SCADA system for monitoring and control of small industrial producing unit

Hristo Nenov¹

Abstract – The topic of this paper is design and development of SCADA system for remote autonomous production units (Remote Automation Unit - RAU) from a "small farm" type. For the purpose of developing the material base of the farm is reduced to a limited number of controlled functional units which could vary depending on the particular implementation on a real production site.

Keywords – SCADA, MODBUS, Java, JPA, PLC.

I. INTRODUCTION

In modern economies it is almost impossible to imagine industries where computers and machines have not have taken control of a significant part of the production processes. The main reason behind this is the need to increase the competitiveness of production in the sector. The formula for achieving this goal is to increase the production quantity and reduce the production costs. The implementation of different tools for mechanization of the production creates an opportunity for that. Agriculture is no exception and modern technologies are becoming more widely used in this industry. The combination of the latest information technologies, global information network Internet, more available control devices and the use of renewable energy sources make the appearance of modern agriculture - autonomous, more efficient and predictable.

II. ARCHITECTURE OF THE SYSTEM

A. General scheme

Remote Automation Unit (RAU) from “small farm” type consist the following modules:

1. Greenhouses with the following adjacent manageable systems and sensors:
   - Drip irrigation system with controllable valve;
   - Sprinkling system with controllable valve;
   - Ventilation;
   - Measuring devices for air temperature;
   - Measurement devices for humidity.

2. Wells - local inexhaustible sources with the following adjacent manageable systems and sensors:
   - Pumping systems;
   - Sensors for water level.

3. Water storage reservoir - capacitive buffer source, which is fed with water from local inexhaustible sources and supplies water for the entire farm, the following adjacent manageable systems and sensors:
   - Pumping systems for water;
   - Water level sensor.

For the purpose of real testing, simplified configuration is selected and it is deployed as follows:

- Greenhouse with an individual drip irrigation system, sprinkling system, ventilation system, temperature sensor and an air sensor for humidity;
- Water storage supplying the greenhouse, with a pumping system for water supply and one sensor for water level;
- Two wells, each with a pumping system connected to the water supply and water retaining with one sensor for water level.

B. Physical layer

Physical layer - covers the management and transfer of data to and from various sensors and controllable devices (sensors, valves, relays). This layer is designed to the specific realities of production sites and goals.

B. Industrial Interface layer

The layer is implemented on one or more programmable logic controllers (PLC) and includes logic for gathering and initial processing of data to and from physical sensors and controllable devices and providing them to the layers of higher-level system. PLC interface has its own logic, which is connected to the other layers of the system by appropriate industrial protocols.

C. Application layer

The layer implements the application management logic. It consist two aspects:
- Management of autonomous remote objects (small farms);
- Business logic implemented on the Central System Proxy Server.

D. Presentation layer

The layer implements Human Machine Interface (HMI) for developing SCADA systems. This is layer of the highest level and provides a graphical interface for system operators.

¹Hristo Nenov is with the Faculty of Computer Science and Automation at Technical University - Varna, 1 “Studentska” str., Varna 9000, Bulgaria, E-mail: h.nenov@tu-varna.bg.
through which they can monitor the working condition, to read different measurements, to oversee occurred events, alarms and archival data

E. Data layer

The layer implements mechanisms for temporary and permanent storage of system data. It covers two aspects:
- Temporary storage of data within the RAU (individual small farms)
- Permanent storage of data source RAU on a main server.

Data collection (Snapshots) and their replication to a central server are performed by the SnapshotProducerService and SnapshotConsumerService, working in parallel on RAU and implementing the principle Producer / Consumer. The exchange service for MODBUS contains abstract controller class Modbus Controller, which implements operations via MODBUS composite instance of ModbusMaster - base library class in modbus4j. The heirs of ModbusController implement different MODBUS modes - TCP, RTU, ASCII, as appropriate instantiation type - successor to ModbusMaster.

**Fig.1. General architecture of the system.**

**III. REALIZATION OF THE SYSTEM**

Each Remote Automation Unit (RAU) is designed to operate in unreliable network connectivity to the central server, but it should provide reliable and trouble-free autonomous management of the production process based on the current configuration. In implementation ScheduledExecutor Service (Java 1.7+) is used to start services at regular configurable intervals:
- Activities of gathering information from the sensory part of the system, primary processing and storage of information in a local database.
- Logic of autonomous management of physical devices (pumps, valves, relays, etc.) based on a current configuration to ensure the needs of production.
- Generate and store events and alarms;
- Send the collected data, events and alarms to the central server.

**Fig.2. Service and Controller for data exchange over MODBUS.**

Updating the status of RAU (Remote Automation Unit). The internal state of RAU is managed by the class DeviceManager, which maintains a collection of devices – “Device”, which represent the inputs and outputs of the PLC. Synchronization of values with those of the registers of the PLC, is carried out by ModbusService, which uses class ModbusController, encapsulating work with MODBUS. ModbusService and DeviceManager sharing data for the state through class Snapshot as the exchange is initiated only from ModbusService.

**Fig.3. Updating of state of RAU.**

**Fig.2. Service and Controller for data exchange over MODBUS.**
Model of automatic control of RAU.
Automatic control is realized by class AutomationController, which applies logical chains of rules. Each type of rule implements specific logic. This mechanism realizes design pattern “Chain of responsibility” and relies on polymorphism technique.

