Reconfiguration of Large-Scale Surveillance Systems

Peter Novák and Cees Witteveen

Algorithmics Group, Faculty EEMCS
Delft University of Technology
The Netherlands
{P.Novak, C.Witteveen}@tudelft.nl

Abstract. The METIS research project aims at supporting maritime safety and security by facilitating continuous monitoring of vessels in national coastal waters and prevention of phenomena, such as vessel collisions, environmental hazard, or detection of malicious intents, such as smuggling. Surveillance systems, such as METIS, typically comprise a number of heterogeneous information sources and information aggregators. Among the main problems of their deployment lies *scalability* of such systems with respect to a potentially large number of monitored entities. One of the solutions to the problem is continuous and timely adaptation and reconfiguration of the system according to the changing environment it operates in. At any given timepoint, the system should use only a minimal set of information sources and aggregators needed to facilitate cost-effective early detection of indicators of interest.
Here we describe the METIS system prototype and introduce a theoretical framework for modelling scalable information-aggregation systems. We model information-aggregation systems as networks of inter-dependent reasoning agents, each representing a mechanism for justification/refutation of a conclusion derived by the agent. The proposed continuous reconfiguration algorithm relies on standard results from abstract argumentation and corresponds to computation of a grounded extension of the argumentation framework associated with the system.

1 Introduction

The METIS project [4, 6] studies techniques supporting development of large-scale dependable systems of systems which aggregate multiple sources of information, analyse them, compute risk factors and deliver assessments to system operators. In this paper we introduce the METIS project's prototype application, which applies the developed concepts to the domain of maritime security and aims to provide advanced situation awareness capabilities for monitoring maritime traffic in national coastal waters. By 'Systems-of-systems' we mean large-scale integrated systems that are heterogeneous and independently operable on their own, but are networked together for a common goal [7]. One of the prominent problems in development of such systems is their *scalability*. Our focus here is on supporting scalability of the system by means of continuous *reconfiguration*, i.e., adaptation to changes in its environment.

The METIS system is a large-scale surveillance system operating in a mixed physical and software environment. It comprises a number of cooperative agents serving as information sources and aggregators. Typically, these would be either situated physical

agents, such as cameras, satellites or human patrols, or software components interfacing various public, or proprietary databases, web resources, etc.

In the implemented prototype scenario, METIS aims at detection of ships suspected of smuggling illegal contraband during their approach to the port under surveillance. For every vessel in the zone of its interest, the system accesses the various information sources and subsequently processes the extracted information so as to finally identify vessels which require operator's attention. The available sources provide information about the ships, including their identifications, crew, ports-of-call, various physical characteristics, possibly even digest of news articles reporting on events involving the vessel, or the crew. Quite often, such information would yield inconsistent, or even contradictory information, which needs to be cross-validated and processed in order to infer the most likely values. The resulting information is aggregated by a hierarchy of information aggregators so that the system is ultimately able to determine whether a particular vessel should be considered a smuggling suspect, or it is able to justify that it is innocuous given the available information. In the prototype scenario, the individual aggregators are represented by various information-fusion components operating over a shared data warehouse, but could include also external agents, such as human experts.

METIS should be deployable both on land, as well as on board of independently operating ships. As a consequence, querying individual information sources and subsequent information aggregation could incur non-negligible financial and computational costs. While accessing a publicly available Internet resource via a fixed broadband connection can be relatively cheap, the bandwidth of satellite communication links used on board of maritime vessels is limited and data transfers incur external costs too. Similarly, accessing proprietary industrial databases, or utilisation of physical agents, such as aerial drones, imaging satellites, etc. can incur rather significant costs to the system's operation. Hence, using all available information sources and information fusion components is not always feasible and in turn one of the problems central to development of such a large-scale surveillance multi-agent system is their scalability.

The problem of configuration and dynamic reconfiguration according to the current system's needs can be thus formulated as follows: *Which information sources and aggregators should be active over time so as to maximize the likelihood of early detection of malicious intents in the most cost-efficient manner?*

Here, we propose an approach to (re-)configuration of large-scale information aggregation systems by modelling the interactions between the individual components in terms of an *argumentation framework* [2]. After introducing the basic concepts (Section 2), in Section 3, we present the problems of configuration and reconfiguration of information-aggregation systems to account for changes in their environments. Subsequently, in Section 4, we show that suitable system configurations correspond to the concept of grounded extensions of an associated argumentation framework. The solution concept is closely related to the well-founded semantics of logic programs, so the relationship opens the door for further study of reconfiguration in relation to standard results in logic programming. A discussion of on-going and future work along the presented line of research concludes the paper. Throughout the paper, in a series of expositions, we describe the relevant parts of the METIS system and identify a class of relevant solution concepts.

