UDC 004.05

Design and implementation a software for water purification with using automata approach and specification based analysis

Sergey Staroletov (Polzunov Altai State Technical University)

The paper covers design and developing software for hardware plant for water purification, the architecture for it, received automaton diagrams of water preparing and normalization based on customer specifications and requirements. Discussing the components of the system, layers of abstractions, verification points, issues to build it. The way of developing well-qualified suchlike systems based on specifications is given.

Keywords: Water purification, software, automaton, energy conservation, verification, requirements engineering

1. Purpose and novelty of the project

Our university received an order for the research work to design and development software under the UN grant for developing countries in the field of energy conservation.

A customer is developing the hardware stand providing preparation of distilled water of the given temperature and testing the energy consumption and water consumption of various connected devices (for example, washing machines and dishwashers).

Water purification is necessary for this project because existing standards [1] for measuring energy consumption and water consumption presuppose to work with distilled water (water preparation process) at a given temperature (water normalization process).

It was necessary to design and implement software that automatically control preparing large volumes of purified water, also measures the energy and water consumption of the connected devices and generates some reports.

The novelty of the project is following: all water preparation and equipment testing should be performed without user intervention by the automatic operations in the hardware stand and the operator's job is only to select the mode, to set parameters and then run the process.

From the programmatic point of view, novelty consists primarily in mandatory to implement algorithms for water purification and control of hardware devices to it, these algorithms must be primarily reliable because water is supplied under high pressure, is heated using amperage

of tens of amperes, and all possible exceptional operations must be processed.

These algorithms must include various multithreaded interactions because the states of the devices are not correlating to each other, and it is necessary to examine at any time the devices, update the interface, make decisions about the further operational logic.

Therefore, it was necessary to design the software architecture properly and verify the proposed algorithms before using them.

This paper covers the software architecture design and algorithms developed by the author, also verification and testing, main issues and the questions about formulation the requirements for building these systems.

The results partially were obtained within the RFBR grant (project No. 17-07-01600).

2. A bit about the hardware

In this article, there is no purpose to describe the hardware. The author did not design and develop it but implemented software for the available hardware. Here will be given information about the components to get an idea of the operation of the equipment as a whole.

Here is a water purification hardware stand (Figure 1):