IV. TESTING OF THE SYSTEM

For testing purposes of the developed SCADA systems, experimental arrangement that reflects the structure of a small autonomous farm RAU and a central server are realized. The experimental setting is made up of the following components:
- Central server on which the application runs RauServer. Implemented on a separate computer that is running Apache Tomcat with tuned Jersey Servlet, REST service requests from RAU. On the same machine instance is running for MySQL database on which stored data are received from RAU;
- RAU (Remote Automation Unit), which was launched as a standalone application (service) on a separate computer. On the same machine is running instance of MySQL database on which the data is temporarily stored until it is sent to them to the central RauServer;
- MODBUS Controller.

V. CONCLUSION

The system is fully operational. All modules are working properly, generating and exchanging correct data. The main objective of the study is achieved. There are a few aspects, which are provided for future development.

REFERENCES

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides “Gang Of Four - Design Patterns, Elements Of Reusable Object Oriented Software - Addison Wesley
[3] Kevin Mukhar, Chris Zelenak, James L. Weaver, Jim Crume JavaEE 5, From Novice to Professional
Abstract – There is a huge gap between the speeds of the processor and the SDRAM memory. Therefore, lots of efforts are being made to decrease the SDRAM latency. One of the ways to achieve this goal is predicting whether to use the Open page or the Close page DRAM policy, i.e. dynamically switching from one policy to the other during program execution. In previous work the author of this paper considered such predictors and in this paper improvements of the dead time predictor are proposed. The dead time predictor is based on access interval time values. When a value that is equal to the last access interval time, multiplied by 2 or 4, elapses, it is predicted that the row has entered its dead time and a row precharge command is issued. Instead of multiplying only by 2 or 4 (as in previous work), in this paper several multiplication values, which are fixed throughout the whole program execution, are tried. Then an adaptive mechanism, which dynamically changes the multiplication value during program execution, is introduced. The gained adaptive dead time predictor yields the best results compared to all the variants with fixed multiplication values, with very little additional hardware requirements.

Keywords – SDRAM, latency, adaptive, dead time, predictor.

I. INTRODUCTION

A desire for better potential utilization of processors, which are becoming faster and faster, demands a memory system with similar performance enhancements. A critical link in a hierarchically organized memory system is the main memory, implemented with chips of DRAM memory (Dynamic Random Access Memory). There is a huge gap between the speeds of the processor and the DRAM memory, called memory gap or memory wall [1], which is also increasing exponentially. The term wall points to the fact that if the trends continue, at one moment in the future we will hit that memory wall, and when this happens the computer system speed will no longer depend on the processor speed at all - it will be completely determined by the memory speed.

The processor-memory speed gap is generally being solved in two ways. First, cache memories are introduced. They are made of fast SRAM memories, with speeds similar to the processor's speed, so they are able to deliver all the needed data in time. But still, when cache misses occur and DRAM accesses are to be made, there are huge delays, long several tens of processor clock cycles. Additionally, these delays are also constantly increasing. This is the reason that various improvements of DRAM memories are also being introduced. These improvements include: synchronized DRAM memories (SDRAMs), larger numbers of independent banks, burst mode, various technological and scheme improvements. The main consequence of these DRAM improvements is increased DRAM bandwidth (the quantum of data transferred per unit of time), but the latency (the first part of the delay, until the first datum arrives) is still large. Namely, DDR, DDR2, DDR3, etc. SDRAM memories all have practically the same DRAM core, with very little differences in the latency parameters. What changes is the bandwidth - DDR2 has twice the bandwidth of DDR, DDR3 has twice the bandwidth of DDR2 [2], etc. On the other hand, the latency changes much more slowly - only about 5% per year [3]. Therefore, in each new DRAM generation the latency part is becoming more and more dominant in the total delay time. It can be calculated that the latency part, when transferring a 128 B data block from the DDR3 SDRAM to the cache memory, comprises about 70-90% of the total delay time. Therefore, lots of efforts are being made to decrease the SDRAM latency.

A classic SDRAM controller uses two possible policies (strategies): Open Page (Row) policy and Close Page (Row) policy. When using the Open Page policy, the accessed row is kept open, which yields low latency if the next SDRAM access is directed to the same row, and high latency if the next SDRAM access is directed to some other row. In the first case (open row hit), the latency is equal to \( t_{CAS} \), and in the second case (open row miss), the latency is equal to the sum \( t_{RP} + t_{RC} + t_{CAS} \) [2]. Here \( t_{RP} \) is the row precharge (closing) time, \( t_{RC} \) is the time needed for opening the row, and \( t_{CAS} \) is the column access time. When using the Close Page policy, each row is being closed after each access. This 'hides' \( t_{RP} \), so the latency is always the same - the sum \( t_{RC} + t_{CAS} \) [2]. A popular way to decrease the SDRAM latency is using a hybrid policy with help of predictors. In a perfect scenario, the Open Page policy should be used in cases of open row hits, and the Close Page policy should be used in cases of open row misses. If we correctly predict when to switch to Open Page policy we will have the latency of \( t_{CAS} \) (instead of \( t_{RC} + t_{CAS} \)) and if we correctly predict when to switch to Close Page policy we will have the latency of \( t_{RC} + t_{CAS} \) (instead of \( t_{RP} + t_{RC} + t_{CAS} \)). The problem is that any wrong prediction actually increases the latency, instead of decreasing it. Therefore, we should minimize the number of wrong predictions.

