# A Performance-centric Approach for Complex Decisions Support

Ate Penders
Thales Research&Technology,
Delft University of Technology
ate.penders@d-cis.nl

Ana Lucia Varbanescu
University of Amsterdam
a.l.varbanescu@uva.nl

Gregor Pavlin
Thales Research&Technology
gregor.pavlin@d-cis.nl

Henk Sips
Delft University of Technology
h.j.sips@tudelft.nl

## ABSTRACT

Many situations in the security domain require decision-making based on complex data, i.e., many variables which need to be taken into account before adequate decisions can be made. For example, in a surveillance scenario, the size and complexity of the area of interest, the mix of objects, and the unexpected behavior of suspects are just a few examples of complex variables to be analyzed in the process. Existing decision support systems provide some analysis, but are typically limited in the complexity they can handle. Therefore, users end up with simplified models which often suffer in the accuracy of their decisions and, ultimately, may lead to incorrect decisions. In this work, we present a framework that can scale to cope with the complexity and time requirements of real-world scenarios, while remaining flexible to handle the ad-hoc adaptation to the situation. We discuss the challenges and solutions for such a scalable and flexible system, and validate it using a target tracking scenario in urban environments of different sizes.

## 1. INTRODUCTION

Border security, maritime security, wildlife protection, or search & rescue operations are examples of critical applications in the security domain. Situation assessment in the security domain requires unstructured (big) data and complex processes to be combined for decision making, which in turn should lead to one or more simple and concrete actions and orders, like "apprehend suspect" or "ignore, false alarm". Ideally, this translation of complex processes and data into actions is performed by a (semi-)automated system for decision support.

The data in all applications in the security domain come from some form of surveillance. The resources used for surveillance (i.e., the sensors) and their availability are dependent on the application: with border security, the pres-



*ICPE'17, April 22 - 26, 2017, L'Aquila, Italy*
*Industrial Paper*



ISBN 978-1-4503-4404-3/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3030207.3044532

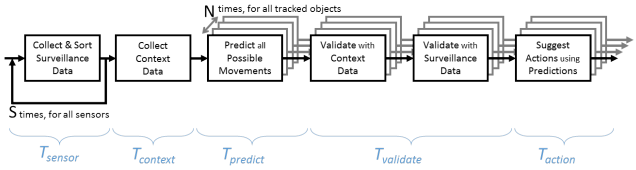


Figure 1: The pipeline of processes in a surveillance system, from extracting the surveillance data from each of the $S$ sensors to the suggested actions based on the predicted movement of the $N$ target objects. This pipeline is executed for every time step.

ence of security cameras can be expected, while search & rescue operations often depend on human surveillance or cell towers. Regardless of these differences, in all cases, the data read from many relevant sensors, of different kinds, must be combined. Reading relevant surveillance data is an essential part of situation assessment. Furthermore, the data is unstructured, heterogeneous, and can be very large in size. A first critical task of the system is extracting and filtering the surveillance data relevant for the decision making process.

The filtered set of surveillance data is used for further analysis. For example, it can be used to create situation awareness by using tracking algorithms that collect data of specific objects over time and predict the most likely next position of these objects. Such prediction is a complex and time consuming task, requiring both historical data of the object, if available, and information on the context of the surroundings in which the object moves (e.g. road maps, traffic information, road blockages) [8]. Therefore, the second critical task of a decision support system is the complex and time consuming analysis process (or group of processes), where the filtered set of surveillance data is used to generate relevant information about the target object (e.g., its destination), eventually leading to one or more concrete actions.

Figure 1 presents the sequence of the components required for a decision support system for object tracking. The system, which collects data from $S$ sensors to track $N$ objects, is organized as a pipeline of processes, and suggests an action per object, based on the surveillance data collected at a specific time. The pipeline is executed at each time step; ideally, a decision had been reached in all previous time steps before a new step begins, a condition that limits the possible frequency of the system. We note there are two iterative blocks in the system. The first one iterates over the num-

ber of available surveillance processes ($S$) to collect all the surveillance data that can later be used by the tracking processes; the second one is performing the tracking process and action suggestion for each of the $N$ tracked objects. Thus, the time step of the system is at least $T_{seq}$, defined in Equation 1; by design, the performance of the decision system is limited by the time-to-decision, $T_{seq}$.

$$T_{seq} = T_{context} + S \cdot T_{sensor} + N \cdot (T_{predict} + T_{validate} + T_{action}) \quad (1)$$

The overarching goal of our work is to provide flexible, high-performance decision systems. Therefore, we propose a first generic framework for security applications, which can adapt to the type of surveillance resources and decision-making scenarios, while still providing fast decisions. Specifically, our main requirements for the framework are: (1) minimize the time per step, and (2) scale with *both* the number of sensors and decisions to be made. We present in this paper our framework's design and implementation, followed by the performance analysis of a case-study. These preliminary results show that our performance-centric design is flexible and scalable, holding promise for a production-ready decision support system for security applications.

The remainder of this paper is structured as follows. In Section 2 we introduce a motivating scenario to demonstrate the requirements of a real surveillance application. In Section 4 we present the common challenges of decision making in security domain applications. We follow-up with the design and implementation of our framework, as presented in Sections 5 and 6, respectively. Section 7 discusses the usage of the framework, in Section 8 the framework is validated, while Section 9 discusses our preliminary results. We conclude the paper in Section 10.