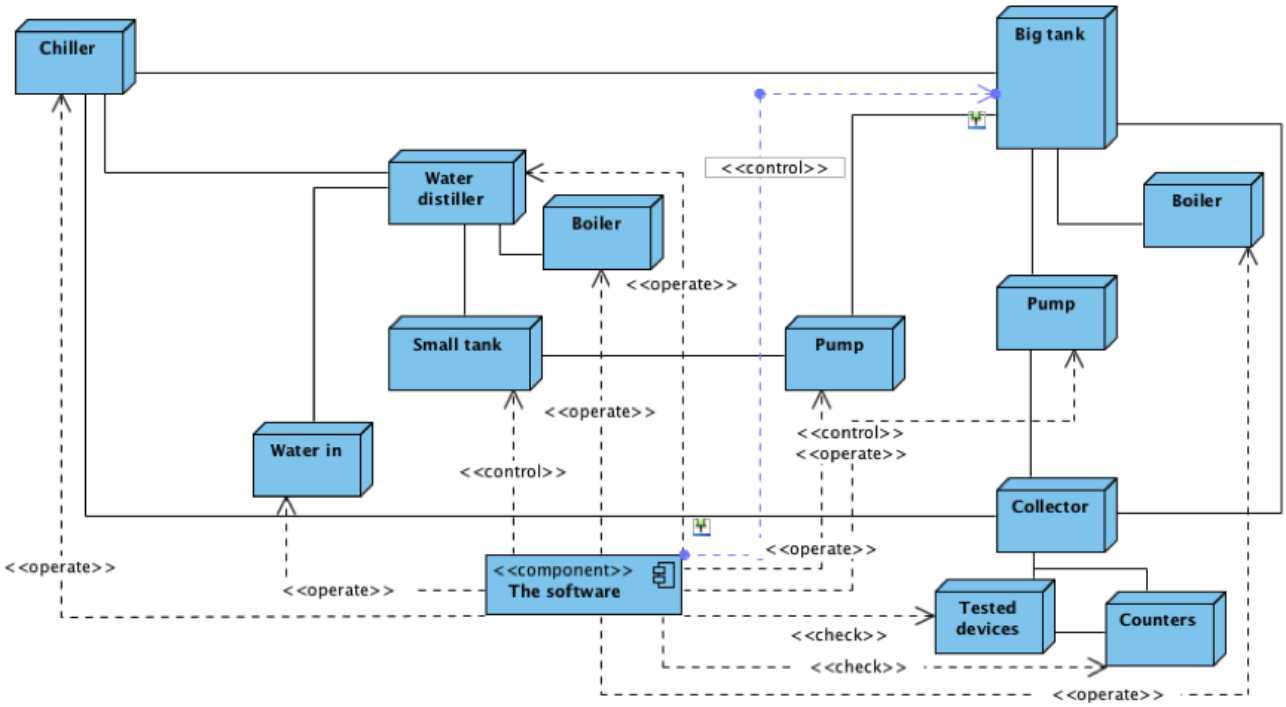

Fig. 1. Diagram of hardware components using in the stand

1. A chiller (conditional element) for cooling the water. Chiller is operated by a special software to set the output temperature based on current air temperature and desired temperature.

2. An aqua-distiller (with heating element) to create a distillate from given water.

3. A small tank for obtaining the current distillate with a level sensor.

4. Large storage tank for obtaining final water with level sensors and a heating element for heating.

5. A small pump for pumping water from a small tank into a large storage tank.

6. Main pump for water circulation in the whole system (a variable frequency drive).

7. Digital equipment for measuring water consumption (water flow meters) and electricity meters.

8. Digital devices with digital and analog I/O ports to connect devices, start relays and sensors to serial port adapters.

9. A collector, through which the devices checked for energy saving are connected and through which the prepared water circulates from the tank.

10. Various valves for switching water flows between the aqua-distiller, the storage tank, the collector.

The interaction with the hardware is implemented on the basis of the Modbus [2] protocol by using digital multi I/O devices, on/off switching relays, obtaining status data via analog and digital inputs. Linear interpolation algorithms are used to translate analog sensor values into the digital form.

The special software component called Devices Map has been implemented to link the devices from devices list (Figure 1) into concrete I/O ports on concrete digital I/O modular devices connected to concrete serial ports on the computer.

3. System architecture

The designed application inside consists of:

- **A device(hardware) layer** - a single software interface for the devices is created and implementation classes for each device in the system are created. This layer is built on the top of the Modbus protocol (and GOST IEC protocol [3] for the electric meters) and own actions for every device (to read some values from device's registers).
- **Serial port layer**. Every device is known by its port (Device Map tool causes a hash table device - port) and for the correct work, it must reserve the port because at any time only one device can operate on the port. Each port layer abstraction should work in **Device Getter Thread**, and to acquire the port to transmit data we must wait for

any other device freeing the port. It could be implemented as a synchronization primitive (mutex with lock/unlock).

- **A layer of objects**. Every device state is represented as an object. If we want to acquire a device state we can get the state in every moment by accessing a getter method in a corresponding object which stores the last received state from a device. Object stores current data and periodically **Device Getter Thread** updates it.
- **Updater Thread** is a foreground process that periodically asks all the objects and gets their state, then updates corresponding UI elements.
- **UI** is a mnemonic representation of the system; it has background images and dynamic elements (text values of system parameters such as temperature, pressure, on/off boxes to show states, graphic objects to draw some primitives like level meter, animations to show chiller's fans) which are updated by Updater Thread. UI also can work in manual operator's mode when operator press to the devices on the mnemonic diagram to start and stop them.

An example of components inter-operation sequence is shown on Figure 2.