In previous work the author of this paper redefined the metrics related to cache memories from [4] to be related to DRAM memories, as follows. Live time is a time interval that elapses from opening a row in a bank until the last access into that row before its closing. Dead time is a time which elapses from the last access to an open row until the moment of its closing. Access interval is a time interval which elapses between two consecutive accesses to an open row in a bank.

Based on these redefinitions, three predictors were proposed [5], the simplest and cheapest of which is the dead time predictor (DTP). It keeps track of the last access interval time value for each of the banks. When a value that is equal to the last access interval time value, multiplied by 2 or 4 elapses, it is predicted that the row has entered its dead time and a proper

1Vladimir Stankovic is with the Faculty of Electronic Engineering at the University of Nis, Aleksandra Medvedeva 14, 18000 Nis, Serbia, E-mail: vladimir.stankovic@elfak.ni.ac.rs
row precharge command is issued. For some of the used benchmark programs multiplying by 2 was better option and for some of them it was multiplying by 4. I wanted to try with different values, like 8 or 16 and see the results. It is obvious that a greater multiplication value causes the system to have less number of misclosed rows (less number of wrong predictions that the row has entered its dead time), but it also causes less number of total predictions, i.e. greater number of omitted chances to close the row in advance. A less multiplication value, on the other hand, causes greater number of total predictions, which means a chance to have greater number of in-time row closings, as well as greater number or errors. It is logical that an ideal strategy would be to dynamically change the multiplication value, in order to adapt to the specific program, or specific part of the program, that is being executed.

II. RELATED WORK

There are number of papers that deal with improving performances of the memory system. Some of them focus on improving performances of the cache memory subsystem and some on improving performances of the SDRAM memory. Some of the papers that consider improving performances of SDRAM memories try to improve the bandwidth, some optimize the SDRAM's power management, while some of them try to decrease latency of the SDRAM memory. This section will focus only on papers and patents that try to decrease the SDRAM's latency, mostly using predictors.

M. Awasthi et al. [6] use a predictor that tracks the number of accesses to a given DRAM page to determine page closure decisions.

Y. Xu et al. [7] use solutions from the 2-level branch predictors to predict whether to close the open page or to keep it open, based on the analogy between the prediction of the branch as taken or not taken and the prediction of the next row as the same or different.

M. Chiyuan and C. Shuming [8] collect statistical information of address distribution of all the instructions waiting for accessing memory and analyze regular changes of access addresses in order to make judgment of the appropriate moment to precharge each DRAM bank.

M. Chiyuan and N. Xiaoqiang [9] propose a new memory schedule policy oriented to stream architectures. Their policy can predict the access behavior of each DRAM bank and uses appropriate controller policy.

Y. H. Son et al. [10] propose asymmetric DRAM bank organizations to reduce the average access latency.

S.-I. Park and I.-C. Park [11] use 2b state machines to predict whether the successive memory access leads to a row hit or not and change the memory mode according to the prediction.

E. Krimer et al. [12] use saturation counters to maintain predictions of whether pending loads will conflict with older pending store operations.

B. Fanning [13] proposes a method and apparatus of dynamically adjusting a memory system's existing paging policy. The method generates a select signal according to at least one input signal and the existing paging policy to the memory system and proceeds to modify the existing paging policy based on the generated select signal.

B. Emberling [14] proposes a predictive optimizing unit that is to be used with an interleaved memory, suitable in computer graphic systems. His unit maintains a queue of pending requests for data from the memory and prioritizes precharging and activating interleaves with pending requests.

III. SYSTEM SIMULATION MODEL

In this paper the same system simulation model and set of benchmark programs as in [5] was used. I integrated the program Sim-Outorder from Simplescalar 3.0 [15] with a DDR3 SDRAM simulator, written by myself. The simulated superscalar CPU issues 4 instructions on every clock cycle and supports out of order instruction execution, with a 2-level branch predictor. The CPU clock frequency was 3.2 GHz. There were 2 levels of caches - L1 contains separate instruction and data caches, both 8 kib large, with direct mapping and line sizes of 32 B, while L2 contains a unified cache, 2 MB large, with set-associative mapping, 4 lines per set, and line size of 128 B. All the cache memories use the write-back policy. The simulated DDR3 SDRAM has the following characteristics: 8 banks per chip, 8 k rows per bank, row capacity is 2 kB, \( t_{RP} = t_{RCD} = t_{CAS} = 12.5 \text{ ns} \).