## 2. MOTIVATING SCENARIO

Situation assessment is a combination of knowledge about the environment, the whereabouts of the target (in case of tracking), how the former affects the latter, and viceversa. All this knowledge is specific to the desired type of assessment, making support systems difficult to generalize.

Target tracking in urban environments is one example of complex decision making, representative in many aspects for various situations in the security domain. Across this paper, we use target tracking as a practical example to illustrate the baseline requirements for a decision support system.

In the scenario of target tracking in urban environments, we are interested in the apprehension of the target by means of blocking streets. The situation assessment therefore should give insight in the required number of blockages, the impact blocking the streets has on the "normal" traffic, and the availability of law enforcement in the area. The scenario is a combination of separate complex processes: (1) prediction of target movement, (2) traffic monitoring, (3) minimal blockage determination, (4) impact estimation, and (5) law enforcement localization.

Each one of these five process may consist of one or more (sub)processes. For example, the prediction of target movement has a prediction model that uses the last known position and direction to predict the next likely position; observations from sensors are used to improve and validate previous predictions, as well as adjust current predictions; finally, the environmental context (e.g., road maps), is also used to improve and validate predictions.

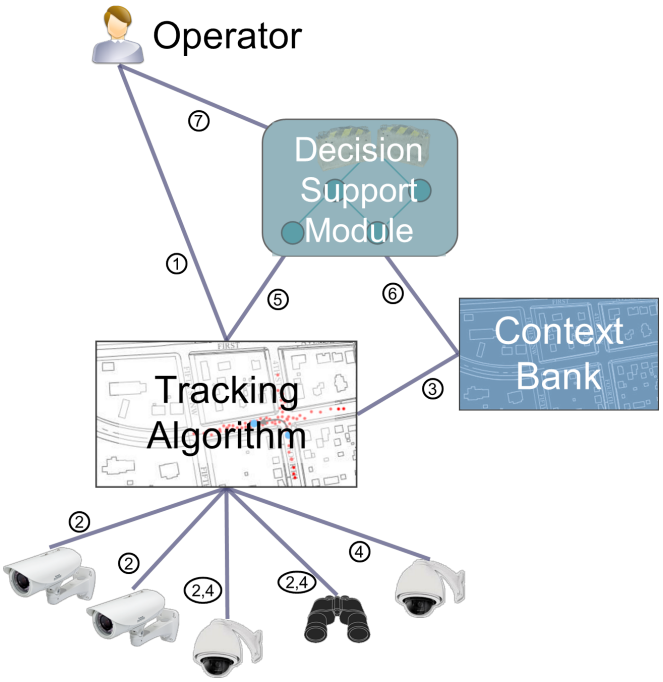


Figure 2: A scenario for target tracking in urban environments: (1) initiate tracking, (2) request sensors in area, (3) validate prediction based on context, (4) new sensor for updated area, (5) report predictions, (6) use context for decision proposal, and (7) suggest action.

Target tracking is typically initialized by an operator, an officer who gets the order to follow and apprehend a suspicious person or vehicle. This request triggers the start of a tracking process, which in turn triggers requests to sensors (e.g., license plate recognition cameras) in a specific area. The tracking algorithm, for example a particle filter [5][9], periodically predicts all possible next positions. The prediction is validated (and adjusted) when the sensors detect the object of interest. In urban environments, but also in other situations, the prediction can be further improved based on knowledge about the context (e.g. road maps, traffic, construction): the target tracker sends its prediction to a context bank, which adjusts the likelihood of these predictions based on a database of context information. Based on the updated prediction the tracker will update its request to sensors[1]. New predictions are reported to the operator and/or a decision support module. The decision support module is in place to assist the operator in providing suggested actions for apprehending the target. The target tracking terminates when a decision is made by the operator or the target is lost (i.e. no observations are done by any sensor for a long period of time, hence the predictions have very low probability). Figure 2 shows a graphical illustration of the described scenario.

## 3. RELATED WORK

A decision support system requires a large collection of (different) processes able to handle and manage the het-

[1]The last position of the target with a radius of the maximum traveling distance between two time-steps of the tracking algorithm plus some margin.

erogeneous and unstructured surveillance data coming from different (types of) sensors. Ultimately, all these data feed a set of predictor processes.

As described in [3], decision support systems for situation assessment scenarios have to be adaptive because (1) the data-to-decision interval is very short, and (2) the evolution of the situation is usually unpredictable. Thus, the set of predictors interested in particular data will change between iterations, due to the dynamic behavior of target objects. Usually, a prediction process interested in surveillance data from a sensor in one iteration, may not be interested in surveillance data from the same sensor in the next iteration, because its target object moved out of range. This dynamic binding and un-binding of processes requires a flexible environment for both expressing *and* analysing the scenarios.

In such a rich environment of heterogeneous processes with dynamic interconnections, a message passing system [2] ensures flexibility and scalability, while employing parallel processing can improve performance. Therefore, in building our decision support framework, we got inspired by actor-based systems [1][4][10], i.e., systems of concurrently executing objects (i.e. actors) that communicate exclusively via asynchronous messages. Each actor buffers the messages it receives in a mailbox and processes them sequentially, one at a time [7]. Where the components in the process chain of Figure 1 can each be represented by an actor, using asynchronous communication avoids unnecessary stalls due to differences in execution time of the various components.

Our design focuses on generality, flexibility, and performance, hence our goal to implement this approach into a framework. To the best of our knowledge, ours is the first framework for decision-support in the security domain which combines heterogeneous processes, actor-based systems, and parallelism, to provide *high performance* in terms of both *speed* and *scalability*.

