



DICES technical report

iForestFire

Project name: DICES
Contract number: No. 03/07
Authors: Darko Stipaničev, Maja Štula, Josip Maras

Executive summary

iForestFire is an intelligent and integrated video based monitoring system for early detection of forest fires. Forest fires are detected in incipient stage using advanced image processing and image analysis methods.

DICES has a goal to advance development of distributed embedded software systems with emphasis on software reusability and predictability of software quality . One of DICES's case studies is the iForestFire system. In order to be able to re-engineer the iForestFire system the current state of the system must be analyzed.

iForestFire is a **Web Information System (WIS)** composed of the following parts:

- Multi-agent application – used for data collection and as a middleware between the other parts of the system.
- Fire detection application – detects fire by image analysis.
- Web user interface.

The following document describes the results of reverse engineering process applied to iForestFire system. It also gives an overview of an Eclipse plugin “phpModler“ which was developed to support the reverse engineering process of iForestFire's web user interface.

Table of Contents

1 Introduction	4
1.1 Relationship between iForestFire and DICES	4
2 iForestFire	5
2.1 iForestFire system structure	5
2.2 iForestFire components	7
3 Multi-agent application	8
4 Web interface	12
4.1 phpModeler	13
5 Fire detection application	19

1 Introduction

iForestFire is an intelligent and integrated video based monitoring system for early detection of forest fires. Forest fires are detected in incipient stage using advanced image processing and image analysis methods.

The purpose of this technical report is to:

- present results of the reverse engineering process of iForestFire system.
- give an overview of an application developed to facilitate static analysis of web application – phpModeler.

1.1 Relationship between iForestFire and DICES

DICES addresses efficient reusability of software components and prediction of the important properties for embedded systems: resource utilisation, and performance, by applying the service-oriented software engineering and component-based software engineering methods and technologies. The project applies and develops new theories for predictability of certain quality attributes providing a) improved and more efficient software development b) optimal solutions of software architecture and components configurations for distributed systems.

The theories will be validated on a case – “iForestFire - Intelligent Forest Fire System” developed at FESB Split. This will enable a thorough validation of the approach and provide input for further development of this system and possible commercialization of the improved product.

2 iForestFire

iForestFire is an intelligent and integrated video based monitoring system for early detection of forest fires. Forest fires are detected in incipient stage using advanced image processing and image analyses methods. Intelligent fire recognition algorithms analyze images automatically, trying to find visual signs of forest fire, particularly forest fire smoke during the day and forest fire flames during the night. If something suspicious is found, a prealarm is generated and the appropriate image parts are visibly marked. The operator inspects suspicious image parts and decides is it really a forest fire or not.

iForestFire is a **Web Information System (WIS)** and the only user interface is accessed through a standard Web browser. The system is based on field units and a central processing unit. The field unit includes a day & night, pan/tilt/zoom controlled IP based video camera and an IP based mini meteorological station connected by wired or wireless LAN to a central processing unit where all analysis, calculation, presentation, image and data archiving is done.

2.1 iForestFire system structure

iForestFire is composed of the following hardware components:

- pan/tilt/zoom (PTZ) video camera
- video web server
- central processing server
- workstation
- surveillance camera
- data web server
- meteorological station

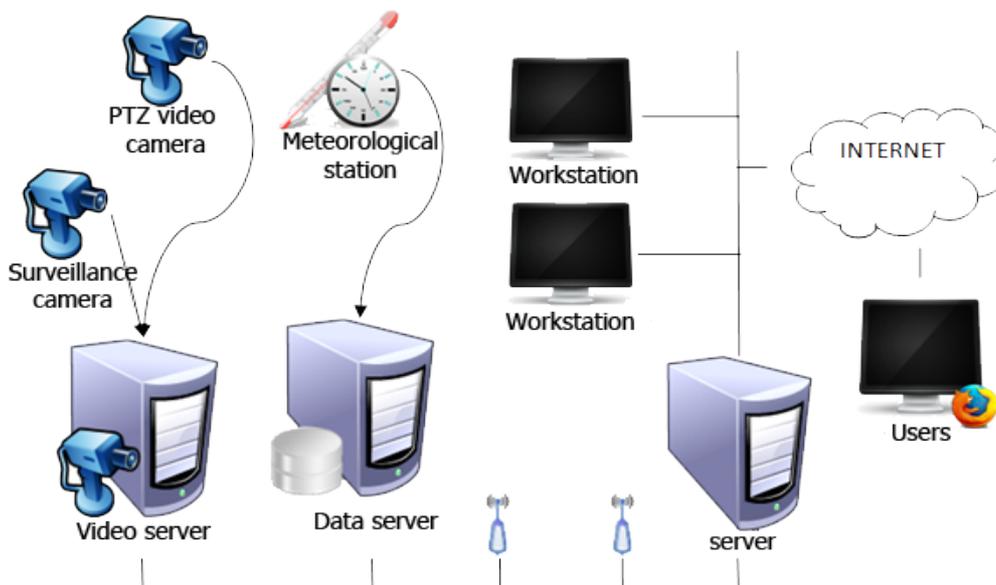


Illustration 1: iForestFire system structure

iForestFire is composed of the following subsystems:

- Subsystem for displaying video recordings and camera control.
- Subsystem for early detection of forest fires.
- Subsystem for search and storage of video recordings.
- Subsystem for search and storage of meteorological data.
- User interface.