METIS 1 *In the prototype scenario,* METIS *should continuously monitor vessels in the coastal waters in the* Dutch Exclusive Economic Zone*, source information about them and process it, so as to finally identify vessels which are suspect of smuggling. Upon detection of a suspicion, the system should notify the user, a* Netherlands Coastguard *officer, who then decides on the subsequent course of action. To put the scenario in perspective, note that the monitored area covers more than 63.000 km^2 and typically contains around 3-4.000 vessels at any given moment in time.*

In the system exposure we consider the following simplified fragment of the prototype scenario: Information-sources available to the system comprise a local copy of IHS Fairplay [5] *database and web-portals of* MyShip.com [11]*,* MarineTraffic.com [10] *and its* Ports of Call*. There are also three physical sensors: a human* coast-guard patrol *in the field, a receiver for* Automatic Identification System *(AIS)* [1] *messages, and a* radar *providing kinematic signatures of vessel tracks as interpreted from the readings of the detected spot positions of the vessel over time. Every vessel from a certain size is required to be equipped with an AIS transmitter and regularly broadcast its identity, type, ports of call, etc. Besides cross-validation and and probabilistic inference over the received data, the individual information-processing components also derive meta-information about quality, certainty and trust of the aggregated information.*

2 Preliminaries

An instance of a multi-agent surveillance system such as METIS, comprises a set of *information processing* agents and a *shared database*. *Information source* agents operate in a *dynamic environment* and feed a shared *data store* which is further processed by a set of *information aggregators* agents. The system's objective is to determine the truth value of a set of *distinguished indicators*, information elements corresponding to some non-trivially observable properties of the monitored entities, such as whether a vessel is a smuggling suspect. Below, we introduce the formal framework for modelling information-aggregation systems, together with the related terminology and notation.

2.1 Information-aggregation system

We model an abstract information-aggregation system as a tuple $\mathcal{S} = (\mathcal{A}, \mathcal{D}, cost)$ comprising a finite set of information-processing agents, a database schema and a cost function respectively. A shared data store of the system is represented by a 3-valued database schema $\mathcal{D}$ comprising a finite set of propositional variables over the domain $Dom = \{\top, \bot, \varnothing\}$ representing the truth values *true*, *false*, and *unknown* respectively. In practice, Dom could include an arbitrary number of distinct crisp values and the METIS system exposure indeed assumes an extended domain of the database schema. Without loss of generality, we also do not distinguish between different interpretations of the unknown value $\varnothing$: *no information* and *value existent, but unknown* [8]. A database snapshot $D : \mathcal{D} \to Dom$ of the schema $\mathcal{D}$ at a given timepoint is a ground interpretation of variables of $\mathcal{D}$. That is, each variable of $\mathcal{D}$ takes a truth value from the domain Dom. $D|x$ denotes the value of the variable x in D. $D_\varnothing$ denotes a database snapshot with all

variables valued as unknown, i.e., for all $x \in \mathcal{D}$, we have $D_{\varnothing}|x = \varnothing$. For convenience, we use the term database snapshot interchangeably with the term database.

The information processing agents $\mathcal{A} = \{A_1, \ldots, A_n\}$ of the system are modelled as function objects over interpretations of the schema $\mathcal{D}$, formally $A_i : \mathcal{D} \times Dom \to \mathcal{D} \times Dom$ for each $A \in \mathcal{A}$. That is, given a database snapshot D, an information-processing agent A takes as an input a set of D-valuations of database variables $in_A \subseteq \mathcal{D}$ and produces a set of new valuations for database variables $out_A \subseteq \mathcal{D}$. $D|in_A$ and $D|out_A$ respectively denote value assignments to variables of in_A and out_A corresponding to those in the snapshot D. The sets $D|in_A$ and $D|out_A$ can be seen as partial interpretations of the schema $\mathcal{D}$. Given two snapshots D and D', we denote $A(D|in_A) = D'|out_A$ relying on agents as partial functions over database snapshots. We model information-source agents as standard information-processing agents with an empty set of input variables $in = \emptyset$ and a non-empty set of output variables $out \neq \emptyset$.

The cost function $cost : \mathcal{A} \to \mathbb{R}^+$ models the costs involved in a single computation run of an agent. Without loss of generality we assume that the computation of valuations of the output variables cannot be disentangled and must be carried out as an atomic operation. Informally, the cost of executing an information source agent corresponds to the aggregate cost of sensing its input variable in the environment. Whenever the cost function is irrelevant in the given context, we simply write $\mathcal{S} = (\mathcal{A}, \mathcal{D})$.