<table>
<thead>
<tr>
<th>Benchmark program</th>
<th>ORH prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>perl</td>
<td>0.07</td>
</tr>
<tr>
<td>compress</td>
<td>0.09</td>
</tr>
<tr>
<td>mcf</td>
<td>0.11</td>
</tr>
<tr>
<td>bzip2</td>
<td>0.26</td>
</tr>
<tr>
<td>ipheg</td>
<td>0.33</td>
</tr>
<tr>
<td>gcc</td>
<td>0.36</td>
</tr>
<tr>
<td>cc1</td>
<td>0.37</td>
</tr>
<tr>
<td>m88ksim</td>
<td>0.47</td>
</tr>
<tr>
<td>li</td>
<td>0.53</td>
</tr>
<tr>
<td>anagram</td>
<td>0.70</td>
</tr>
<tr>
<td>go</td>
<td>0.72</td>
</tr>
<tr>
<td>go*</td>
<td>0.74</td>
</tr>
<tr>
<td>li*</td>
<td>0.80</td>
</tr>
<tr>
<td>m88ksim*</td>
<td>0.89</td>
</tr>
<tr>
<td>anagram*</td>
<td>0.90</td>
</tr>
<tr>
<td>compress*</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Executions of 11 benchmark programs were simulated: cc1, compress, ipheg, li, m88ksim, perl, go from SPEC95, bzip2, gcc, mcf from SPEC2000 and anagram. The last one is a program that finds all the possible words that can be obtained by combinations of the letters of 3 names: Todd Austin, Scott Breach and Guri Sohi. For 5 of the benchmark programs the simulated L2 cache size was large enough for the entire program's data to be placed in it. As a consequence the SDRAM had small number of accesses. In order to 'fix' this, versions with smaller L2 cache sizes were also tried for these programs, which yielded larger numbers of SDRAM accesses.
The versions with larger L2 caches are signed with a star (*) after the benchmark name. The new versions give variety to the benchmark program set, since they obtain rather different open row hit probabilities, as can be seen in Table I—the probabilities range from only 0.07 for perl to even 0.91 for compress*, with gradual growth in between. This variety of open row hit probabilities really tests the used predictor and its prediction accuracies.

IV. DEAD TIME PREDICTOR WITH FIXED MULTIPLICATION VALUES

The DTP (dead time predictor) is based on access interval time values. Simulation results show that the average dead time is much larger than the average access interval time, hence when a value that is equal to the last access interval time, multiplied by 2 or 4, elapses, it is predicted that the row has entered its dead time. To implement the DTP the DRAM controller needs several things:

1. There has to be one counter for each bank to take care of the elapsed time since last access. In order to minimize the counter's length, they may be triggered with a signal derived by dividing the DRAM's clock.
2. A register for each bank is needed for storing the last access interval time. Every time there is an open page hit in any of the banks that means occurrence of a new access interval time, hence the counter's value for that bank should be stored into the proper register and the counter should be reset. The operation of multiplying by 2 or 4 could be implemented by fixing the least significant bit to zero (the two least significant bits if multiplying by 4) and then storing into this register from the next position(s).
3. The DRAM controller also needs one comparator for each bank. In case the counter's value becomes greater or equal to the register's value the row is to be closed.
4. We also need a queue with orders for bank precharges from the DTP. The SDRAM controller performs a queue read operation and issues a row precharge of the proper bank(s), every time it is idle.

First I tried simulating 8 variants of the DTP: multiplying by 2, 4, 8, 16, 32, 64, 128 and 256. The gained average DRAM latencies, expressed in processor clock cycles, are shown in Fig. 1. We may notice 2 groups of programs. In the first group, which comprises the first 7 programs (perl to cc1) the strategy x2 (multiplying by 2) is the best and the strategy x256 (multiplying by 256) is the worst. For some of the programs in this group there are significant differences between x2 and x256. For the rest of the programs the "middle" strategies (x8, x16 or x32) are mostly the best, while x2 and x256 are the worst. These results suggest trying an adaptive mechanism, which dynamically changes the multiplication value. The goal is to obtain performances similar to the performances of x2 in the first group of programs and similar to the performances of x8, x16 and x32 in the second group. Another advantage of such a mechanism might be the fact that the optimal multiplication value may change during program execution for the same program - in one part of the program it may be optimal to use x2 or x4, while in some other part it may be optimal to use x64 or x128, or even x256. A proper adaptive mechanism may optimally utilize all such situations.

V. ADAPTIVE DEAD TIME PREDICTOR

In order to dynamically change the multiplication value, we may simply add a few bits to each of the banks, and use them as a saturated counter. The counter's value indicates the proper multiplication value. Every time the opened row is misclosed the counter is incremented (except its value is maximal). Every time we omitted to close the opened row the counter is decremented (except its value is 0). The only needed change in the control of this adaptive DTP would be the multiplying itself, which can be simply accomplished by a shift operation (for proper number of positions). This means that the needed changes in the hardware, required to implement the adaptive mechanism would be rather small.

Three variants of the new adaptive DTP were tried - using only 1 bit (switching only between x2 and x4), using 2 bits (changing between x2, x4, x8 or x16), and using 3 bits (changing between x2, x4, x8, ..., x128 or x256). The results are shown in Fig. 2. The expressions 'WFMV' and 'BFMV' in this figure stand for Worst and Best Fixed Multiplication Value, respectively - the worst and the best of the 8 strategies from Fig. 1. We can observe 2 things in Fig. 2. First, there are no huge differences between 2b, 2b and 3b - 1b is almost as good as 2b and 3b. Second, in practically all the cases, the 2b and 3b strategies yield identical (or in some cases slightly better) performances as (than) BFMV. This means that we accomplished the goal - the adaptive DTP yields the same (or slightly better) performances as the best possible basic DTP - x2 in the first group of programs and the "middle" strategies in the second group.
It is obvious that the amount of needed additional hardware elements is very low. However, to confirm this, as well as to evaluate the actual costs, I implemented the DTP on an FPGA chip, Xilinx SpartanII family, model xc2v500-6fg256. The old x2 version demands 17,718 equivalent gates, with 1,404 used 4-input LUTs, and the new 2b version demands 18,454 equivalent gates, with 1,484 used 4-input LUTs. We may conclude that the adaptive DTP yields nice improvements with very little additional hardware investments. Future work will include studying possibilities of introducing similar adaptive/intelligent mechanisms to the other two proposed predictors.