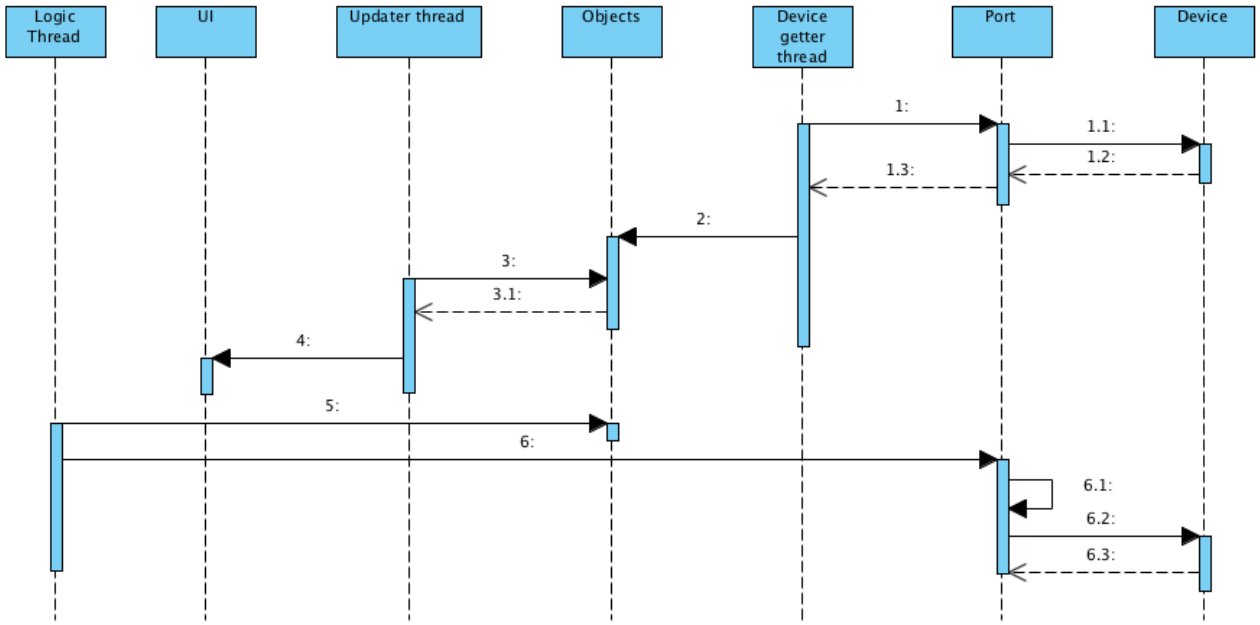

Fig. 2. Sequence diagram of possible components inter-operation

Here, **Device Getter Thread** starts to update the device's state. It requests a **Port** for given device, possible wait for the port, then when the port is free it runs device state update routine (which constructs Modbus or different protocol message corresponding to the device number and device updating algorithm), then gets the state and updates the **Object** to its

device. Then **Updater Thread** gets the current state of the device object and updates the **UI** corresponding to the new device state value.

Described in the next sections **Logic Thread** can get the object state and make decisions based on it. During the logic cycle, it can turn on/off a special device (for example, starts/stops the main pump). It asks port when it becomes free for the corresponding device, then acquires it and executes a routine in the **Device layer**.

Implementing these ideas is not very difficult to a developer who knows how to operate with the threads and how to lock resources from data races with using simple locking. Implementing the device layer requires reading its specifications, understanding the protocols by using original software and looking to transmitting and receiving data. The correctness of it is checking by testing techniques.

4. Requirements for development

Requirements engineering [4] is a way to collect customer's requirements for the system. Our goal now to help software engineers work together with the customers to build adequate error free software because the definition of the word "error" is nonconformity of program's behaviour to the original specification of requirements. Why not use this project to analyse such documents like "Requirements for development" to extract the correct specifications from it and integrate the requirements engineering, developing and verification processes to minimize troubles of producing such systems?

About the water purifying software. The software engineer can create the software to start/stop the devices and watch the states based on devices specifications but cannot create the whole inter-operation system because he usually doesn't know anything about purifying the water of even what is the chiller or aqua-distiller, he needs a customer and his needs. In this project, firstly, the task of implementation of given system architecture has been done, so that the operator could run the whole process of water purifying manually by pressing keys in the application and visually monitor the result on the screen and a real installation. Further, as a result of negotiations, the algorithm of automatic actions of the program was approved in the form of the following document (some excerpts are given in Appendix A).

Here we see the description of the algorithm in some high-level form. The algorithm is based on the management of devices that were specified in the picture of the interface (like in Figure 1), and the functional is predetermined by water purifying specialist.