# 4. CHALLENGES AND APPROACH

The usability of decision support frameworks in real-life situations is often hindered by their performance, as they become too complex to cope with decision taking in due time. Our goal is to address this challenge at a structural level, aiming to improve the per-step performance of data collection, analysis, and decision making by design.

Consequently, we impose the processing time of the decision taking pipeline (Figure 1) to be within given time constraints, as seen in Equation 2, where $T^*$ is the maximum period between observations.

$$T_{seq} \leq T^* \tag{2}$$

Note that the actual time constraints are entirely dependent on the situation. For example, for our target tracking in an urban environment, if we want to track a car driving at an average speed of 30 km/h and the data processing takes 10 minutes (that is, $T_{seq} = 10min$), the car could move 5 km away from the last known position, making any suspect apprehension unfeasible. When the execution time is reduced to $T_{seq} \leq 2min$, the car only moves 1 km, making it still possible for law enforcement to apply the suggested action, and reducing the unpleasant consequences in the area. In a scenario of wildlife tracking, the average speed of moving objects is closer to 5 km/h, making the range of movement in 10 minutes roughly 800 meters; this is an acceptable range. However, because the density of sensors covering the area is likely to be much lower than in a city, chances are high to miss the few available sensors. Again, reducing $T_{seq}$ is desired to decrease the chance of missing the important sensor observations and with it increase the likelihood of success.

It is difficult to determine an acceptable threshold for the interval of observation. In our experience so far, this threshold is a combination of situation specific variables, like average speed of the target object and density of the sensors in the area, and possible/desirable decisions. Therefore, building a framework where performance and scalability are addressed by design, and can be further tuned by using a better computational solution, is the right way forward.

When analyzing the pipeline for decision support, we identify two types of tasks that are both time consuming and involve large sets of data, and therefore may hinder processing speed:

- sorting and filtering surveillance data, and
- analyze the surveillance data and produce relevant target information.

For large-scale scenarios, like decision support for security domains usually are, $T_{seq}$ (see Equation 2) is rapidly increasing since both $S$ and $N$ tend to grow. As increasing $T^*$ is not really an option for real-life situations, we conclude that a sequential solution will not scale. Instead, we must reduce $T_{seq}$.

To reduce the processing time of the data collection and filtering, we distribute the workload, i.e., we move the sorting and filtering closer to the physical location of the sensors, allowing this iterative process to be done in parallel by the sensors. This is possible because each sensor operates independently, so the communication to and from the sensors can be asynchronous. Distribution also allows us to efficiently handle the dynamics in the scenario by connecting and disconnecting to the relevant sensors and only using the data that is relevant for the current situation. The distribution improves performance, but introduces a new challenge: resource discovery - i.e., determining which are the relevant sensors. We address this additional challenge in the next section.

For the task of analyzing surveillance data and producing target information, synchronization between parts of the analysis process is required, making it less suitable for a distributed approach. Instead we apply parallelism to speedup the task itself. For example, a tracking algorithm that needs to predict the most likely next position of the object must consider multiple options, for each option a number of steps evaluate the options to determine its likelihood and finally the different options are compared against each other. In between the different steps (e.g. create new potential option, is this option possible or block by construction, use surveillance data to determine the current direction and compare against this option) some or more synchronization is required but the different options can be evaluated in parallel. Figure 3 shows the setup of the process chain that uses the distribution of the $S$ sensors and a distribution for the $N$ active object tracking. Note that, for simplicity, this figure does not show the parallel design of the analysis steps.

To summarize, with the new processing pipeline based on parallel processing, Equation 2 is rewritten as Equation 3. Where time indicator $T_{communication}$ is the cost of moving

the process blocks to distributed or parallel nodes, introducing more communication and communication over longer distances.

$$\max(\max_{S}(T_{sensor}), T_{context}, \max_{N}(T_{predict})) + \max_{N}(T_{validate} + T_{action}) + T_{communication} \leq T^{*} \quad (3)$$

Our work focuses on designing and implementing a generic decision support framework able to handle such a hybrid solution of distribution combined with parallel computing for surveillance situations. In the following sections, we discuss the framework's requirements and specifications, its architecture, and present a first implementation able to integrate and interpret the data from the active sensors towards recommending a decision.

## 5. FRAMEWORK DESIGN

Figure 3 presents the architecture of the proposed framework. To allow our framework to run efficiently, we must make sure that modules operate without fast processes suffering long idle times due to slower processes. Therefore, our framework uses asynchronous communication in a dataflow like model: as soon as the data is available, it is further communicated to the processing pipeline. With independent sensors, the lack of ordering due to asynchronous communications poses no challenges to consistency. Further, to handle the dynamic nature of decision support systems and the distribution of work, resource discovery and dynamic (dis)connecting of resources is required (illustrated by the solid and dashed lines in Figure 3).

### 5.1 Resource discovery and Dynamic (dis)connecting of resources

The challenge of resource discovery is solved by introducing a *yellow pages service*. As the name suggests, this service acts the same as a lookup table to find addresses of the available processes grouped on their type. When a process $X$ (e.g. a tracking process) needs data from a different type of process $Y$ (e.g. sensor), $X$ will use the yellow pages to discover and localize the instances of $Y$ and use the addresses information to establish a communication link.