VI. CONCLUSION

The optimal multiplication value that should be used in the dead time predictor varies from program to program. In some programs the smallest value (2) should be used, and in some programs the used value should be much greater (8, 16 or even 32). The optimal value may also change during program execution - various parts of the program may need different multiplication values to yield minimal SDRAM latency. In this paper an adaptive mechanism was proposed. It dynamically changes the multiplication value, and as such always chooses the optimal value for each program, or each part of the program, with very little additional hardware requirements. Computer simulations show that the gained adaptive dead time predictor yields the best results compared to all the variants with fixed multiplication values.

ACKNOWLEDGEMENT

The research presented in this paper was supported in part by the Serbian Ministry of Education, Science and Technological Development [TR32012].

REFERENCES

Abstract — The paper treats problems related to crochet software development. Such applications are intended for Computer-aided design of handmade textile. Generally, they help people dealing with traditional home crafts. Moreover, these software applications can be used for storing samples of the old knitting and crochet which are ethnographic exhibits. These software systems can be developed with tools or wizards oriented to ethnographic researches. Detailed overview of graphic representation of crochet charts is done. Examples of such charts are created. According to the technology, the paper suggests an approach for crochet software development.

Keywords — Crochet software, Knitting software, Computer-aided design, CAD systems.

I. INTRODUCTION

Computer science and technologies enter in almost all branches in the modern life. They are applied even in branches where until recently the modern technologies were not in use. Such example is a handmaid textile.

Traditional home crafts as weaving, knitting, embroidery, crochet, etc. have a long history. Because they are related to the manufacture of cloths and household articles, they have practiced for centuries all over the world. Nowadays few people continue the tradition of these home crafts, although they are practiced rather as a hobby, than a necessity [3], [14]. However, it has to be mention that the interest to homemade textile provokes another tendency – putting on the market home made knitwear, embroideries, crochet as souvenirs or as customized boutique products, turning home crafts into a business [1], [2], [8].

The design process in which pre-established model of a product, is required whether for manual or machine-made product. There are many software applications for Computer-aided design of home made textiles. Most of existing hand-craft software is oriented towards embroidery and tapestry [13], [14], [15]. There are also applications for Computer-Aided Design of knitwear, so called knitting software [6], [7], [11], [12], [17]. Embroidery design software can be used as knitting software, but imposes restrictions on the type of knitted models. Also, knitting software may be used for crochet designing. Desiring to present their product as a multipurpose, companies state theirs application more than one activity [5], [10]. At all, the idea for multipurpose software for Computer-aided design of home made textile is not good. The main reason is that nevertheless the design process in embroidery, knitting and crochet are resemble each other, each home craft has its own peculiarities and different way of describing the models.

The software applications for Computer-aided design of crochet charts are much less common than those for designing embroidery and knitting. Usually, the companies dealing with knitting software state that products are usable for crochet design [5], [10]. It’s valid just only of models, whose patterns can be described in rectangular grid. The most crochet patterns are worked in the round. For this kind of models such software is inapplicable.

The examples of crochet software are Crochet Charts [4] and Filet Crochet Software [9]. First one is oriented to models crocheted in the round; second one is for fillet patterns worked flat. The advantages of Crochet Charts are: the product is tailored to the manufacture of crochet; presents the crochet diagram with crochet instructions, a well-done user interface. The product has some flaws. As the user designs the structure stitch by stitch, so the editing process is complicated. Another disadvantage is that software does not respect the rules of next crochet row coming. This makes the diagram unrealistic. Accordingly, it is possible to create crochet charts, which in practice can not be realized.

This paper represents an approach for crochet software development. The project takes into account both feature of crochet patterns and the way they have been made. The application can be used for designing new models, as well as for preserving and storing samples of old, existing crochet. In this sense, the software can be used for storage of ethnographic exhibits.

II. FEATURES OF CROCHET PATTERN DESCRIPTIONS

A. Descriptions of crochet patterns

Crochet patterns are described by crochet charts and/or crochet (work) instructions. Crochet charts include graphic symbols, each of them correspond to a crochet stitch. At all, a crochet chart follows the shape of the real crochet pattern as it’s represented in fig. 1. In most cases a legend describing stitches is added to the crochet chart. Figure 2 represents crochet chart with a description of the stitches.

Crochet symbols are type of semi-graphical characters. The most frequently used crochet symbols standardized by Craft Yarn Council are represented in fig. 3 [16]. Usually, a legend describing crochet symbols is added to crochet charts as it is shown in fig. 2.