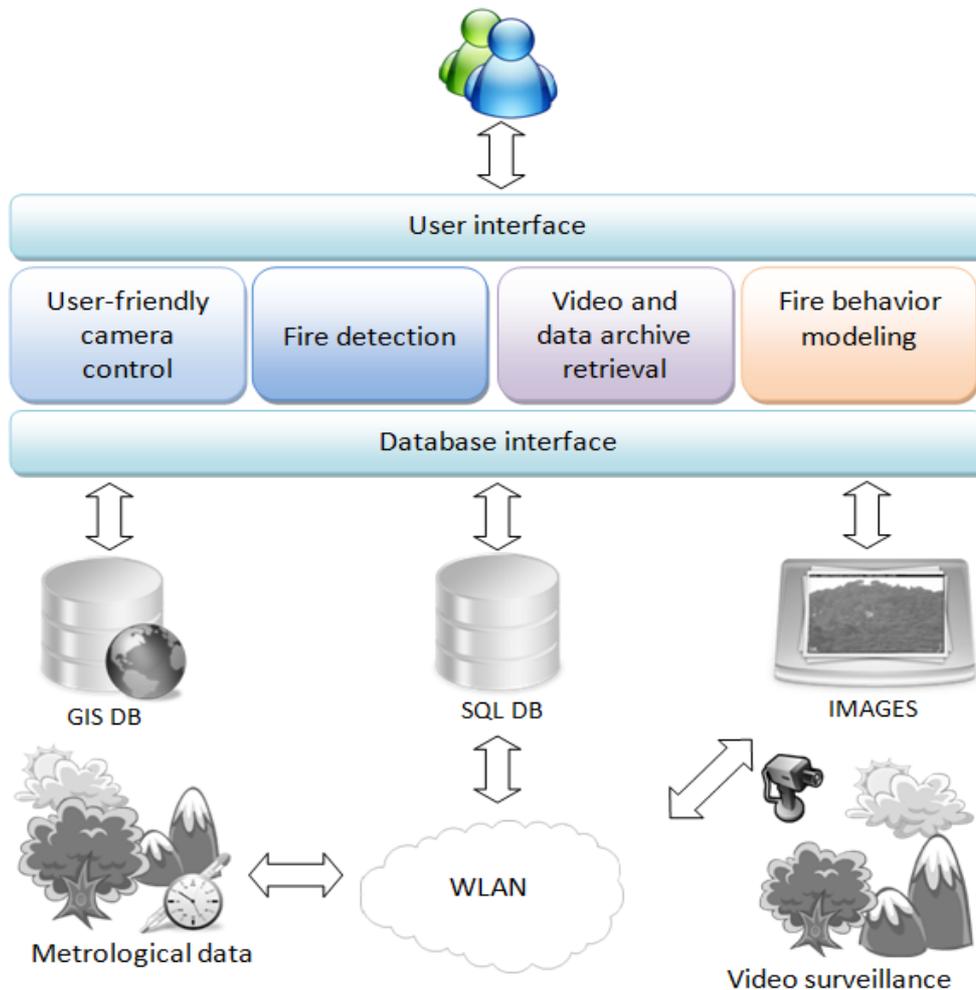


Illustration 2: iForestFire subsystems

2.2 iForestFire components

iForestFire is a three part system based on the web type client-server architecture. It is composed of the following parts:

- Multi-agent application – used for data collection and as a middleware between the other parts of the system. It was developed in JAVA using JADE (Java Agent Development Framework) and JESS libraries (rule engine for Java platform).
- Fire detection application – detects fire by image analysis. It was developed in C.
- Web user interface – developed using following technologies: PHP, HTML, JavaScript, Ajax, CSS...

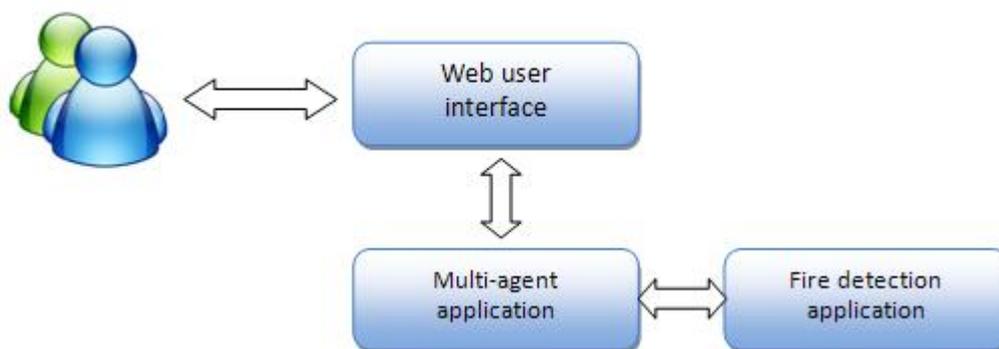


Illustration 3: iForestFire main components

3 Multi-agent application

Multi-agent application is a JAVA application used for collecting data from the outside world. It is also acting as a middleware between other parts of the system. Application's functionalities are implemented with the following agents:

- DataBaseAgent, implementing the BazaCyclicBehavior, is an agent that starts with the application. Depending on the current system configuration (number and type of cameras, meteorological devices...) it instantiates agents necessary for data retrieval and system management.
- CameraAgent, implementing the CameraCyclicBehavior, is in charge of cameras. It creates PresetAgent and PresetAgentNoPTZ agents.
- ACLMediatorAgent, implementing FifoMediatorBehaviour.
- MeteoAgent, implementing MeteoTiniBehaviour or MeteoAxisBehaviour, is in charge of data retrieval from meteorological devices.
- NodeObserverAgent, implementing MeteoObserverBehaviour, is in charge of MeteoAgents.
- PresetAgent, implementing PresetBehaviour, is an agent who's main function is to fetch images from the designated camera preset position and store them in the specified folder.
- PresetAgentNoPTZ, implementing PresetAgentNoPTZBehaviour.
- VirtualCollectorAgent, implementing BazaSunBehaviour.