It is expected that the system will have many more sensor processes available than the ones the tracking process is actually interested in (i.e., those that are in the area of interest). In our tracking example, a tracking algorithm is interested in sensor data of red cars, but data from a sensor 300 miles from the current position is irrelevant at this moment. As the tracked object moves over time, previously irrelevant sensors can suddenly become relevant and vice versa, meaning that the tracking algorithm needs to be able to connect and disconnect to relevant sensors on demand.

Filtering and selection of available sensor processes based on the interest of the tracking process is called *negotiation*, and it is designed to avoid unnecessary communication: a data request message is sent to all sensors, but those sensors outside the area of interest will simply ignore it and send no response. Negotiation is an instance of the more generic mechanism of dynamic, on-demand activation and deactivation of connections, which allows different processes to connect and disconnect via asynchronous message passing.

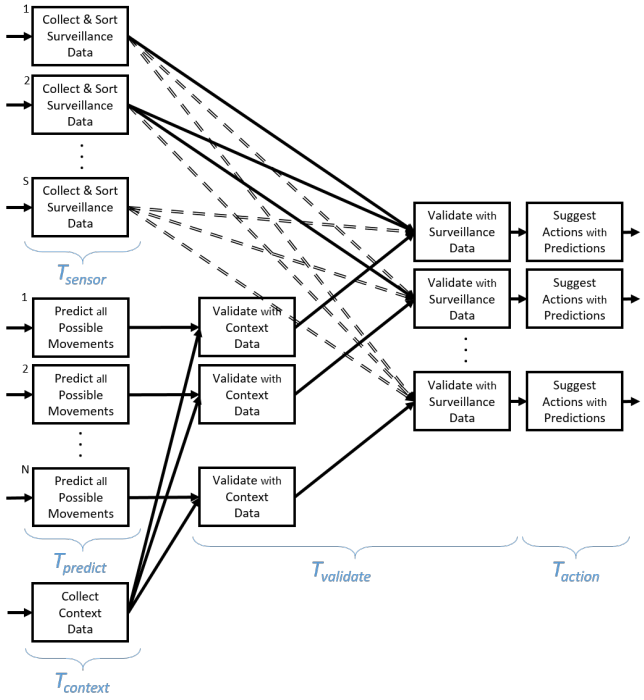


Figure 3: The multiple pipelines of processes in a surveillance system, using our framework. The collection of surveillance data can run in parallel with the execution of the prediction step; afterwards, only the relevant sensors are connected to the validation step. These pipelines are executed for every time step.

### 5.2 Distinguish information flows

As the system is designed to scale with the number of requests, keeping track of the origins of a request and its own flow of information through multiple processes (i.e. distinguish information flows) is another very important feature that has to be handled by the framework. In the tracking example, a request to follow a black van is a completely different flow than that of tracking a red car, and the estimated next position of the red car is of no relevance to the operator interested in the black van. Therefore, each process needs some notion of *task management*: each request from a process will trigger a new task (or replace a task) in the current process, and any result produced in this task is therefore directly linked the request.

To summarize, our framework consists of a yellow pages service that allows the initial resource filtering based on a given set of features (i.e., type), a negotiation mechanism that reduces the list of producers to only the relevant ones (given a specific scenario), and a task manager that allows information flows cross paths without interference (see Figure 4).

## 6. IMPLEMENTATION

Our proposed framework is based on the principles of the Dynamic Process Integration Framework (DPIF) [6] – a framework that allows modular algorithms to connect and allow human experts to interact with one or more of the modules – adapted to high-performance situations, that allows us to find resources using a service oriented approach.

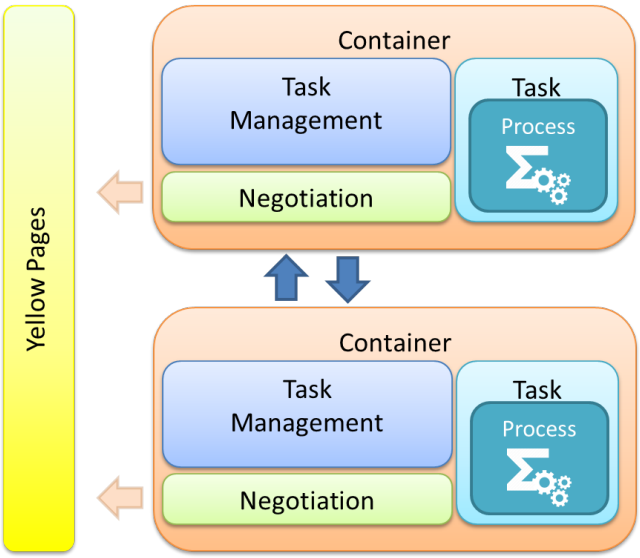


Figure 4: Overview of the framework design. Each process ($A$, $B$ and $C$) is connected to a task; processes negotiate before establishing a connection to exchange information asynchronously through Akka messaging. The Yellow Pages helps discovering and locating other processes; the task management ensures separation of the information flows from and to a process.