A configuration $C \subseteq \mathcal{A}$ of a system $\mathcal{S} = (\mathcal{A}, \mathcal{D}, cost)$ is a set of information processing agents *active* in $\mathcal{S}$ in a given timepoint. Notation for input and output variables of an agent naturally extends to configurations, that is $in_C = \bigcup_{A \in C} in_A$ and $out_C = \bigcup_{A \in C} out_A$. Assuming a single execution of each agent in C, the $cost$ function straightforwardly extends to configurations too: $cost(C) = \sum_{A \in C} cost(A)$.

Given a configuration $C \subseteq \mathcal{A}$ of a system $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ and a database snapshot D of the schema $\mathcal{D}$, we say that a database D' is an *update* of D by C iff for each variable $x \in \mathcal{D}$, such that $D|x \neq D'|x$, there exists an agent $A \in C$, such that $D'|x = A(D|in_A)|x$. That is, each variable modified in the update D' w.r.t. its original value in D, is a result of a computation of some agent from the configuration C. We say that a configuration C is *supported* by a database snapshot D iff for all agents $A \in C$ we have both $A(D|in_A) \subseteq D$, as well as $A(D|out_A) \subseteq D$. That is, the information processing performed by each agent A of the configuration C is reflected in the snapshot D.

$C(D) = D'$ denotes an update D' of D by a configuration C. Note, not all of the outcomes produced by all agents of C need to be reflected in the database update. Alternatively, we say that D' is an update of D by a partial database D_u iff whenever $D|x \neq D_u|x \neq \varnothing$, we have that $D'|x = D_u|x$ and $D'|x = D|x$ otherwise, and we also denote $D' = D \oplus D_u$. We model evolution of a system $\mathcal{S}$ under a configuration $C \subseteq \mathcal{A}$ as a (possibly infinite) sequence of database snapshots $\lambda_{\mathcal{D}} = D_0, \ldots, D_k, \ldots$, such that each $D_{i+1} = C(D_i)$ is an update of D_i for all $i \in \mathbb{N}_0$. Such a $\lambda_{\mathcal{D}}$ is called a *C-evolution* of $\mathcal{S}$ from D_0 on. Finally, note that given a configuration C which is supported by a database snapshot D, every update $C(D)$ equals D. In that case, we say that D is *stable* w.r.t. C.

Evolution of a system strongly depends on both the nature of the active configuration, as well as the particular order in which the agents of the configuration work over the database. We say that a configuration $C \subseteq \mathcal{A}$ of a system $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ is *normal*

agent	in	out	*cost*
AIS	$\emptyset$	*aisID**, *aisType*$'$	10
FairPlay	$\emptyset$	*fpID**	10
MyShip	$\emptyset$	*myShipID*†	200
MarineTraffic	$\emptyset$	*mtID*†	300
MarineTraffPorts	$\emptyset$	*portCalls*‡	1000
Radar	$\emptyset$	*track*$'$	2000
Patrol	$\emptyset$	*isSpoofingID*‡	9500
TrackAnalyser	marked $'$	*vesselType*‡	1000
CheckDefault	marked $*$	*isSuspectID*†	200
CheckSpoofing	marked $\dagger$	*isSpoofingID*‡	800
checkSmuggling	marked $\ddagger$	*isSmuggling*	2000

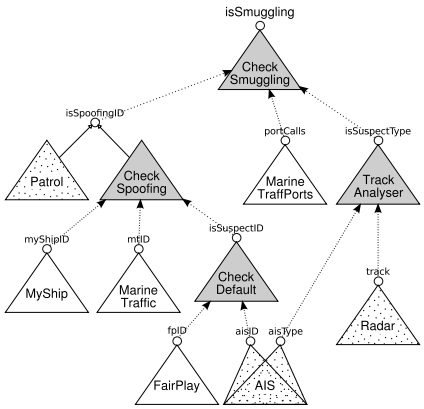

Fig. 1. METIS system agents (left) and their interdependencies (right).

iff all C-evolutions of $\mathcal{S}$ from every database snapshot D_0 on, eventually *stabilise*, i.e., reach the same stable state regardless of the order of execution of the individual agents in C. More formally, there is a unique database snapshot $C^*(D_0)$ of the schema $\mathcal{D}$, so that for all C-evolutions $\lambda_\mathcal{D} = D_0, \ldots, D_k, \ldots$ of $\mathcal{S}$ from D_0 on, there is an index $k_{\lambda_\mathcal{D}} \geq 0$, such that for all $i \geq k_{\lambda_\mathcal{D}}$, $D_i = D_{k_{\lambda_\mathcal{D}}} = C^*(D_0)$ and C is supported by D_k.