1Elena Zaharieva-Stoyanova is with the Department of Computer Systems and Technologies at Technical University of Gabrovo, 4 H. Dimitar str., Gabrovo 5300, Bulgaria, E-mail: zaharieva@tugab.bg.
To get accurate information of pattern workmanship, the crochet charts adds work instructions. They give details how crochet each row. Practically, sometimes operating instructions are alternative manner to describe a crochet pattern. It should be mentioned though that does not always work instructions completely replace the crochet pattern. People dealing crochet often prefer to hold a crochet chart as pattern description. A crochet chart follows the shape of the pattern, so it represents how the crochet pattern looks like. Looking the chart the master can find errors leading to inability to make the model. Using charts skips also the language barrier. If the user does not speak language of written operating instructions but he is aware with crochet symbols and charts, it is sufficient to understand how realize the pattern. Work instructions in turn complement the chart and bring any necessary background information. For example, in precise work instructions no need for the user to count the chart how many crochet symbols are in a row. Overall, the full description of a crochet pattern should include both the charts and work instructions. These instructions also include materials used and the hook’s gauge.

B. Graphic representation of crochet charts

Development of computer aided design software always takes into account the peculiarities both the description of the models and technology of their implementation. In this case, it is necessary to take into account both the characteristics of the graphical representation of the charts and the way that makes the product.

As it has been already mentioned, the graphic description of crochet patterns using diagrams or charts represents their handmade. Unlike knitting patterns where diagrams are described in rectangular grid, crochet charts could be represented in various shapes. Just a few models can be represented in rectangular grid as each item corresponds to a stitch. In general, crochet symbols are put on the screen in more open order. Moreover, following the pattern shape each symbol could be rotated from 0 to 360 degrees. These features greatly make the Computer-aided design of the crochet charts more difficult than Computer-aided design of knitting patterns.

Crochet can be worked either flat or in the round. The model in fig 1 is an example of crochet worked in the round and the model in fig. 2 is an example of flat worked crochet. The second type crochet charts are resembled to the knitting charts – the stitches described by symbols are put on rectangular grid. Unlike knitting charts, because there is no precise layout of crochet stitches, the rotation of crochet symbols is possible (see fig 2). For patterns worked in the round, stitches are places in the circle and crochet symbols follow this layout (see fig. 1). It’s possible to modify the circular shape of the pattern to an oval, triangle, hexagon or any other polygon. Crochet symbols are positioned to follow the shape of the pattern.
Both of the kind of patterns (worked in flat or in the round) begins with a chain - a number of stitches named chain. Practically, the first chain stitches form the base of the crochet. Also, each row begins with one, two or more chain stitches. For patterns worked in the round, the chain stitches are closed with a slip stitch, thus the base is formed as a circle. For patterns worked in flat, the chain base does not close and the crochet continues as knitting forming odd and even rows. There are patterns worked in the round in which the base does not close in a circle. This kind of patterns acquires an oval shape.

They are common patterns that comprise a plurality of repeating segments. This technology is named “patchwork” or Irish crochet. Developing crochet software, it must be provided for editing this type charts - the location of the segments and the links between them.

### III. APPROACH TO CROCHET SOFTWARE DEVELOPMENT

In existing solutions [4], [9], crochet software development follows the approach to knitting software development – the charts are edited symbol by symbol as each symbol corresponds to a stitch. The paper suggests another approach – to represent a crochet pattern with their chart and work instructions together. The chart and its operating instructions must be interconnected, so change the chart to reflect to work instructions and vice versa – edit the working instructions to change the chart.

Initially, it’s necessary to determine the working method – crochet in flat or work in the round. The way of crochet significantly affects how will be represented graphically subsequently chart. In the first way may be used a rectangular raster grid - like knitting software development. An example of such crochet char is given in fig. 4. However, it should provide that, unlike knitting symbols in crochet charts very often violated the right rectangular order of crochet symbols. In terms of graphics representation, this means that should be allowed each of the symbols in order to be rotated around its own starting point.

In the second way, crochet symbols are arranged in the round in the chart – the shape is a circle or an ellipse. This type of the charts suggests using a polar coordinate system for their graphical representation. An example of such a chart is presented in fig. 5. Provision should be made that possible circular or oval shape can be transformed into a triangular, square, hexagonal or other polygonal shape. It is also possible circular shape be modified in more complex shape, a star, for example, as the chart in fig.2. It’s convenient, according to the type of figure, the user alone to determine the number of axes in the polar grid. Due to the specifics of crochet in this type of chart should be allowed rotation symbols around their starting points.

At all, the description of crochet chart includes its shape and the number its rows. The crochet chart comprises a plurality of rows, each of which contains a certain number of crochet symbols. Respectively can be defined the following drawing objects:

\[
\text{CHART} \rightarrow \text{ROW} \rightarrow \text{SYMBOL}
\]

These three really existing objects determine classes needed for a chart description. There are two approaches to creating a crochet pattern:

- To create a pattern by editing symbol by symbol, to shape lines that form the entire chart.
- Initially create the overall shape of the chart, separate it as rows and set the characters in each line.

Both approaches are recommended for well-developed software. It’s necessary to declare classes corresponding to the objects chart, row and symbol. Classes must have methods for graphic transformations: translation, rotation, scaling. The chart is defined by its shape – circle, ellipse or polygon. The rows follow the chart shape but each row can be somewhat of changing its size. Each row consists of crochet symbols. Each symbol has type, size, and location.

To avoid possible technical errors in the graphic construction should be laid:
- An initial row includes a set of chain stitches
- Start point for each row marked with a number and required number of chain stitches and slims, if it necessary.

Regardless of the type of chart depending on the working mode, each of crochet symbols can be rotated. It is necessary to provide for a graphical transformation rotation of each symbol.