Due to different technologies used in the development of iForestFire system, it was necessary to pay special attention to means of communication between different components. iForestFire uses the following communication mechanisms:

- PostgreSQL database
- Files
- Named pipes
- Unix signals

In the process of reverse engineering, for each agent the following diagrams were created:

- Agent class diagrams:

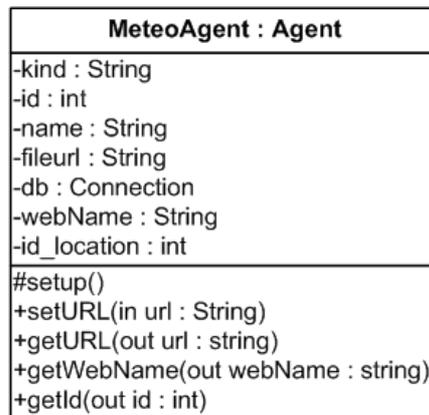


Illustration 4: MeteoAgent class diagram

- Diagrams showing communication with database tables:

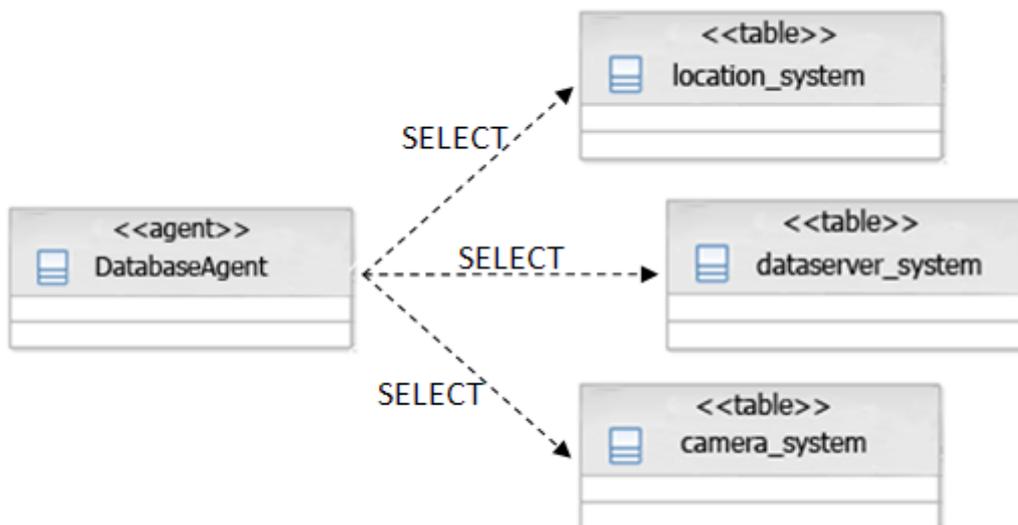


Illustration 5: DatabaseAgent communication with database tables

- Diagrams showing communication with other agents:

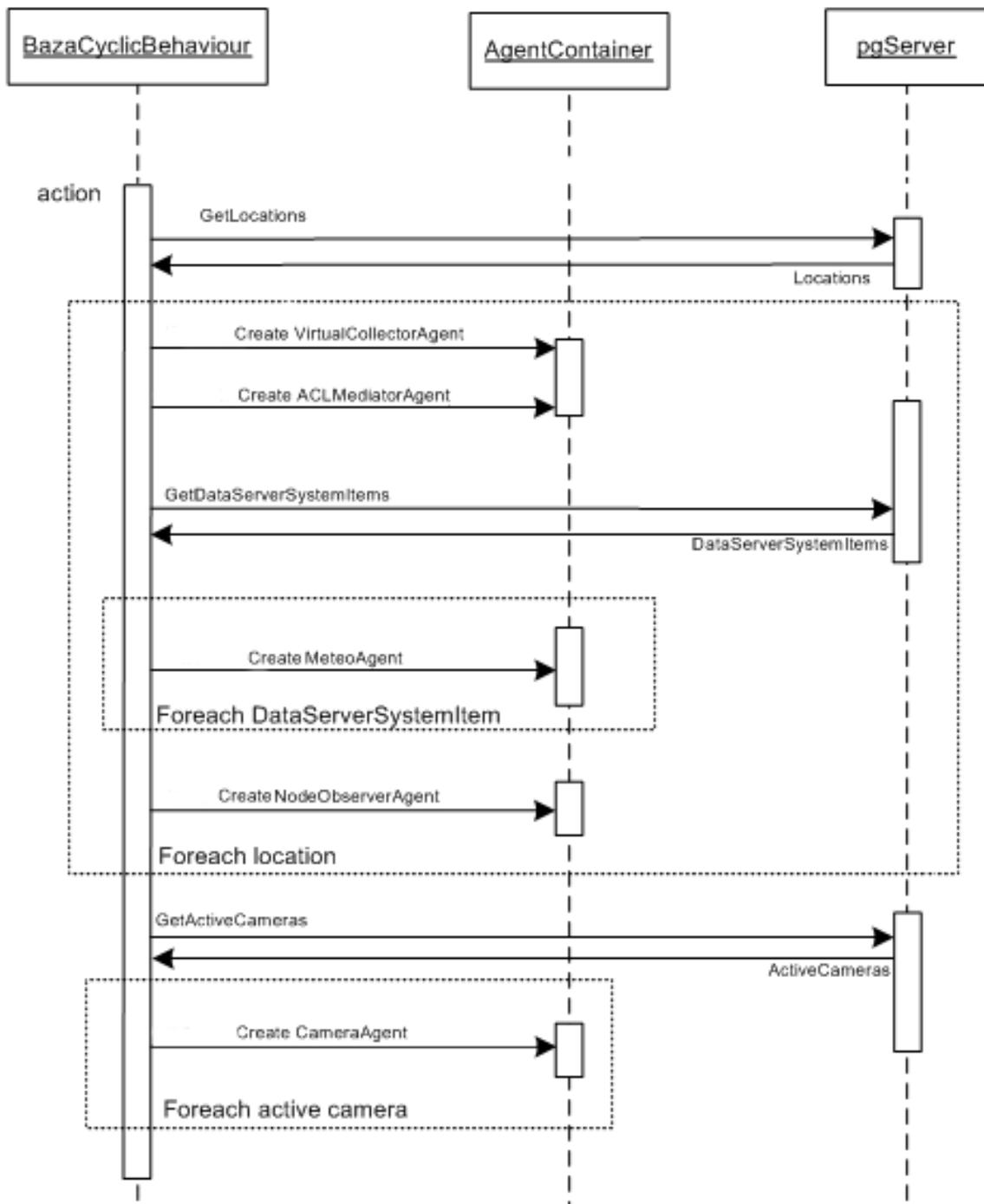


Illustration 6: DatabaseAgent agent communication

- Diagrams showing file access:

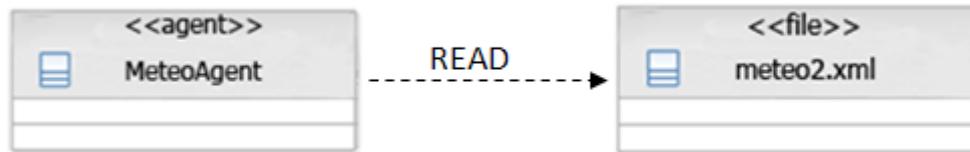


Illustration 7: MeteoAgent file access

- Diagrams showing communication using signals:



Illustration 8: PresetAgent signal communication

4 Web interface

iForestFire is a web information system that uses web as a user interface that enables data retrieval for users or other systems.

Client pages are developed using technologies such as HTML and JavaScript and are dynamically generated from server pages developed using PHP. The server part of iForestFire web interface is communication with other components of iForestFire via the PostgreSQL database.

Main web user interface functionalities are:

- Displaying current and archived video recordings.
- Informing operators about detected alarms and displaying archived alarms.
- Remote video camera control.
- Displaying current and archived meteorological data.

iForestFire web user interface is composed of about three hundred scripts and pages.

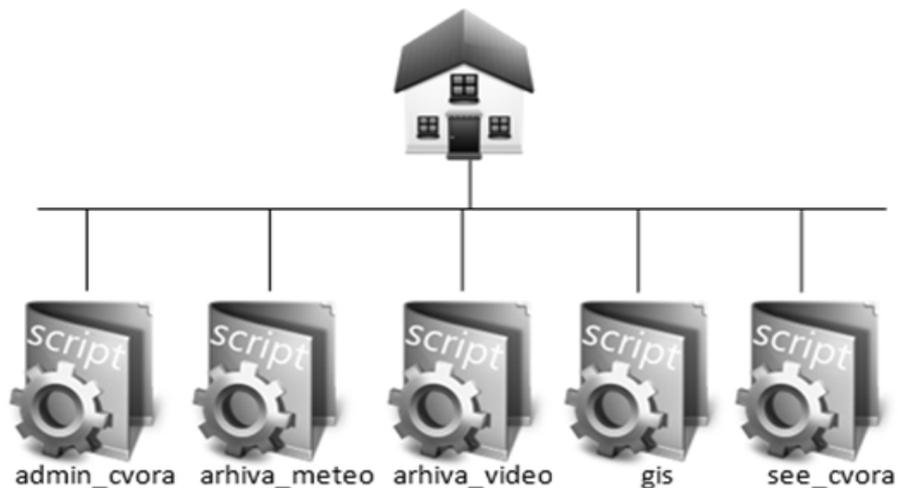


Illustration 9: Web interface data structure

In order to ease the maintenance process it was determined that for each script the following diagrams should be created:

- Diagrams showing which resources of other web pages current page is using.
- Diagrams showing used JavaScript functions.
- Diagrams showing accessed database tables.

Due to a large number of scripts a tool for static analysis of php web application, phpModeler was developed.

4.1 phpModeler

phpModeler is a Java application developed with the goal to facilitate static analysis of the web user interface of the iForestFire system, but it can be used for static analysis of any other web application developed using php, JavaScript or HTML.

The main goal of the application is to generate static UML diagrams of the selected web application using WAE (Web Application Extension) for UML.

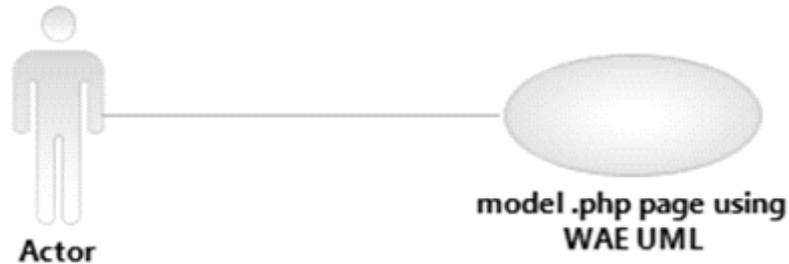


Illustration 10: phpModeler use case

The application can be used for generating UML diagrams displaying:

- Resources of other web pages that the current page is using.
- Used JavaScript functions.
- Accessed database tables.