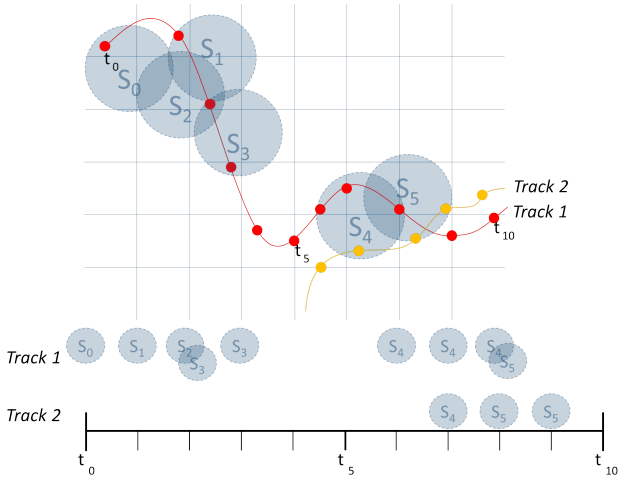


Figure 5: Two target objects move over time in the area of interest (the red and yellow dots). There are five sensors, $S1-S5$, and the task management ensures a clean separation of information flows as two different target objects cross paths and need different data from the same sensors.

For the communication between process blocks we use Akka[2], an actor-based middleware [1] that supports asynchronous communication [10], a feature we require to make the large variety of (types of) processes inter-operable without creating serious performance bottlenecks. Using an actor-based middleware allows us to represent each process block as an actor or group of actors, where actors use asynchronous communication among themselves. For example, a validation process gathers movement prediction, surveillance data, and context data from various resources and pushes its validated predictions to the next step in the process chain. The validation process can be represented by a group of actors, validating, in parallel, all different predictions of the target movement.

For process blocks to discover and be discovered by resources they have to be aware of their process types and the required types of input resources [6]. The process type is registered at the yellow pages service along with the address of the actor performing this type of process. Other processes that require data from this process type as input can now lookup the entries in the yellow pages and use the addresses to establish a communication link to these actors to create an information flow.

The yellow pages service running in the system is globally accessible and allows resource discovery with simple search requests. The retrieved addresses from the yellow pages can be used to establish a peer-to-peer communication between processes. In the initial design of the framework, the yellow pages is a central service, but future designs will support a distributed and fault-tolerant service.

## 6.1 Information Flows

Since process types and their required inputs are not situation specific, and the request from within a situation assessment is, additional filtering is required on top of the basic search provided by the yellow pages, namely negotiation. After a list of potential information suppliers is requested from the yellow pages, the requester sends its criteria (e.g. location, maximum response time) to participate in the information flow to the suppliers. The suppliers will match the criteria against their own settings and decide if they can participate in the proposed information flow, by replying with a accept or reject. The requester now has a set of relevant suppliers and can continue with his specific task. This negotiation procedure is a process running separately from the actual algorithm to reduce interference.

Arrangements to work together on a specific topic for a specific situation (*contracts*) established in time step $t$ may or may not be useful in the next time step $t+1$, depending on the algorithm and the situation. As an extension to the above described negotiation procedure, reevaluating active contracts allows an updated set of criteria to keep the list of suppliers up to date with respect to relevance. In other words, when the process decides that the criteria for the input data have changed, the negotiation procedure is repeated; existing contract are reevaluated and potential new information suppliers are considered. The negotiation allows the system to only connect to the relevant resources, reducing the communication links.

Filtering on the relevance of suppliers is an important step in the creation of information flows. However, since security domains tend to cover large areas it is unlikely for a single information flow to be active in the entire system. With multiple information flows, their communication paths are expected to cross every now and then. By introducing task management, we keep track of the active tasks of every actor, where a task is a wrapper around a local instance of the process that is part of a specific information flow. Any received data is routed by the task management to the correct task, the task in turn needs to transfer the data to the correct part of the process. As illustrated in Figure 5: object one on *Track 1* is covered by sensors $S_1$ - $S_5$; as object 2

[2]Akka's webpage: http://akka.io/.

on *Track 2* crosses the path of object 1, they can both use the data from the same sensors without mixing the actual information flows.

Since both the negotiation and the task management functionality are agnostic to that of the processes, the framework require each process to be wrapped inside a container of generic components (the orange box in Figure 4). The container exists of a task management, the negotiation, an interface with fixed set of process calls, and the yellow pages mechanism (the yellow box). Each process is required to implement a common interface, accessible by the task, with fixed structure to communicate with other processes, using primitives like request for input data, receive input data, and report output data. As the container is agnostic to the implementation of the process, a few configuration parameters, like input and output data formats used by the process (described in more detail in section 7), allow the framework to be adapted to many situations.

## 6.2 Scaling up

For large-scale scenarios, like decision support for security domains usually are, the framework has to be scalable to run hundreds or thousands of processes that are scattered across big areas (like a country's road network). Akka already has support for a large distributed setup, i.e., using multiple compute nodes for hosting and processing the actions of thousands of actors, as well as efficient asynchronous communication between actors.

Increasing the number of sensors, $S$, and/or the number of objects being tracked, $N$, leads to a proportional increase in the number of processes, and therefore actors. Scaling up the number of processes is relatively easy: once the yellow pages system contains the addresses of the actors running the desired process type, any process can exchange data on-demand with other processes. Our current implementation of the framework uses a central yellow pages service accessible by all running actors, this service can easily store tens of thousands of addresses, without any penalty on performance. Future implementations will have a distributed yellow pages, thus avoiding any potential bottleneck caused by quick developing scenarios, where a lot of processes require resource localization very often.

The components of the framework have a small footprint, and the communication over per process is small as well; however, the interaction of all these actors can pose scalability challenges to the system. Meaning, when $T_{communication}$ becomes the dominant factor in Equation 3, thus the communication overhead exceeds the computation times. To further reduce the framework's overhead, each actor runs its different tasks concurrently, by spawning more actors. The implementation of a process can apply a similar strategy when it requires a parallel solution, to improve performance.

