An Extendible Simulator for Sensor Network Simulations

Maarten Corzilius
Michiel Kamermans
Tijn Schmits
Mark Smids

Universiteit van Amsterdam, 2005

Abstract

This article concerns the implementation of a simulator for sensor networks operating in crisis situations. The simulator runs based on scenario scripts defined in XML and is designed to interface with script-defined virtual sensors as well as distributed perception networks to generate scenario analyses when acting as web service.
In its current implementation, this simulator runs as an independent program under a java environment, modelling the world relevant to a crisis scenario and giving visual feedback on what is happening during the processing of a crisis script. We outline the design of this simulator, exemplify its function by illustrating what scripts are used, and how the components of the simulator interface, as well as suggest how this system can be extended.

1. Introduction

One of the most important aspects in large scale crisis situations is situation awareness. The availability of information can be vital to rescue workers and people on the scene, where a lack of information can result in uncoordinated actions that may lead to unnecessary loss of life or damages. Situations such as where the fire brigade enters a building to rescue people trapped inside while an on scene report to the police contains the fact that the building is in fact vacant, leads to unnecessary steps being taken to resolve the crisis. This problem can be avoided by proper information management.

The current decentralised method of information management involves supervising coordinators between the various parties involved, who are highly dependant on low level and often scene localised information. It is their task to make sense of the many reports coming in during the hectic period of the crisis and to convey their analysis of the crisis and recommended actions to all parties involved.
One way to augment this process is to employ sensor networks for information gathering. Sensor networks play an important role in the process of gathering information regarding crisis situations, as they can supply information without human supervision or operation. Current common sensor networks are for instance CCTV systems in high risk areas such as light industrial sites, or sensor grids for monitoring emissions and temperatures at chemical plants. Another example is the increasing number of CCTV in city centres in the Netherlands, used by emergency services.
With the popularity and effectiveness of these networks increasing, the road has been paved for integrating autonomous intelligent components into these networks, to fulfil the role of monitor and crisis analyser. Networks that contain such components are known as Intelligent Sensor Networks, of which the ideal form can be thought of as a decision making system that is tapped into all sensors present on the scene of a crisis. This decision making system would be able to poll all available resources for information required to

come up with an accurate analysis of the crisis situation, and recommend courses of action for the various emergency services on the scene. This collective system of sensor networks and decision mechanism is currently being researched by the University of Amsterdam, in association with various organisations, as “the Combined Project.”

Of course, before intelligent sensor networks can be used in real life, they need to be thoroughly tested to make sure they perform as well as, or even out perform, human operators. The most effective way to do this is to run simulations of crises and see whether the decision making mechanism comes up with analyses appropriate for the simulated crisis it’s been assigned to operate in. The simulator used for this essentially simulates all sensor readings during a crisis, and the decision making mechanism is given this sensor data to analyse the crisis while the simulation runs.

This article focuses on the simulator aspect of the intelligent sensor network, its construction, and its extendibility to operate as an off site remote service using web service technology [1], as well as intelligent agent architectures currently available.

2. Project background

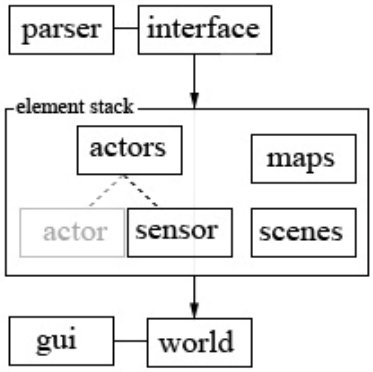
The task of creating a functional simulator with graphical user interface, operating on a simulation script was assigned to four master students of artificial intelligence at the University of Amsterdam as part of the course “Design and Organisation of Autonomous Systems” coordinated by drs. Arnoud Visser. Additionally, the task was allotted a one month timeframe to be completed in, with Peter Lith being assigned as project supervisor.

Given the time constraints, certain functionality aspects of the simulator were prioritised, such as proper scenario processing and visualisation using the platform independent JAVA language, while other tasks were given low priority, such as the implementation of the simulator as web service and embedding it in current autonomous agent architectures.

3. Simulator design

A functional simulator can be divided into three principal components. First and foremost, there is the simulated reality, housing everything from simulation maps to simulated actors. This part we have labelled “the world” in our simulator design. Secondly, this world needs to be visualised in some way, which in our implementation of this simulator is done through a straightforward graphical user interface. The third component that can be identified to run concurrent with the world itself is the processing of the simulation script. This process, while not a principal component in terms of the simulation, is essential for setting up the world and everything in it, as well as providing the script for how the world changes over time.