IV. Conclusion

Despite the fact that in the modern world, unlike in the past, people are not required to make handmade products, there are some people who deal with this either as a hobby or as a small business. Moreover, traditional crafts and home occupations are also part of the cultural heritage; in practice people who deal with them keep traditions alive. Modern information technology can be successfully used for presentation and preservation of cultural heritage. Development of knitting and crochet software helps the people dealing with these home crafts. Moreover, these software applications can be used for storing samples of the old knitting and crochet which are ethnographic exhibits. These software systems can be developed with tools or wizards oriented to ethnographic researches.

In comparison with knitting and embroidery software, there are fewer crochet software applications. The reason is the larger difficulties in graphics system development due to the essence of the crochet technology. This paper treats the problems related to crochet software development. Detailed overview of graphic representation of crochet charts is done. Examples of such charts are created. According to the technology, the paper suggests an approach for crochet software development.

REFERENCES

Graph Construction Algorithm for finding the Shortest Path in a Maze

Milena Karova¹, Ivaylo Penev², Mariana Todorova³, Hristiyan Bobev⁴, Neli Kalcheva⁵

Abstract – The paper presents implementation of an algorithm for movement of a robot from a start position to a final destination in a maze. The algorithm solves two main problems: transformation of the maze into a graph and finding the shortest path from an initial to a final position. The algorithm is implemented as a part of an application, using already generated text image, obtained by a picture of the maze from top. Walls (obstacles), empty spaces, starting position of the robot and the final destination are marked on the image. The maze is presented as a bit set to form a graph which vertexes are valid positions of the robot (i.e. the robot can rotate without touching a wall). The algorithm finds the shortest path, marks movement commands and saves them into a text file.

Keywords – Bit Set, Breadth First Search, Graph, Image, Maze, Shortest Path

I. INTRODUCTION

The movement of a robot in a maze is a problem with many applications into lots of areas. Therefore the problem is an object of a broad research interest.

The main problems, which have to be solved, are presentation of the maze in a proper form and implementation of an algorithm for movement of the robot from initial position to a final destination (exit) into the maze. In the existing solutions the robot sensors or the camera of the robot are usually used [1, 3, 4]. Using these devices only a part of the maze could be shot. As a result information for the obstacles (the walls) in the maze is obtained during the robot movement. This approach is suitable, if the robot moves in an unknown or dynamically changing environment, but has some disadvantages:
- Requires permanent recalculation of the movement trajectory;
- Troubles the application of effective algorithms for path finding.

Various shortest path finding algorithms for robot movement in a maze have been researched. Some of the commonly used algorithms are: Dijkstra algorithm, A* algorithm, breadth-first search (BFS), depth-first search (DFS) and etc.

Dijkstra’s algorithm is one of the simplest algorithms.

Starting from the initial vertex where the path should start, the algorithm marks all direct neighbors of the initial vertex with the cost to get there. It then proceeds from the vertex with the lowest cost to all of its adjacent vertices and marks them with the cost to get to them via itself if this cost is lower. Once all neighbors of a vertex are checked, the algorithm proceeds to the vertex with the next lowest cost [2, 5].

A* is like Dijkstra’s algorithm in that it can be used to find a shortest path. A* algorithm is the most popular choice for path finding, because it’s fairly flexible and can be used in a wide range of contexts. It is one of a family of graph search algorithms that follow the same structure. These algorithms represent the map as a graph and then find a path in that graph. Depending on the environment, A* algorithm might accomplish search much faster than Dijkstra’s algorithm [5].

Graph traversing in depth (Breadth-First-Search or BFS) is algorithm for path searching in trees or graphs [6]. Searching starts from a given vertex in the graph (or from the root in case of a tree) and all the unvisited neighbors are visited first. The algorithm is proposed by E. F. Moore, who used it for finding the shortest path in a maze.

The algorithm, proposed in this paper, is called AlgoHris. It solves two main problems:
- Construction of a graph using the image of the maze, shot from above (for example by the camera of a drone);
- Finding the shortest path from an initial position to a final destination in the graph.

The AlgoHris algorithm is compared to three other algorithms: Backtracking, A* and Genetic Algorithm GAPP.

II. ALGORITHM ALGOHris FORMULATION

The AlgoHris algorithm considers an image of the maze. The image is shot from above, for example by the camera of a drone. Thus AlgoHris is integrated into an application for shooting a maze from above, finding shortest path in the maze and moving a robot from an initial position to a final destination in the maze (Fig. 1).

Fig. 1. Schema of the whole project
The proposed algorithm consists of two parts:
- Construction of a graph, presenting the maze;
- Finding the shortest path from an initial position to a final destination, using the graph.

A. Graph construction age

The construction of a graph is an essential problem for the robot movement. This problem is hard due to the limited resources of the robot platform (computing power, memory).

The proposed AlgoHris algorithm uses presentation of the maze as a set of characters 0 and 1, where 0 marks free (possible) position for the robot movement and 1 marks busy position (i.e. an obstacle or a wall). Fig. 2 presents a simplified maze and a graph with an initial position and a final destination.

![Fig. 2. Construction of a graph for an example maze](image)

The algorithm constructs unweighted, indirect graph, formed by all pixels, which could be a valid position of the robot in the maze. A position is valid, if all pixels up, down, left and right in a distance equal to the robot size are free. Such construction of the graph guarantees, that only the possible routes are presented in the graph. Over this graph various algorithms for path finding could be applied.