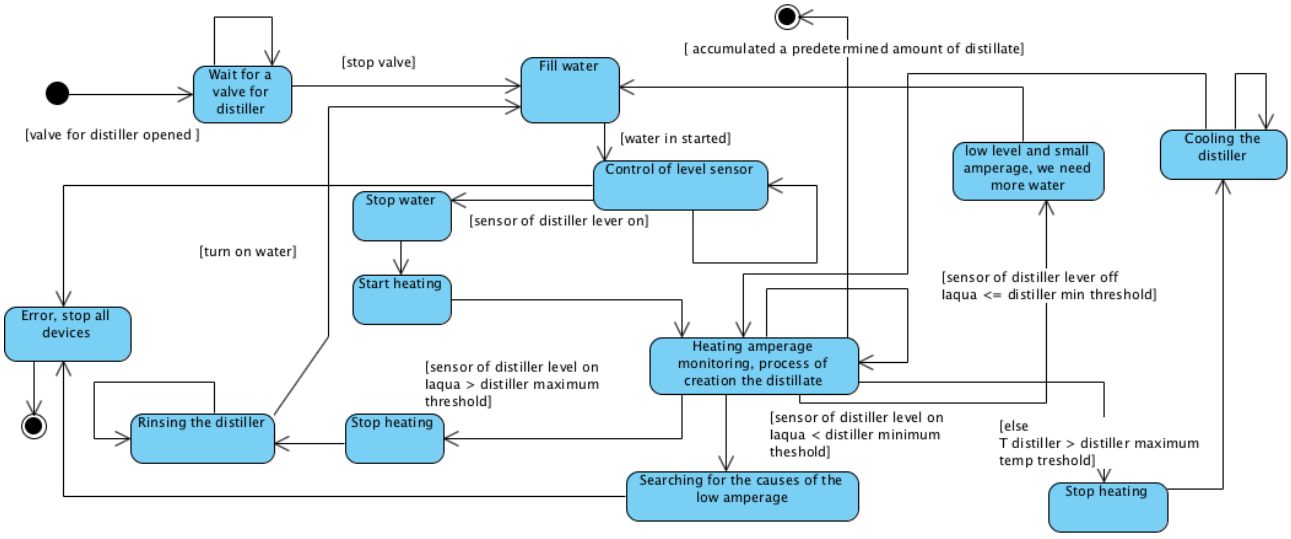

Fig. 3. State machine for water preparation

Next, based on given specification, it was decided to build the program in the form of a finite automaton, because it is clear how to implement a state-based algorithm as an automaton: each action corresponds to one or more states, and the actions of the automaton also correspond to actions as a result. Principal diagrams of logic are given below (Figure 3 and 4). The final step of programming is to implement the automatic purification algorithms. It is now not a difficult thing because the software architecture is designed than developed, device layer is developed and tested, and we have state machine diagrams as a model. Developing the algorithms for testing connected devices for energy and water consumption is not covered here because of simplicity (running water circulation between collector and storage tank; counting the meters, and plot the graphs).

5. Verification of the automata model

During this process we had to verify some verification points:

1. Actions to start/stop devices do not affect the current process of obtaining information.
2. Check pumping: if we got some level of water in the small tank all the water must be pumped to the storage tank.
3. Distillate preparing cycle will be completed, or an error message will be displayed.
4. Distillate normalization cycle will be completed, or an error message will be issued.
5. Will not fill water above the edge of distiller / tanks.
6. Pumps do not work without water.

