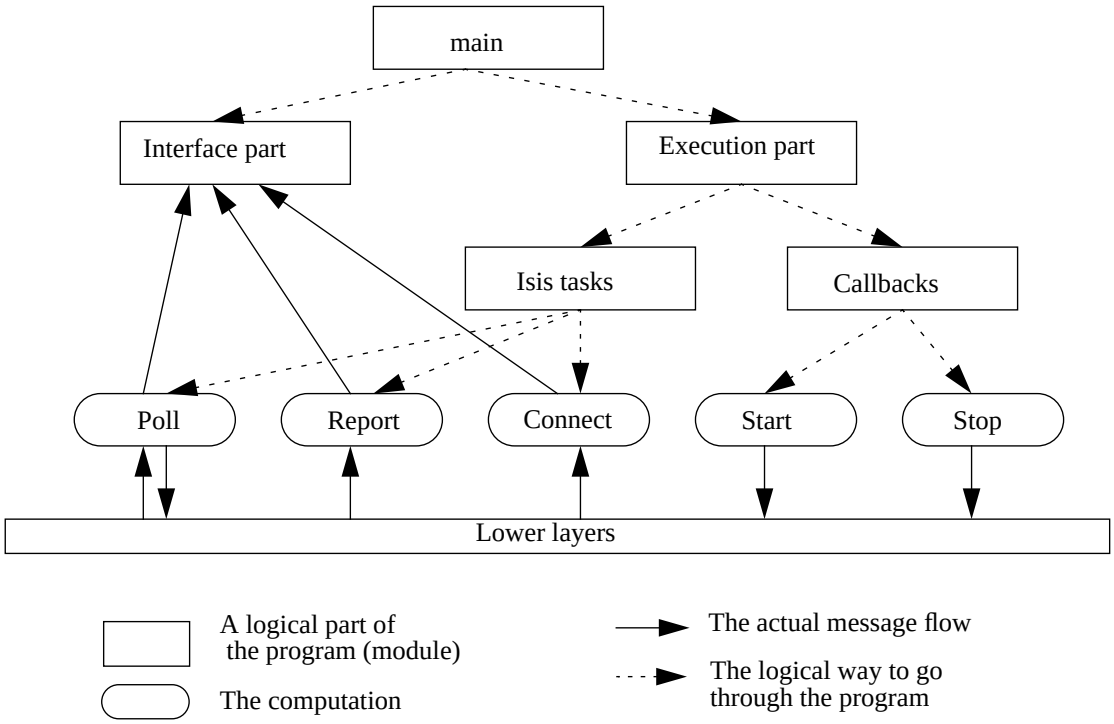

FIGURE 8. The design of The Interface Layer.

9.3 Interface description

An action in the ToolManager often propagates through several procedures. The action as described will be located in the procedure as close to the actual execution as possible. The call structures described are those which in any way crosses the architecture layers (the round boxes in Figure 8).

StartTool (from_button, client_data, *callback)

Purpose: Starts the specified tool by calling a shellscript and executing a command with the parameters read from the interface window.

Parameters: XWindows administrative purpose.

Called by: Callback from ToolManager menu.

Calls: None

Located in: TmDialogsIF.c

StopTool (w, client_data, *callback)

Purpose: Sends a stop message to a tool in the Utility Layer.

Parameters: XWindows administrative purpose.

Called by: Callback from ToolManager menu.

Calls: None

Located in: ToolManagerIF.c

Connect (id)

Purpose: Receives start confirmation from a tool, and changes the tool status.

Called by: The Utility Layer

Calls: None

id
Identifies the tool just started.

Located in: ConnectEntry(), ToolManagerIF.c

Report (text)

Purpose: Receives reports from tools to be displayed to the user.

Called by: The Utility Layer

Calls: None

text
The text to be displayed.

Located in: ReportEntry(), ToolManagerIF.c

PollTools ()

Purpose: Sends a message to each running tool. Expects the tool to ack this message, identifying itself with its id number. If not ack'ed it assumes that the tool is dead and notifies the user about this.