Because the simulation scripts are in XML [5] format we were able to implement commonly used systems for processing XML documents, providing the simulator with the freedom to simulate any given scenario involving



The object relations within the simulator

sensor networks at any location. Given the fact that the scripts were written in a highly flexible mark-up language, a wide range of possible simulations is possible, provided the information describing the scene follows the form chosen to represent scenarios. We start by describing this format, as it in part functions as a template for how to construct and model the world.

3.1. XML scripts

Simulation scripts are divided into two main blocks. The first of these blocks define the attributes of the world to be simulated, and contains information sets for actors, sensors, and maps relevant to the simulation. While under the current format only sensors and maps are used in simulation scripts, the extension for actors (such as people on the scene or simulation relevant moving objects like vehicles) can be easily added because of

```
<definitions>
  <sensor_definitions>
    <templates>
      <sensor_template type="sensorsuite">
        <subsensor type="camera"/>
        <subsensor type="microphone"/>
        <subsensor type="GPS locator"/>
        <subsensor type="gas sensor"/>
      <sensor_templates>
      <sensor_template type="camera">
        <unit name="frame">
          <preset xdim="640"/>
          <preset ydim="480"/>
          <preset framerate="25"/>
          <preset resolution="24"/>
        </unit >
      </sensor_template>
      <...more templates...>
    </templates>
    <scene_sensors>
      <sensor id="ss.001" type="sensorsuite" location="N 132 42 3 E 67 20 0"/>
      <sensor id="ss.002" type="sensorsuite" location="N 132 42 10 E 67 20 4"/>
      <... more sensors ...>
    </scene_sensors>
  </sensor_defintions>
  <map_definitions>
    <map  id="map.001"  file="map-001.jpg"  topleft="N  132  42  0"  E  67  19  0"
topright="N 132 42 0" E 67 21 0" bottomleft="N 132 43 0" E 67 19 0" bottomright="N
132 43 0" E 67 21 0"/>
  </map_definitions>
</definitions>
```

Figure 1 – XML definitions block

the format chosen. An example of this block has been given in figure 1.

As can be seen in the figure, the sensor definition block is separated into two subsections, one for sensor template definitions and one for actual scene sensors. This separation means that it's quite easy to add new types of sensors when required, as well as adding new sensors on the scene of existing types, without having to completely specify scene sensors in terms of subsensors and their measure units and presets.

Also apparent from the figure, both maps and scene sensors carry GPS location values. For sensors, this value denotes the location of the sensor in the world, where for maps the locations stand for the corner points of the map, which is used to align all the sensors visually.

The second block in the script contains all the dynamic information, represented as scenes. Each scene describes, at a certain moment in time, both the current situation, as well as the changes that occur in the world at that moment, in terms of actor movement, sensor readout and map updates when required.

As can be seen in figure 2, each scene has a title and a natural English description, as well as optional action and event blocks. The action blocks allow the simulator to be triggered for input, while the event blocks specify events occurring in the world, such as sensor readings changing, or location changes warranting the loading of a different map in the simulator.

Before the information in this script can be used by the world, it first needs to be turned into program objects. This is done by parsing the XML using a SAX parser, and feeding the information obtained to an interface that converts the plain text into program objects. The SAX parser is characterised by running through an XML document without retaining any history, so intelligent parsing of incoming elements is important to ensure that the world gets proper objects to work with. This is done using an interface that turns generic element representation into real objects, based on their type and content, which stores all these objects until the world is built.

```
<scene time=”timestamp”>
  <title>Scene title</title>
  <description>Scene description</description>
  <actions>
    <action>Trigger an action</action>
    <... more actions ...>
  <action>
  <events>
    <event type=”script/sensor”>
      <description>event description</description>
      <s_event sensor="sensor name">
        <measure unit="unit" value="measured value"/>
      </s_event>
      <command>register value</command>
    </event>
    <... more event ...>
  </events>
```