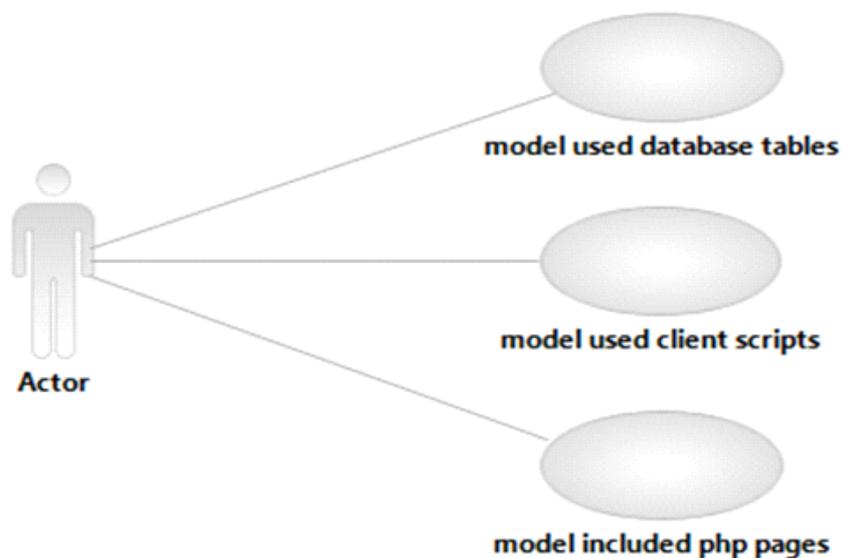


Illustration 11: phpModeler use case

phpModeler is comprised of the following components:

- **phpCodeCrawler** is a page parsing component that generates an object model of the selected web page. The object model is defined in a custom xml format and stored in a temporary file.
- **UmlGenerator** is a component used to generate UML diagrams in a standard format used by the majority of UML tools.
- **Model** is a JAVA class library defining a WAE UML model.
- **phpModelerPlugin** is an Eclipse plug-in integrated in the Eclipse IDE enabling users to generate UML diagrams of selected php pages.
- **phpModelerGUI** is a standalone JAVA application that can be used to generate static UML diagrams of selected php pages. It's functionality matches the functionality of the phpModelerPlugin component and it was developed that one could generate UML diagrams without the need of Eclipse IDE.

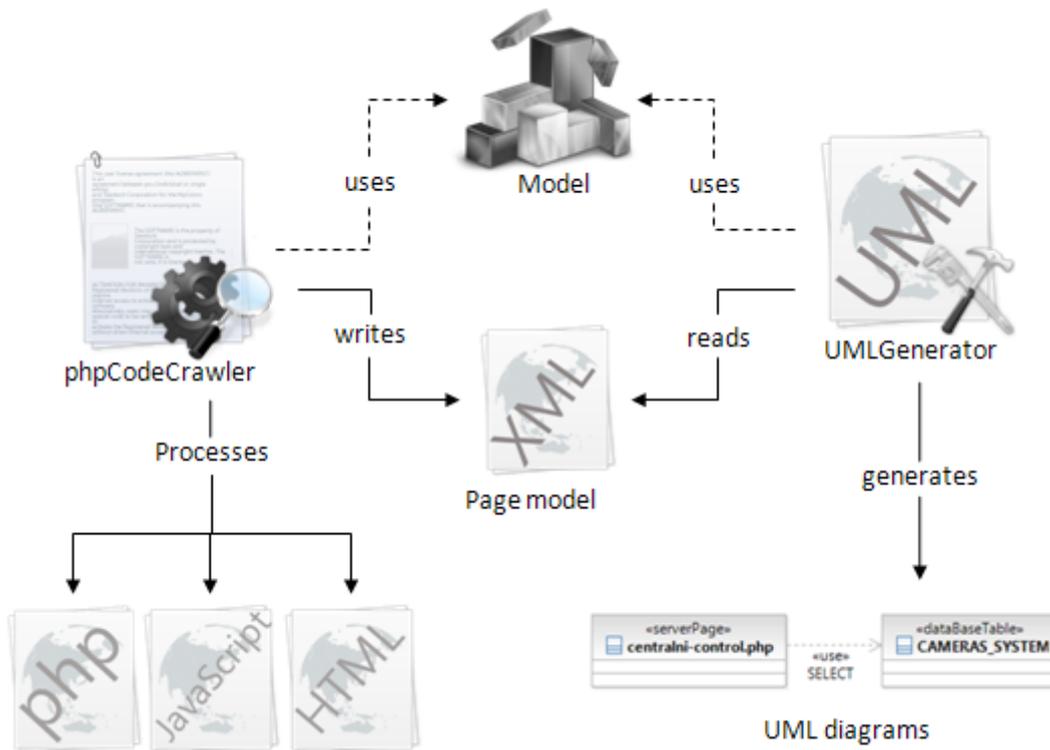


Illustration 12: phpModeler components

The following pictures show the user interface of the phpModeler application:

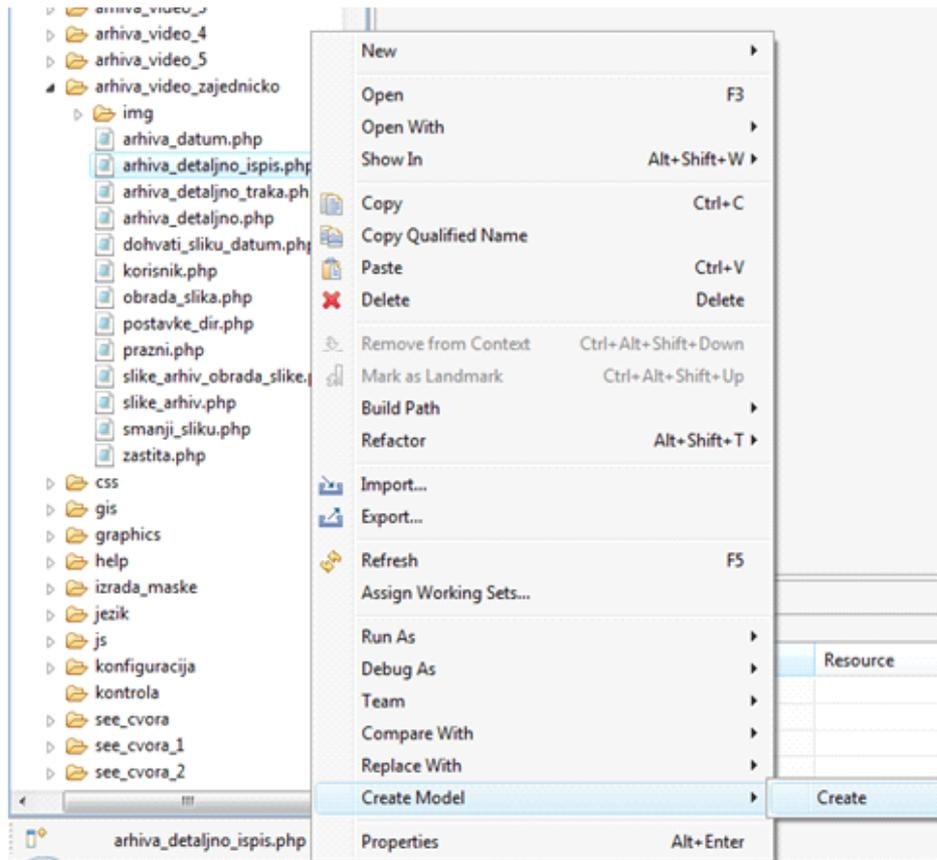


Illustration 13: phpModelerPlugin in Eclipse IDE

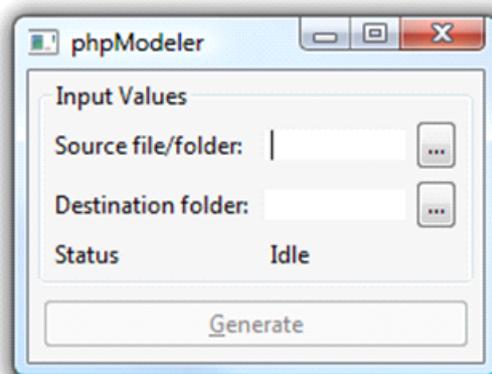


Illustration 14: phpModelerGUI

The following diagram can be used to show the process of generating UML diagrams:

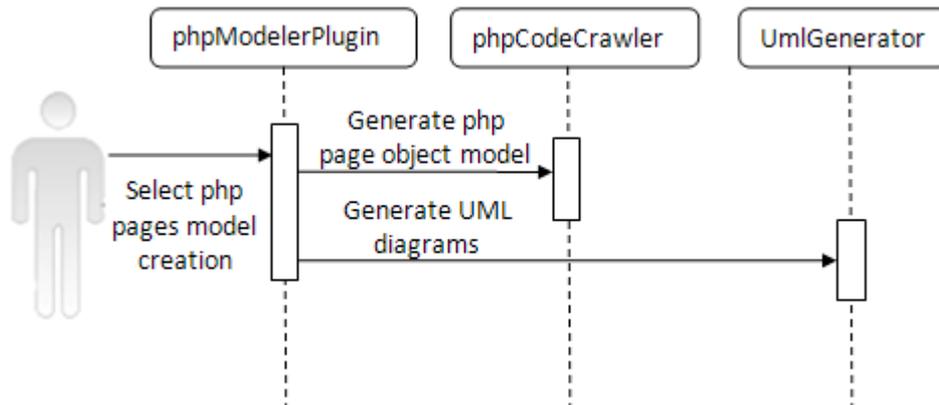


Illustration 15: Generating UML diagrams with phpModelerPlugin

phpModeler can be used to generate the following types of diagrams:

- Used database tables diagrams

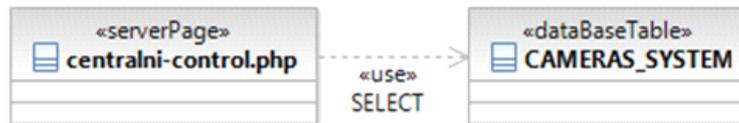


Illustration 16: Used database tables diagram

- Used server scripts diagrams

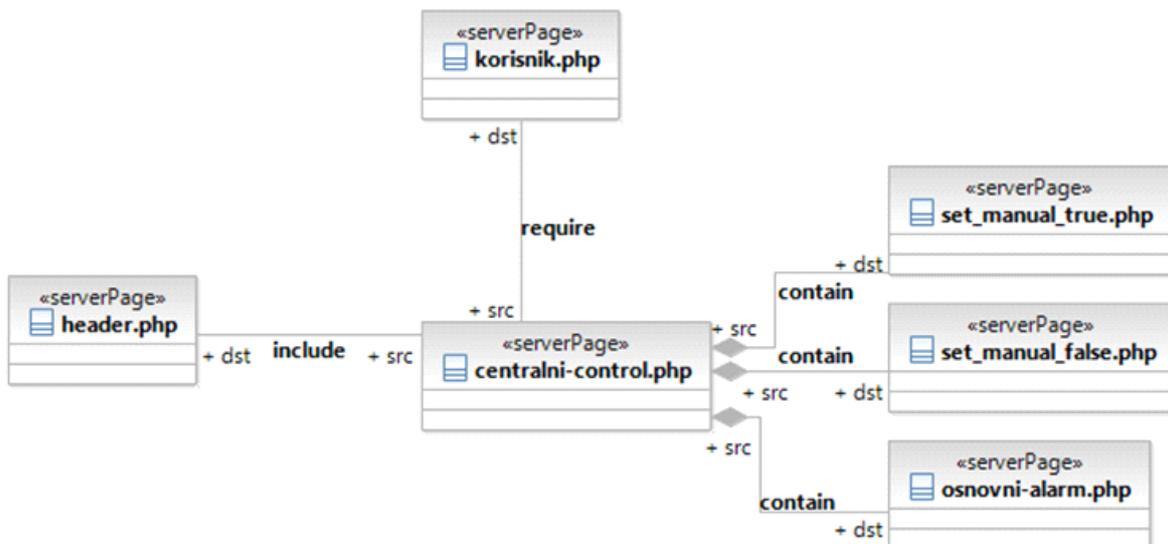


Illustration 17: Used server scripts diagrams

- Used client scripts diagrams

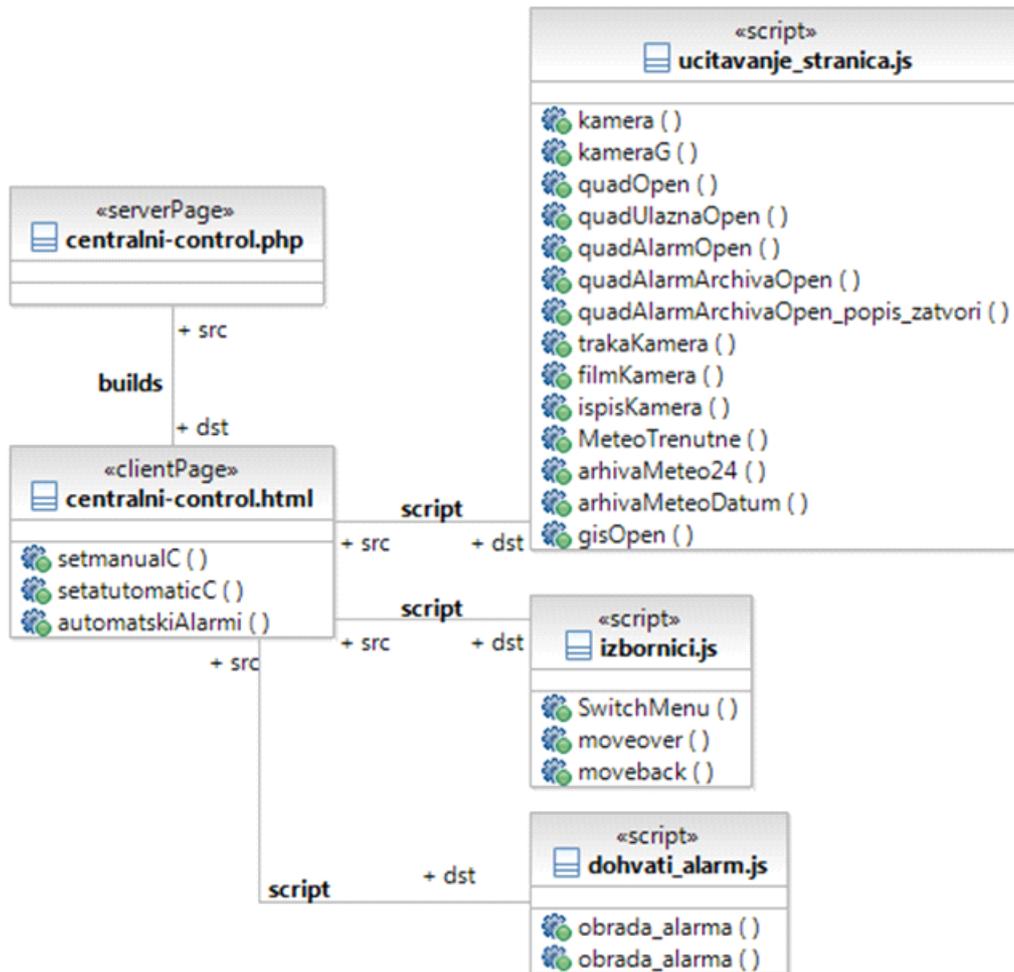


Illustration 18: Used client scripts diagram

phpModeler has a feature that enables the user to create, for each entity, diagrams that show which other entities are dependent of it. This feature simplifies the process of change.

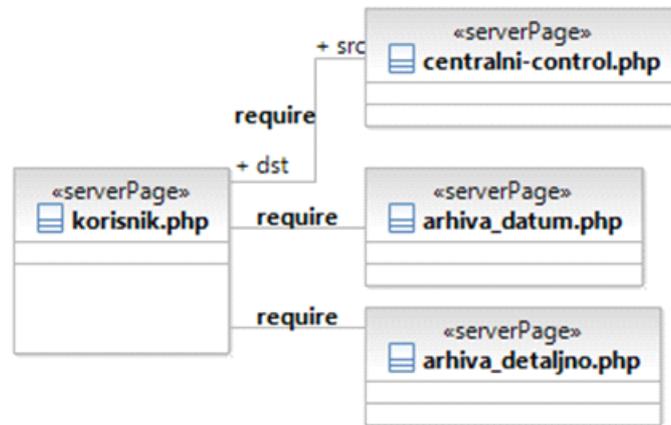


Illustration 19: Diagram that shows which scripts use korisnik.php

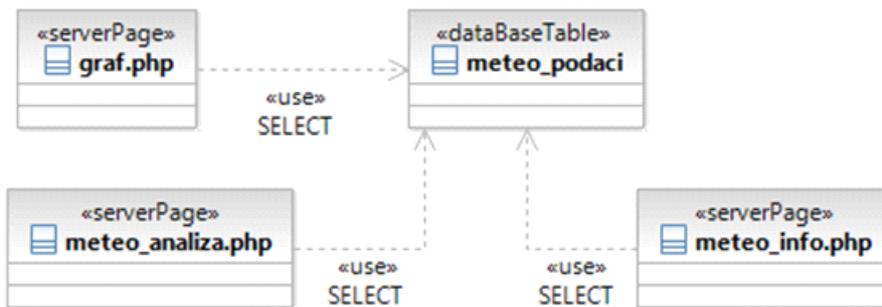


Illustration 20: Scripts depending on meteo_podaci table

5 Fire detection application

Fire detection application detects fire using intelligent image analysis, trying to find visual signs of forest fire, particularly forest fire smoke during the day and forest fire flames during the night. If something suspicious is found, a prealarm is generated and the appropriate image parts are visibly marked.