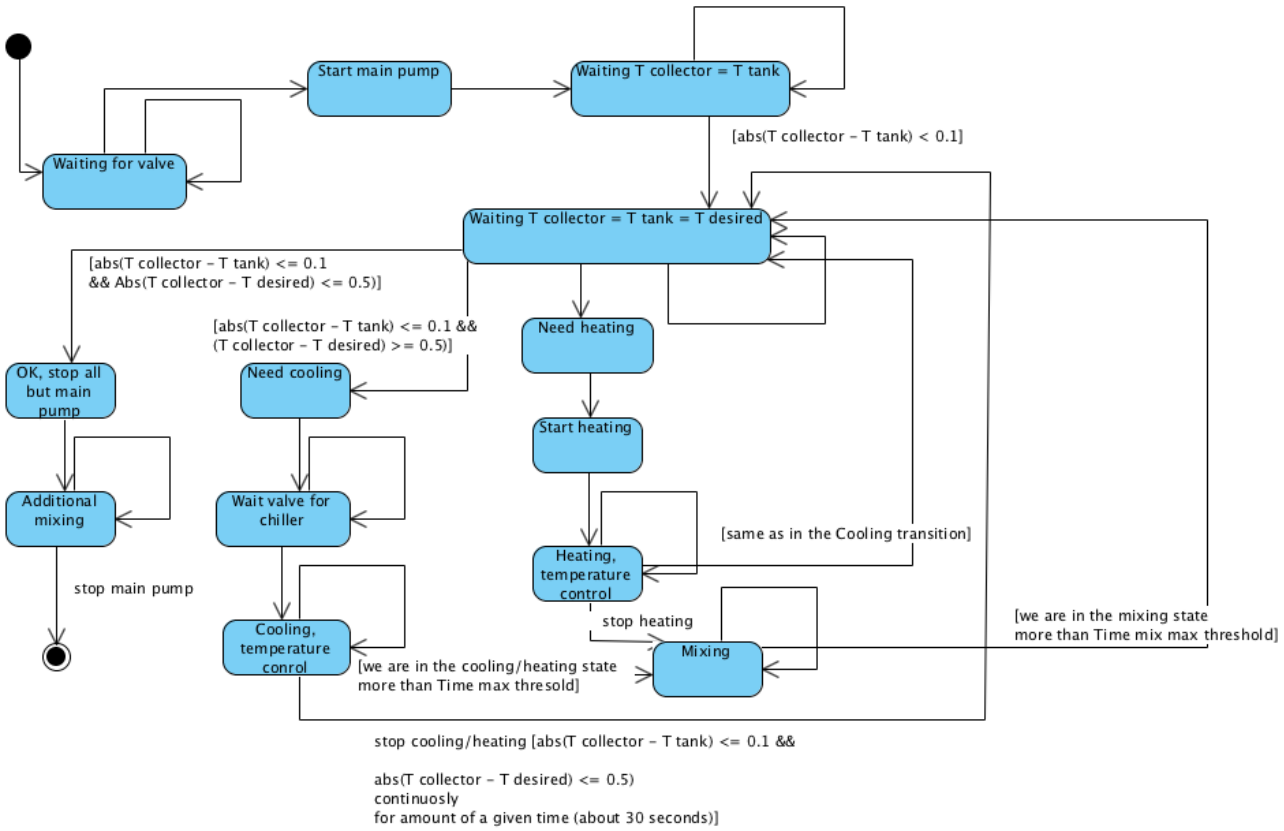

Fig. 4. State machine for water normalisation

7. You cannot start a heater / distiller if the level of water in the tank is small.

8. In each mode, water flows are redirected correctly.

9. Some devices can be turned on/off after some time after the control impact and the system must work correctly.

These points were given by the customer after implementing the system architecture and running it in manual mode by the operator and doing some experiments. Also, some of these checks were implemented in hardware PID control [5] methods.

To verify these key points with Model Checking approach, we need a checker which supports multiple processes and has abilities to implement a model of a finite automaton. Almost all checkers offer it. Moreover, also we need some predicates based on sensors values, automata states and system's states expressed in Linear-Time Logic because of delayed time in our verification points. We use Spin Model Checker [6] because of simplicity to achieve the desired result.

For example if we need to model the state changing process in the water normalization process and emulate the temperature mixing, we could write the following code in Promela

based on states from Figure 4, mtype state definition, loop and switch of possible states:

```
mtype = {StartMainPump,
WaitingTcollectorTtank,
WaitingTcollectorTtankTdesired,...};

mtype state = StartMainPump;

byte Tcollector = 10;
byte Ttank = 20;
bool mainPumpStarted = false;

active proctype WaterNormalProc() {
do
:: {
if
::(state == StartMainPump) -> {
mainPumpStarted = true;
state = WaitingTcollectorTtank;
}
::(state == WaitingTcollectorTtank) -> {
printf("WaitingTcollectorTtank");
if :: (Tcollector == Ttank) ->
{
state = WaitingTcollectorTtankTdesired;
};
:: (Tcollector != Ttank) ->
{
if ::mainPumpStarted -> Tcollector = Tcollector + 1; fi
}
:: (Tcollector != Ttank) ->
{
if ::mainPumpStarted -> Tcollector = Tcollector - 1; fi
```

```
}
:: (Tcollector != Ttank) ->
{
if ::mainPumpStarted ->Ttank = Ttank + 1; fi
}
:: (Tcollector != Ttank) ->
{
if ::mainPumpStarted -> Ttank = Ttank - 1; fi
}
fi
}
::state = WaitingTcollectorTtankTdesired -> {
...
}
fi
}
od
}
```

Here we use the non-determinism in Promela to achieve the random temperature mixing.