Parameters: None

Activated by: a change in the tools group.

Calls: The Utility Layer

Located in: ToolManagerIF.c

10 Discussion and open problems

The design of a toolbox for the construction of distributed (monitoring) applications is one of the main objectives of the StormCast project. We also try to generalize aspects of this application which later can be used in design and implementation of distributed monitoring applications. Obviously, standardized interfaces between different components ease the construction of such applications. In the design of large scale distributed applications, it is necessary to choose an optimal solutions.

One of the basic issues in designing a distributed application is how to reduce the communication costs. As described earlier, communication in StormCast v.2.1 is mainly based on a generalized synchronous RPC mechanism. However, whenever a continuously demand for data is concerned, synchronous communication means an unnecessary delay.

Software management is one of the complex issues in any distributed application and system. The problem in a large-scale distributed system is to provide system managers with the means to monitor and adjust a fairly large collection of different types of geographically distributed components (Schroeder, 1993). One approach to this problem, as described in (Schroeder, 1993), requires that each component defines and exports a management interface, using RPC if possible. Each component is managed via calls to this interface from interactive tools run by human managers.

This paper has reported on the first attempt to form an application program interface to StormCast and StormCast-like applications. So far the six layer architecture has appeared to conduct a suitable framework for this. As it appears from the former sections, the architecture is not completely followed in StormCast 2.1. The reason for this is mainly that version 2.1 is assembled from several earlier versions.

11 Concluding remarks

In this paper we have described our effort toward the standardization of software components in the StormCast application. We have discussed the general motivations for such a work, the layered architecture and interfaces between different layers. In addition, the standard interfaces used to communicate with different modules in the application have been

presented. The six layer architecture has turned out to be an appropriate way to construct this kind of applications.

References

Birman, K.P., Renesse, R. van (1994). *Reliable distributed computing with the ISIS toolkit.* IEEE Computer Society Press, New York.

Farsi, V. (1994). *Scaling and a Distributed Application*. Computer Science Technical Report 94-17, Department of Computer Science, University of Tromsø, Norway.

Farstad, W., Johansen, D., Hartvigsen, G. (1994). *Satellite Communication applied in a distributed application*. Computer Science Technical Report 94-XX, Department of Computer Science, University of Tromsø, Norway.

Hartvigsen, G., Johansen, D. (1989). StormCast – a Distributed Artificial Intelligence Application for Severe Storm Forecasting, In: M.G. Rodd, T.L. d'Epinay (Red.), *Distributed Computer Control Systems 1988.* Proceedings of the Eight IFAC Workshop (Vitznau, Switzerland, 13-15 September, 1988). Oxford: Pergamon Press, 1989, pp. 99-102.

Johansen, D. (1993). Yet another exercise in distributed computing. In: F.Brazier and D.Johansen (Eds.), *Distributed Open Systems in Perspective*. IEEE Computer Society Press, New York.

Johansen, D., Hartvigsen, G. 1991. StormCast – A Distributed Application. In: *Proceedings of the Autumn 1991 EurOpen Conference* (Budapest, Hungary, 16-20 September, 1991). European Forum for Open Systems, Buntingford, Hertfordshire, U.K., pp. 273-286.

Macro, A. (1990). *Software Engineering, Concepts and Management*, Prentice-Hall.

Morris. D., Tamm, B. (1993). *Consise encyclopedia of software engineering*, Pergamon press.

Robertsen, Ø. (1992). *StormCast versjon 2.0*. Semester thesis in Computer Science. Department of Computer Science, University of Tromsø, Norway. (in Norwegian)

Schroeder, M.D. (1993). A State-of-the-Art Distributed System: Computing with BOB. In: S. Mullender (ed), *Distributed Systems. Second Edition.* Addison-Wesley.

Satyanarayanan, M. (1993). Distributed File Systems. In: S. Mullender (ed), *Distributed Systems. Second Edition.* Addison-Wesley.

Sommerville, I. (1992). *Software Engineering. Fourth edition.* Addison-Wesley.