In summary, our choice for Akka as a backbone of our framework, as well as the distribution of processes over actors, the simple, query-driven design of the yellow pages system, and the on-demand activation and deactivation of the inter-process links allow for smooth scaling up of the system. The use of parallel and distributed computing, as well as the concurrent design of processes, allows our framework to reach high performance.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns1:processdef xmlns:ns1="xml.process">
    <function name="License plate camera" outputCapability="LPC"
              autoCreateTaskOnStartup="false" className="sensor.MappedSensor">
        <variables>
            <outputRequest variableName="AreaOfInterest" variableType="datatypes.MapData" />
            <outputRequest  variableName="LicensePlate" variableType="java.lang.String" />
            <outputResponse variableName="Location" variableType="datatypes.MapData" />
            <outputResponse variableName="Timestamp" variableType="java.util.Date" />
            <outputResponse variableName="Certainty" variableType="java.lang.Double" />
        </variables>
        <arguments>
            <argument name="name">LPC1</argument>
            <argument name="location">POLYGON((103.87412301731814
            1.3587278034721435,103.86965264805309 1.3599333597062853,103.86638010009861
            1.36322700357916 27,103.86518225000309 1.3677262118682691,103.86638007125877
            1.3722254283670943,103.86965261921326 1.3755190886594102,103.87412301731814
            1.3767246531032713,103.87859341542303 1.3755190886594102,103.88186596337749
            1.3722254283670943,103.8830637846332 1.3677262118682691,103.88186593453769
            1.36322700357916 27,103.87859338658319 1.3599333597062853,103.87412301731814
            1.3587278034721435))</argument>
            <argument name="socketHost">localhost</argument>
            <argument name="socketPort">3232</argument>
        </arguments>
    </function>
</ns1:processdef>
```

Figure 6: The process configuration file of a *Licence Plate Camera* that requires the target object's license plate *and* the area of interest (that will be used in the negotiation phase). Its response is a location, a timestamp and a certainty-of-observation value.

# 7. FEATURES AND USABILITY

To be applicable for a variety of decision support scenarios, our framework is designed to require little setting up effort for different scenarios. Because each process requires the same set of basic components – the means to find other resources and the actual communication to these resources, these components (described in section 5) do not have to be altered in any way. What the components do require is a *configuration of inputs and outputs*. In other words, the inputs and outputs define a resource type.

## 7.1 Configuration

The system configuration is the key element that allows the whole system to function, since it describes the *type* of resources in the system, and their potential interconnections (i.e. the output of a process is a potential input/resource for other processes). It further describes the format in which data is delivered by specifying the parameters of requests and answers; these parameters form a sort of standard of communication between processes, which ensures coherent data interpretation.

For example, at high level, a *Licence Plate Camera* component type is defined by a license plate and the coordinates of an area of interest as inputs, and the location of the license plate as output. Figure 6 shows an example of how the configuration file of the *Licence Plate Camera* surveillance process looks like. The license plate is stored in the *LicensePlate* variable in the request for data, to be able to filter its surveillance data. The camera produces the observed location plus a value reflecting the certainty of the observation. The second value in the request (*AreaOfInterest*) can be used in the negotiation phase. Next to the variables, arguments can be passed to the process, like the position of a device and the procedure for connecting to it. Since a camera does not require any input resources, no variables of this type are shown in the example, but the configuration of input variables uses a syntax similar to that of the output variables.

## 7.2 Simulation

To test and evaluate the system, a simulator is used to mimic the real world. This simulator generates objects (vehicles in the case of the tracking scenario) that randomly

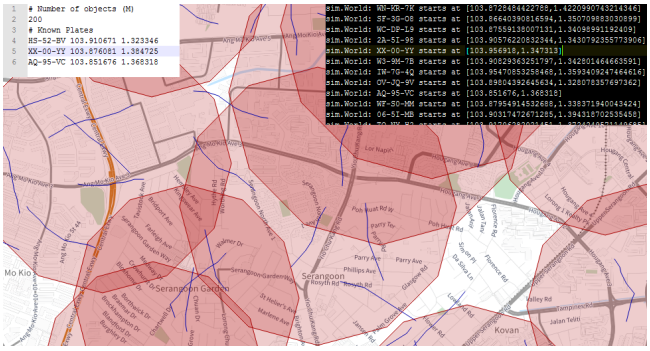

Figure 7: The simulation randomly generates $M$ objects, except for a few "known" objects (top left). The objects (the blue lines) move through the world while the sensors (the red areas) periodically check the objects in their line of sight.

Figure 8: Screenshot of the operator view, showing the predicted next position of the requested car using a heat map.

move through an area, following context information in the form of a road map.

When starting the simulator, different parameters can be set - e.g., the number of objects in the environment and a set of objects with their starting positions. For example, vehicle with license plate *XX-00-YY* should start at position *A*. Next to this required vehicle, the simulator should fill the area with other random vehicles up to a total of $N$ objects. With the use of context data (e.g. road maps, traffic information, obstruction due to construction) all $N$ objects navigate through the environment using random choices of turns.