METIS 2 (system structure) *In the prototype scenario (Figure 1),* METIS *features 7 information-source agents (white), including 3 physical sensors (dotted), and three non-trivial information-aggregation agents (grey). The costs of accessing the information sources are only illustrative and estimate the communication bandwidth to access the databases, or the cost of querying the physical sensors. The costlier databases tend to provide richer information about vessels. The cost associated with an information-aggregation agent roughly estimate the computational costs of its execution. In the figure, the dotted lines indicate input-output dependency of information-aggregation agents, while the solid line arrows indicate merely that the agent derives a given variable, otherwise indicated by putting the variable bullet on top of the corresponding agent triangle.*

The CheckDefault *aggregator consults the local physical sensor and cross-validates the self-transmitted vessel identity with those listed in the* IHS FairPlay *database. Upon a failure to match the identities of the vessel, the system performs a deeper check of the vessel's identity (*CheckSpoofing*) in order to determine whether it does not actively spoofing it, that is whether it actively lies about its identity. The physical sensor* Patrol *involves a coast-guard patrol either physically checking the identity of the vessel, or possibly sending an unmanned aircraft to perform the task. Similarly to* CheckSpoofing *information-aggregator, the* Patrol *information source is capable to determine whether the vessel is actively spoofing it's identity. It is of course possible that the two inferred valuations of the* isSpoofingID *variable do not match and the conflict needs to be resolved.*

Should the system indeed conclude that the vessel is spoofing its identity, it escalates to the highest-level information-aggregator CheckSmuggling *consulting the*

most expensive information sources and performing the deepest analysis of the vessel's background so as to assess its potential involvement in smuggling. The TrackAnalyser *processor matches the vessel's kinematic track signature from the* Radar *sensor to the vessel type retrieved from* AIS*. Should the vessel turn out to be a suspect smuggler according to the* METIS*'s analysis, the valuation of* isSmuggling *information element is communicated to the system operator via a GUI warning. Note, all the involved agents assume a domain of the underlying database extended with enumerations of possible identities, etc. and can also produce the unknown valuation $\varnothing$ for each of their output variables.*

2.2 Environment

An information-aggregation system, such as METIS, is situated in a dynamic environment which changes over time. It reads values from it, monitors it, and derives non-trivial information on the basis of the collected evidence. We model an environment as a database schema $\mathcal{E}$ over crisp truth values $\{\top, \bot\}$.

A system $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ can be embedded in an environment $\mathcal{E}$ when the two database schemas coincide in exactly the variables produced by the information-source agents of $\mathcal{S}$. That is, each variable $x \in out_A$ of an agent $A \in \mathcal{A}$ with $in_A = \emptyset$ is included in the environment too, i.e., $x \in \mathcal{E} \cap \mathcal{D}$ and we denote $\mathcal{D}^{\mathcal{E}}_{in} = \mathcal{E} \cap \mathcal{D}$. A variable $x \in \mathcal{D}^{\mathcal{E}}_{in}$ in a database snapshot D of $\mathcal{S}$ *reflects* the state of the environment E iff $D|x \neq \varnothing$ implies $D|x = E|x$. We say that the system $\mathcal{S}$ is *embedded* in E iff computations of all the information-source agents reflect the state of the environment. That is, for all $A \in \mathcal{A}$ with $in_A = \emptyset$ all variables from out_A in the snapshot $A(D)$ reflect E.

The dynamics of the environment is captured by its evolution over time modelled as a (possibly infinite) sequence of database snapshots $\lambda_{\mathcal{E}} = E_0, \ldots, E_k, \ldots$ To ensure correspondence between an evolution $\lambda_{\mathcal{E}}$ of the environment $\mathcal{E}$ and an evolution $\lambda_{\mathcal{D}} = D_0, \ldots, D_l, \ldots$ of a system $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ embedded in $\mathcal{E}$, we require that there exists a sequence of indices $i_0, \ldots, i_m, \ldots \in \mathbb{N}_0$, such that the variables from $\mathcal{D}^{\mathcal{E}}_{in}$ in D_i with $i \in i_j \ldots (i_{j+1} - 1)$ reflect the environment state E_j for $j \geq 0$. That is, at every timepoint, the system is embedded in the current state of the environment.