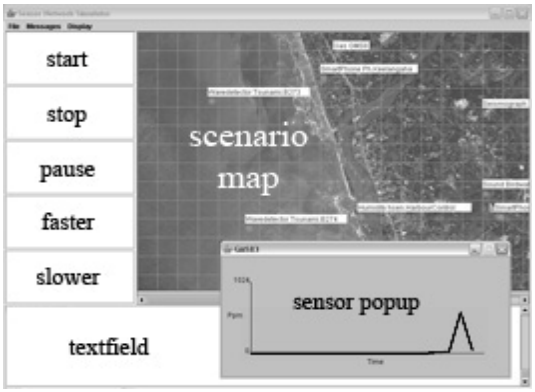
Figure 2 – XML scene block

3.2. Creation of the world

When the world is created, it requests all sensor templates from the interface, as well as all the scene sensors, using the templates to create instances of the sensor types for each scene sensor it gets from the interface. It also requests all maps from the interface, after which the world sets itself to initialised and ready to start running the scenario.
Processing of the scenario is based on timestamps. The world has an external clock running and pre-fetches scenes from the interface, letting the external clock run until the timestamp of the clock matches the timestamp of the next queued scene. When the two clock match up, the scene is processed, and the world modifies any sensors, maps and actors that may be indicated as having changed by the event.

3.3. GUI interfacing

Whenever the world modifies any of its internal settings based on processed scenes, this change needs to be visualised to the user. This is done by calls to the GUI.
The GUI initially starts up as a blank system, waiting for the world to finish setting up all the initial scene sensors and available maps. When the world has finished doing this, it sets a flag that lets the GUI know that it is ready to start processing scenes. The GUI then loads the initial map as indicated by the first scene in the script, and visually places all the sensors on this map at their designated GPS locations.
The GUI also has a set of control options to load scripts, start, pause/resume and stop playback of scripts, as well as controls to speed up or slow down the simulation.



The GUI interface with a sensor popup

3.4. The sensors

In this system, virtual sensors are created based on their definition in the XML script. Given the modular nature of the way sensors have been implemented in the current Simulator, adding new types

The simulator GUI in action re

all that needs to be done is to define a custom sensor type extending the basic sensor frame, with XML definitions for the units and presents for these units in the simulation script. A sensor template class can then be written that generates the sensor based on this definition. This has as advantage that even real sensors can be described in terms of their units and presets in the script, which the world can then theoretically work with.
Each sensor also has the ability to generate popups that can be sent to the GUI to visualise the sensor's current readings as well as reading history in the form of text and graphs, depending on what makes most sense visually for the user.
However, external real world sensors require the additional presence of a

communication system, so that they can interface with the world. For this the concept of the web service is best suited.

4. Web services

In broad terms, a web service is a program which makes itself available over the internet and uses an XML based messaging system to send and receive information from other online sources. The information exchange is typically done using messages formatted according to a known message structure. Currently, most web services use the Web Service Description Language (WSDL, [3]) to define their message systems, which allows other programs to interpret and communicate with any web service that utilises WSDL, regardless of the operating system or programming language the program was written in.
The use of WSDL, an XML based system, allows web services to tie into each other autonomously, opening up the possibilities of web-based autonomous agents, and it is in this context that the real power of the simulator, and the sensor network it is meant to test, can be found.
The power lies in the fact that through the use of web services, there is no longer a need for a centralised system. If real sensors instead of virtual sensors are used then these can, by implementing web service front ends, interface with the simulator from any given distance, as long as there is a web connection between the sensors and the simulator. Likewise, a decision mechanism can be located anywhere and still fulfil its role as analyst as long as it can connect, as agent, to the simulator.

4.1. The DPN

A currently interesting decision mechanism that is being considered for use in the intelligent sensor networks discussed is the Distributed Perception Networks (DPN), which is based on the concept of a Bayesian reasoning network operating on values derived from agents (sensors), in which resource allocation of

```
<wsdl:message name=”sensorName”>
  part name=”Sensor” type=”xsd:string”></wsdl:message>
<wsdl:message name=”sensorReading”>
  <part name=”Reading” type=”xsd:string”></wsdl:message>

<portType name=”sensorPort”>
  <operation name=”poll”>
    <input message=”tns:sensorName”>
    <output message=”tns:sensorReading”>
  </operation>
</portType>

<binding port=”sensorPort”>
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http”>
 <operation name=”getReading”>
    <soap:operation soapAction="http://the_combined_project/getReading"/>
    <input ... />
    <output ... />
 </operation>
</binding>
```

Figure 3 – a WSDL specification for World

which agent is reserved for which part of the network is dynamically determined.
This DPN requests information from sensors, polling them for their values to be used in the perception network to come up with hypotheses and situation analyses, or even issue commands to sensors to request more detailed information from them while the DPN updates its analyses.
The requesting of information and issuing of commands is performed through web service interfaces, and can be done either directly between the DPN and sensors, or tunnelled through for instance a simulator that is processing the sensors of interest to the DPN.