In the context of target tracking, each sensor connects to the simulator, requests all objects within its range, and applies its own filter on this subset of objects to try and find the objects it is asked to look for, similar to how it will operate in real-life situations. Figure 7 illustrates the simulated world, using the configuration from the top left corner of the image to generate 200 objects of which only 3 have a known starting position, the rest receive a randomly generated starting position and random license plate. Figure 8 shows an example how the result of the tracking system can be used to help the operator make a decision, by showing a heat map of the predicted movements based on the observed surveillance data. Where the heat map is constructed of a set of points, each with a likelihood of the target being there, displayed with a colour indicating the likelihood as warmth (red: very likely, blue: not likely).

Figure 9: From left to right: a synthetic scenario with one junction, a more complex synthetic scenario, and a real life scenario of a small city.

For the setup of the simulator multiple information and configuration files are required: (1) files or database containing the context data, (2) the set of structured objects, and (3) some tuning parameters (e.g. number of noise objects, the frequency of updating the objects, the maximum speed of an object).

## 8. VALIDATION

For evaluation, we use the specific case-study of target tracking in urban environments and we focus on correctness. The idea for this validation is to change the environment (i.e. the road map) to a case that has known results - that is, it is either obvious or known in advance how the situation evolves and must be resolved. If correct, the simulator will match the ground truth in its behavior.

At the moment, no ground truth data is available (and generating it is another research problem in itself). Therefore, to validate the correct functionality of our simulator and system, we use synthetic scenarios. These synthetic scenarios are simplified environments - e.g., a main road with a single junction, were it splits into two side roads (left most image in Figure 9). In such cases, a full 1-to-1 comparison can be made between the expected results and those automatically calculated. When we observe that the tracker properly follows the car into the right direction, the communication between sensors, context bank and tracking algorithm is correct. As for the decision model, it should choose to close down the one side road and not both or the main road, to actually minimize the impact on normal traffic. Once these synthetic microtests are passed correctly, we further validate against realistic scenarios, which use real road maps and data (two examples are shown in Figure 9). In this case, validation is done by randomly sampling intermediate results and comparing them against the real-world and/or simulated results.

We note here that, the system we have built so far works correctly and fully automated for the simple microtests. Sample-based validation for larger synthetic cases is work in progress. Furthermore, we note that empirical validation cannot guarantee the full correctness of the system, but rather its compliance to the expected outcome for a large number of (simple) situations. For example, see Figure 10: the simulator correctly reads the data from the sensors and computes the predictions for the future position of the tracked object, but it can only take a clear decision in the situation on the left, not in the situation on the right.

We conclude that the simulator is accurate in following the targets on the map, which essentially proves that it is able to manage several processes and sensors correctly. We emphasize that this matches its goal to support decision

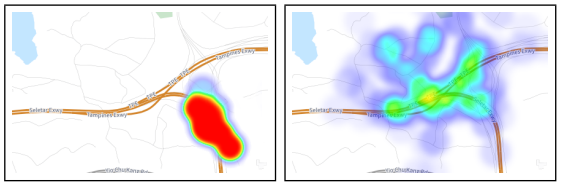

Figure 10: Left: the simulator correctly predicts the trajectory of the object (high intensity on the heat-map); a decision can be made to shut off the road before the junction. Right: the simulator detects the T-junction and predicts both rows as probable destinations. No single-road can be selected to be closed before the a new measurement is made.

making by providing the operator enough data to make an informed decision based on high-probability predictions.

## 9. PERFORMANCE ANALYSIS

For the framework to be applicable in the targeted security domains, scalability is mandatory. Therefore, we present a first attempt to analyze the scalability of the system.

Our metrics for scalability are processing time and information transfer time. Processing time is defined as the time between arrival of new information and the time an (updated) output is ready. For the transfer time, the definition is less straightforward, because it could involve more than a mere exchange of information from A to B. Thus, to transfer information in a situation where the processes are already connected, the transfer time equals the time to exchange the information (exchange cost). To transfer information when no connection exists, the transfer time includes both the time to find potential receivers and establish a connection (negotiation cost), and the actual sending of the information (exchange cost). We therefore define the transfer time as the sum of the negotiation cost and data exchange cost, where the negotiation cost for an established connection is zero.

For measuring the processing time of the system we ran experiments with varying numbers of sensors ($S$) and tracked target objects ($N$) between 1 to 30. In each experiment, we simulated the movement of 50 objects randomly through an urban environment (see the rightmost scenario in Figure 9) to provide the sensors with data. We position the sensors to cover the whole area, to ensure a known number of connections (meaning the negotiation has a known outcome namely all sensors will be connected).