The graph construction algorithm uses radius of the circle, described around the robot (Fig. 3). This presentation is used for checking the valid positions of the robot in horizontal and vertical direction.

![Fig. 3. Robot presentation, using described circle](image)

B. Check for valid position in horizontal direction

The valid positions in horizontal direction are checked according to the radius of the described circle (i.e. according to the robot size) (Fig. 4).

![Fig. 4. Valid positions in horizontal direction](image)

If there are no valid positions in horizontal direction, no vertex is added to the graph for these positions (Fig. 5).

![Fig. 5. No valid positions in horizontal direction](image)

C. Check for valid position in vertical direction

The valid positions of the robot in vertical direction are formed by OR operation of each pair of rows from the set, presenting the maze (Fig. 6).

![Fig. 6. Check for valid positions in vertical direction](image)
If the OR operation of a pair of neighbor rows results in a character 1, then an obstacle is available on this position and the robot could not be in the position. No vertex is added to the graph for the position (Fig. 7).

Fig. 7. An obstacle in vertical direction

If the OR operation of a pair of neighbor rows results in characters 0 only, then the positions in vertical direction are free. A vertex is added to the graph (Fig. 8).

Fig. 8. Example for valid positions in vertical direction

D. Finding the shortest path

If the described graph, containing only the possible positions of movement, is available, an algorithm for finding the shortest path could be applied over the graph.

The algorithm, proposed in this paper, implements breadth first search. The algorithm finds the shortest path between an initial and a final vertex, passing through least number of visited vertices.

The algorithm uses the following data structures:
- Queue of the graph vertices;
- List of the visited vertices;
- List of the possible paths.

First the initial vertex is marked as visited and is added to the queue. Afterwards the neighbor vertices are traversed. If a neighbor vertex is not visited, in the field “previous” of the vertex the number of the previous neighbor is written. The vertex is marked as visited and is added to the queue. The loop continues until the queue is not empty and the final vertex is not visited.

Fig. 9 presents the algorithm execution for an example graph.

E. Programming implementation

The most challenging problem in the implementation of the described algorithm is proper presentation of mazes with large dimensions (e.g. 1600x1013 pixels). Such images have to be quickly transformed to a graph, considering the limited memory and computing power in the selected robot platform.

In the implementation of the AlgoHris algorithm the maze is presented as a set of bits. Each bit is a structure of elements with only 2 possible values: 0 (true) or 1 (false). A programming class is realized, emulating an array of Boolean elements. Each element reserves only one bit, which is eight times less than the simple character data type. Each bit could be accessed individually as in ordinary array.

III. Experimental results

The experiments are carried out in two directions:
- Comparing the times for graph construction for mazes with various dimensions;
- Comparing the algorithm with three other algorithms, using the same dimensions (height and width in pixels) of the maze – Backtracking, A* and genetic algorithm GAPP.
In the first experiment the times for completing the separate steps of the whole algorithm are presented in Table I.

**Table I**

<table>
<thead>
<tr>
<th>Width of the maze (pixels)</th>
<th>Height of the maze (pixels)</th>
<th>Time for graph construction (ms)</th>
<th>Time for shortest path finding (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>480</td>
<td>304</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>480</td>
<td>304</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>480</td>
<td>304</td>
<td>46</td>
<td>6</td>
</tr>
<tr>
<td>1000</td>
<td>820</td>
<td>72</td>
<td>23</td>
</tr>
<tr>
<td>1000</td>
<td>820</td>
<td>73</td>
<td>24</td>
</tr>
<tr>
<td>1600</td>
<td>1013</td>
<td>105</td>
<td>52</td>
</tr>
</tbody>
</table>

The results show, that the implemented algorithm processes relatively large image (1600x1013 pixels) and finds the shortest path for less than 160 ms.

The data, obtained by the second experiment, are presented on Fig. 10. In the case of small dimensions of the maze there is no significant difference between the times for shortest path finding. The genetic algorithm is comparatively slow and inapplicable to large dimensions of the maze. AlgoHris achieves shorter time than the Backtracking and even than the A* algorithm when increasing the dimensions of the maze.

![Fig. 10. The shortest path finding time for four different algorithms](image)

On Fig. 11 the approximation function for AlgoHris is shown. The function is

\[ y = 23x - 18.667, \quad R^2 = 0.9845 \]

The dependence between the time for path finding and increasing the maze dimensions is proportional and the algorithm is stable.

**Fig. 11. Trendline correlation for AlgoHris**

**IV. CONCLUSION**

The proposed AlgoHris algorithm for movement of a robot in a maze has the following advantages in comparison to the existing implementations of other algorithms:

- It uses an image of the maze, which makes possible fast processing of the pixels in the image to find a path in the maze;

- The maze is presented as a bit set, which significantly increases the graph construction, containing only the possible paths for movement of the robot.

Although in the current implementation AlgoHris uses the breadth first search method to find the shortest path, other methods for shortest path finding as Dijkstra’s algorithm and A* algorithm could be effectively applied over the graph, produced by AlgoHris.

The future work will consider implementation of algorithms for finding paths in mazes, which are not shot in advance. The camera or the sensors of the robot will be used. This way the robot will be able to move in dynamically changing environment.

**REFERENCES**