4.2. Extending the simulator

In order to set up the simulator as a web service for remote use, a web service front end plug-in must be written for the World element of the simulator. The World should function as a service for the DPN, providing the DPN with sensor readings and other data which the DPN should require. The DPN should be able to invoke a series of public World functions through proxy, defined by the Simple Object Access Protocol (SOAP, [2]), which is a lightweight protocol for exchange of information in a decentralised, distributed environment standardised by the W3C. To give an idea of how the World might function as web service we propose a WSDL specification as listed in figure 3 on the previous page.
In this specification functions and arguments are defined which will be used during the communication between the World and the DPN.

Note that we, keeping with the nature of XML, have defined all arguments to be strings, bearing in mind that many different sensors can have many different return types, all of which can be represented by a string. The precise format of the strings is not set, thus any desired format containing type specification of sensor readings can be set, to provide an indication for the client to further process the data.
Future developments might include a request-response operation pattern, which will provide two-way communication, thus providing the DPN with the means to not only request information from sensors, but also to issue action commands to the sensors. Keeping in mind that changes in the world are event based, the web service plug-in should also be designed to relay these action commands as events.

5. Optional Further Extensions

Currently certain features associated with simulation are not implemented, such as moving single actors and groups of actors, visual feedback for risk assessment (grid colouring), user interaction for sensor manipulation and interfacing with real sensors. Also because in its current implementation the Simulator is only capable of processing the script it is unable to simulate certain crisis elements itself. One might think of the world creating sensor events which relate to the dispersion of gas after the simulator (either as random factor or as user input) has triggered a gas leak somewhere. This type of crisis prototype event can be used to add versatility to a crisis simulation.

5.1. Actors

Actors have been partially implemented, with sensors being specific extensions on the actors concept, but no direct instances of actors have yet been created. Like sensors, actors can move around in the world, but unlike sensors they cannot be polled. Their movement can be used by the DPN for scene analysis, but no communication is possible between the DPN and actors, unless actors carry a sensor like a mobile phone. Adding actors into the simulation can also clarify the nature of the crisis to the user, as certain people or vehicles typically play a key role in a crisis situation. For example, the display of a leaking tanker in the Rotterdam harbour will make a crisis simulation clearer to users (and spectators at demonstrations) than if this were omitted.

5.2. DPN hypothesis overlay

Currently, maps are built up as multiple adjacent map patches, which can be used for visualising local information. One of the most obvious instances of local information is local risk assessment. During a simulation run the hypotheses about the risk in certain areas generated by the DPN can be visualised for instance by grid colouring of the patches using the common red-green concept to stand for unsafe and safe areas respectively.

5.3. User Interaction

The current implementation is also limited in the sense that sensors displayed in the crisis scene can only be manipulated by events described in the XML script. For interactive use it would be functional if certain properties of sensors could be changed by the user in real-time, such as camera orientation, sensor suite location or sensor sample rate. A second form of functional user interaction that can be added is runtime control where the user can skip to a certain time in the simulation run, rather than having to wait for the simulator to arrive at the desired point in time.
Implementing user interaction will require a link back from the GUI to the world, something which in the current implementation does not exist yet.

5.4. Real sensor interfacing

A final extension is the use of real sensors instead of virtual sensors for demonstration purposes. In order for this to work, the real sensor will need to be hooked up to its own web interface, allowing it to directly communicate with the world in a language the world understands. Events sent by this sensor need to be placed in the proper spot in the event queue, which is not real-time in the current implementation, given that it operates on a predefined script. Adding real sensors will require modifying the way the world stores and processes events.

6. Concluding remarks

This project had to be extendible from the outset, with this in mind we have created an easily extendible and where required easily revisable code base. Since this project will be continued in the coming years it is important that this system is transparent and easy to use. We believe we have succeeded in this

task and have taken future technology into account while laying the foundations for the simulator aspect of the combined Intelligent Sensor Network project.

Technological references and whitepapers

[1] “Web Services Essentials”, Chapter 6: WSDL Essentials, O’Reilly, http://www.oreilly.com/catalog/webservess/chapter/ch06.html
[2] “SOAP version 1.2 specification”, W3C recommendation, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/
[3] “Web Services description language”, W3C recommendation, http://www.w3.org/TR/wsdl
[4] “Sensor Network Simulator Project Page” (this article can be found here in digital form), http://student.science.uva.nl/~msmids/DOAS/
[5] “Extensible Markup Language 1.0”, W3C recommendation, http://www.w3c.org/TR/2004/REC-xml-20040204/