Running an experiment with $S = 5, N = 5$ means that 5 objects (out of the total 50) are tracked during the whole experiment, and all 5 of the validation blocks receive information from each of the 5 sensors active in the system. Projecting this setup on the pipeline of Figure 3 we can calculate the number of transmitted messages for each new output, namely $S+4$ messages: from each sensor, the movement prediction and the context collection to the validation, in between the two validation blocks and one from validation to the action suggestion step. So in a setup of $S = 5, N = 5$ we have $5 \times (5 + 4) = 45$ messages for each iteration. For a large setup of $S = 30, N = 30$, we have 1020 messages each iteration. Based on these numbers we expect an exponential growth in communication time.

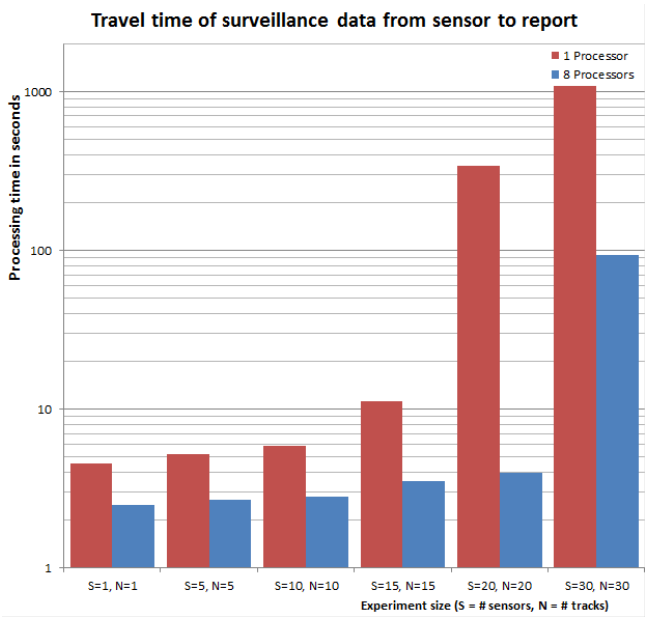


Figure 11: Processing time in seconds for target tracking – time from arrival of new surveillance data to the time a new output is ready, using 1 and 8 processors. Note the increasing number of sensors $S$ and tracks $N$, where each sensor connects to all available tracks.

For the experiments we used two hardware configurations:

| | Name | Clock Freq. | Processors |
|---|---|---|---|
| single-core | Intel Xeon CPU E5-2450 | 2.5GHz | 1 core |
| multi-core | Intel Xeon CPU E5-2450 | 2.5GHz | 8 cores |

Figure 11 shows the median processing time of our simulation over 1000 iterations.

We observe that already with a single sensor and track the *multi-core configuration* (8 processors) outperforms the *single-core configuration* (1 processor), since the different blocks in the pipeline can now run in parallel allowing, for example, a next iteration to start before the previous finished. Furthermore between $S = 1, N = 1$ and $S = 20, N = 20$ the processing time increases with only 1.5 seconds against 336.5 seconds for the *1 processor* experiment. These performance numbers demonstrate that (a) performance is challenging for a system with relatively low density of objects, tracks, and sensors, and (b) that our proposed decision support system is able to make use of system parallelism. For the $S = 30, N = 30$ experiment, both hardware configurations show a huge increase in processing time, which is not surprising given the fact that we move from 480 to 1020 messages per iterations and the number of processing blocks in this setup adds up to 151 blocks. Clearly, although our system can still scale to cope with such data-intensive, complex cases, the time-to-decision can becomes unacceptable. To address this challenge, our current work focuses on increasing system efficiency through better parallelisation strategies.

We also note that gathering the results from the $S = 30, N = 30$ scenario on the *single-core configuration* was very difficult because only 25% of the iteration finished successfully, due to the lack of computation and/or memory

resources. It is important to note that no such issues appeared when switching to the 8-core configuration. These results demonstrate *both* the importance of scalability for such decision support systems, and the ability of the proposed framework to automatically use more resources, when available.

# 10. CONCLUSION

In the security domain, many complex situations need to combine unstructured data and complex processes to effectively support the decisions of a human operator. Similar scenarios (with even larger variability and complexity) emerge today in the Internet-of-Things (IoT) domain.

In this work, we presented a generic, scalable framework for decision support in complex situations. Our contributions are: (1) identifying the functional and computational requirements of such a framework, (2) proposing a first architecture for this framework, (3) implementing a first functional prototype for semi-automated decision support, and (4) defining and testing validation scenarios for surveillance applications.

Our results so far are promising in terms of correctness and simple-case performance. Our immediate future work focuses on complex scenario implementation and testing, thus identifying and tackling the complex scalability challenges and boundaries for real situations. Furthermore, we currently investigate the applicability and the limitations of our framework for IoT applications.

# 11. REFERENCES


[1] C. Hewitt, P. Bishop and R. Steiger. *A Universal Modular ACTOR Formalism for Artificial Intelligence*, IJCAI, pp 235âĂŞ245, 1973.

[2] J. Burns, *A formal model for message passing systems*, Computer Science Department, Indiana University, 1980

[3] R. Sprangue, *A framework for the development of decision support systems*, MIS, pp 1-26, 1980

[4] G. Agha, *ACTOR: A Model of Concurrent Computation in Distributed Systems*, MIT Press series in artificial intelligence, 1986

[5] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson and P. Nordlund, *Particle Filters for Positioning, Navigation, and Tracking* IEEE Transactions on Signal Processing, 50(2):425âĂŞ437, 2002.

[6] A.Penders, G. Pavlin and M. Kamermans, *A collaborative approach to construction of complex service oriented systems*, IDC, pp 55-66, 2010.

[7] S. Tasharof, P. Dinges and R. Johnson, *Why Do Scala Developers Mix the Actor Model with Other Concurrency Models?*, ECOOP, pp 302âĂŞ326, 2013.

[8] H. Koen, P. de Villiers, G. Pavlin, A. de Waal, P. de Oude and F. Mignet, *A framework for inferring predictive distributions of rhino poaching events through causal modelling*, Fusion, 2014.

[9] R. Claessens, A. de Waal, P. de Villiers, A. Penders, G. Pavlin and Karl Tuyls, *Multi-Agent Target Tracking using Particle Filters enhanced with Context Data*, AAMAS, 2015

[10] A. Rosà, L. Chen and W. Binder, *Profiling Actor Utilization and Communication in Akka*, Erlang, pp 24-32, 2016