During the using the Model Checking approach we found some lacks in the transitions in the specification, it did not cover all the possible transitions to the final states.

6. Analysis, results and issues

As a result, the software has been developed successfully. We use the methods here ("→" means the sequence in the design and developing process): design of architecture → device level implementation and testing → overall internal architecture implementation → testing → specification of control algorithms → automata model → implementation of water purifying by given automata model → verification → overall system testing. It seems that the automata approach has been chosen extremely right: the given specification could be easily translated to automatons and each automaton could easily run from the thread in an infinite loop. Also, the proper architecture planning before construction of the logic, plays a huge role. Verification can help to refinement the specification and set-up critical hardware checks.

The main issues during project implementation were:

1. We cannot simulate the device layer, so we need to test on real devices.

2. The specification of control algorithms was given from the customer only after the overall internal architecture implementation because he is not sure about the functionality of the hardware so it was difficult to estimate the effort and the software cost before starting the project.

3. The specification of control algorithms was changed several times after testing the functionality because of lacks in the specification.

The results of the current project show us the importance of the specification for the control systems and necessity to develop a special approach to design such systems. A BDD (Behaviour-driven developing) [7] is only one approach in software engineering now which is being used to develop and test the software based on a specification in natural form. It is proposed to extend the BDD language Gherkin to describe steps in the specifications in natural language and to describe the requirements as verification points. It will allow a software engineer to start work with the specification, implement the tests based on it, then implement the software based on steps and verify this software by generating necessary automatons and set-up the requirements. It is a subject of further research.

References

1. IEC 60456:2010. Clothes washing machines for household use - Methods for measuring the performance. https://webstore.iec.ch/publication/2188
2. Modbus protocol. http://www.modbus.org/specs.php
3. GOST IEC 6107-2011.Data exchange while reading meter values, tariffing and load management.
4. D. Alrajeh, J. Kramer, A. Russo, S. Uchitel. Elaborating Requirements using Model Checking and Inductive Learning. IEEE Transactions on Software Engineering (Volume: 39, Issue: 3, March 2013). pp. 361-383
5. Ang, K.H., Chong, G.C.Y., and Li, Y. (2005). PID control system analysis, design, and technology" (PDF). IEEE Trans Control Systems Tech, 13(4), pp.559-576.
6. Spin – Formal Verification Tool. http://spinroot.com
7. M.Wynne, A.Hellesoy, S.Tooke. The Cucumber Book, Second Edition. Behaviour-Driven Development for Testers and Developers. The Pragmatic Bookshelf, 2017, 336p. ISBN: 978-1-68050-238-1

A. A fragment of customer's specification of water normalisation process

```
Algorithm: Distillate cycle.
(Active before filling the large / storage tank)

Step 0: Start preparation ...
Step 1: The water pressure is checked at the input
(check sensor No ...) ...

Step 2: Filling the distiller
The level sensor in the distiller is monitored for
(time of opening the valve) after opening valve N...
Yes / parameters OK:
1) Valve ... is closed;
2) Transition to step 3.
No / the parameters are not normal:
1) Message (Filling of filling of the distiller);
2) The automatic mode is switched off.

Step 3: Distillation
1) The heating of the distiller is activated (Button ...);
2) Parameters are monitored:
- amperage;
- temperatures of the distiller;
- filling sensor;
- the sensor for filling the storage tank.
Yes / parameters OK:
1) Accumulation distillate in the tank
2) Executing step 3 again
No / the parameters are not normal:
- The amperage exceeds the set maximum:
```

```
1) The heating of the distiller is switched off;
2) The transition to 3.1.
- The amperage with the fill sensor turned on is below
the set minimum:
1) The heating of the distiller is switched off;
2) The transition to 3.2.
- The amperage with the filling sensor switched off is below the
set minimum:
Go to step 1.
- The temperature is higher than the given maximum:
1) The heating of the distiller is switched off;
2) The transition to 3.3.

Step 3.1: Draining of salted water ...
```