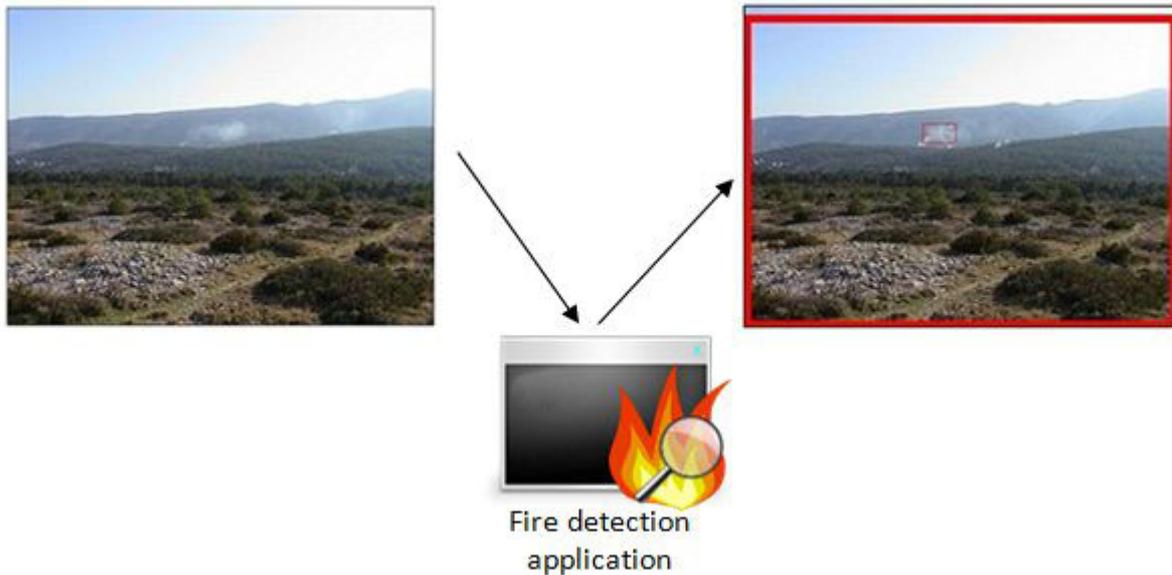


Illustration 21: Fire detection by image analysis

The process of fire detection starts with reading information from the database in order to get information about installed cameras and their preset positions. Then, for each preset position of each camera, fire detection application starts a new parallel process “cameraProcess()” which analyses images from that preset position.

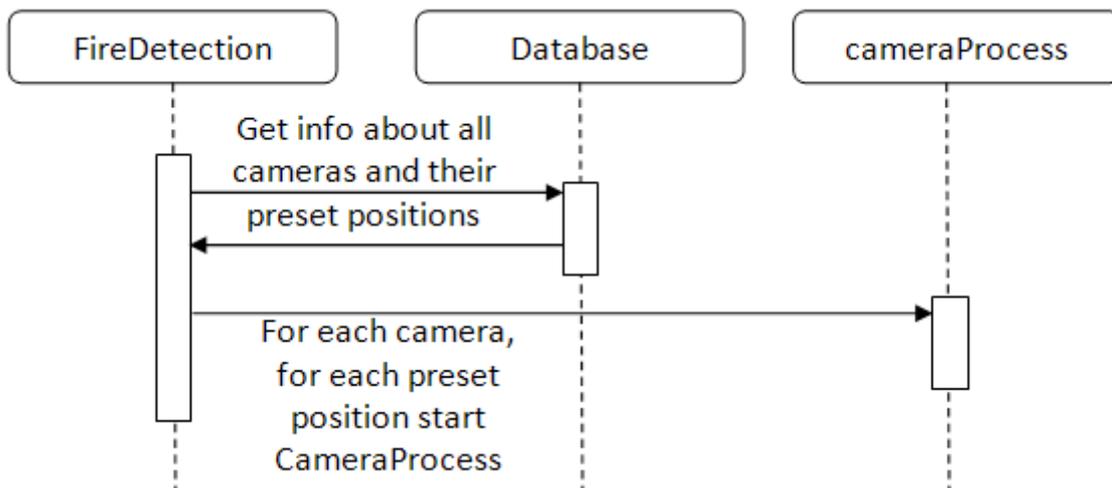


Illustration 22: Fire detection sequence diagram

When PresetAgent from the multi-agent application saves a new image it signals it to the cameraProcess which analyses the new image (using motion detection, color detection...). If the cameraProcess finds an alarm it saves the alarm in the database and marks fire elements (smoke, fire) on the image.

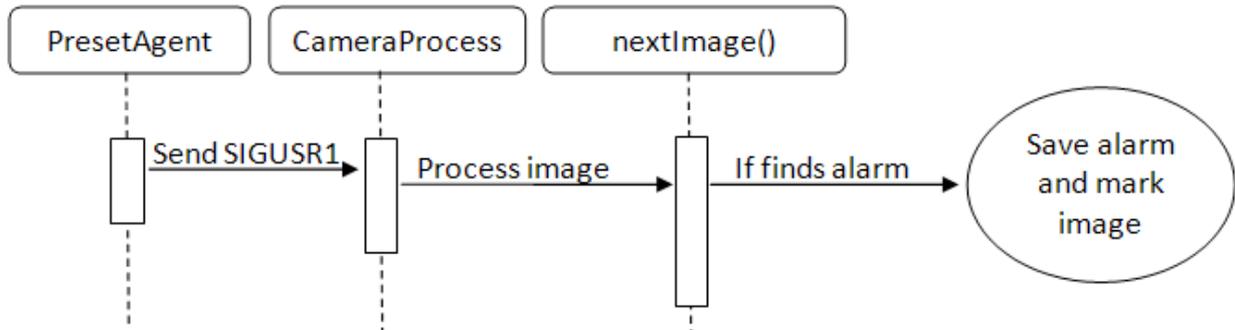


Illustration 23: CameraProcess