METIS 3 (evolution example) *A configuration capable to produce the system evolution depicted in Figure 2 could include the agents* AIS*,* FairPlay*,* CheckDefault*,* Radar *and* TrackAnalyser *executed subsequently in that order up to the database snapshot D_4. In the 5th evolution step, the* AIS *agent would produce an unknown valuation $\varnothing$ for the* aisID *variable due to a failure to retrieve a crisp information from the environment (e.g., due to a failure of the vessel's AIS transmitter). Subsequently the* CheckDefault *agent would have to produce $\varnothing$ also for the* isSuspectID *variable too as one of its inputs is $\varnothing$. The total cost of execution of this configuration would be 3420. The environment of the system evolves in a sequence E_0, E_1, E_2 and its changes are reflected in the evolution of the system's database snapshots.*

3 Configuration and reconfiguration problems

Assessments of a surveillance information-aggregation systems like METIS could have real-world repercussions. For instance, after deriving that a vessel could be a smuggling

$\mathbf{D_0}, \mathbf{E_0}$	$\mathbf{D_1}$	$\mathbf{D_2}$	$\mathbf{D_3}, \mathbf{E_1}$	$\mathbf{D_4}$	$\mathbf{D_5}, \mathbf{E_2}$	$\mathbf{D_6}$
aisID$^{\mathcal{E}}$	*aisID*	*aisID*	*aisID*$^{\mathcal{E}}$	*aisID*		
aisType$^{\mathcal{E}}$	*aisType*	*aisType*	*aisType*$^{\mathcal{E}}$	*aisType*	*aisType*$^{\mathcal{E}}$	*aisType*
	fpID	*fpID*	*fpID*	*fpID*	*fpID*	*fpID*
		isSuspectID	*isSuspectID*	*isSuspectID*	*isSuspectID*	
			track$^{\mathcal{E}}$	*track*	*track*$^{\mathcal{E}}$	*track*
				isSuspectType	*isSuspectType*	*isSuspectType*

Fig. 2. An example evolution of the METIS system database. Only variables valued $\top$ are listed. The variables marked $\mathcal{E}$ are read from the corresponding environment update.

suspect, a warning would be indicated to the operator, who might then consider contacting the vessel himself, possibly even sending a patrol to the location. Such actions, however, need to be *justified* in the operational scenario. In consequence, any crisp conclusion computed by the system must be explainable and defensible by inspecting the structure of inferences from basic evidence in the environment. In turn, we are interested in system configurations, which can either crisply answer distinguished queries, such as suspicion of smuggling, or, if that is not possible, the operator needs to be sure that there is no such configuration given the current state of the environment and the system's implementation. In the following, we implicitly assumes that the system is embedded in an environment state reflected in its current (initial) database snapshot.

Problem 1 (configuration problem). Given a tuple $\mathfrak{C} = (\mathcal{S}, \phi, D)$, with $\mathcal{S} = (\mathcal{A}, \mathcal{D}, cost)$ being an information-aggregation system, $\phi \in \mathcal{D}$ a query variable, and D being an initial snapshot of $\mathcal{D}$, the *information-aggregation system configuration problem* is to find a normal configuration C, a solution to $\mathfrak{C}$, such that all evolutions of $\mathcal{S}$ rooted in D stabilise in the snapshot $C^*(D)$ and C satisfies the following:

1. $\phi \in out_C$, i.e., C contains at least one agent $A \in C$ capable to derive ϕ. The resulting *query solution* is a valuation $C^*(D)|\phi$ computed by the configuration C;
2. for each variable $x \in in_C$, we have $x \in out_C$ and $C^*(D)|x \neq \varnothing$; and finally
3. there is no configuration C' with $C \subset C'$ satisfying 1 and 2, such that $C'^*(D)|\phi \neq C^*(D)|\phi$. In that sense, C is maximal.

We say that a configuration C is an *optimal* solution to the reconfiguration problem $\mathfrak{C}$ iff $cost(C)$ is minimal among the solutions of $\mathfrak{C}$.

Condition 1 of the definition above stipulates that the solution configuration indeed provides a valuation of the query, although this can still be valued as unknown $\varnothing$. Condition 2 formalizes the intuition that the query solution can be traced back to the evidence from the environment and computations of a series of crisp variable valuations by the individual agents of the system, that is a justification for the query solution. While theoretically it would acceptable to base computations of a crisp conclusions on unknown valuations of input variables for the interpretation of $\varnothing$ *value existent, but unknown*, it wouldn't be so for inferences based on *no information* valuations. In the former case, the interpretation would behave rather as a kind of a crisp valuation. Consequently, without loss of generality we assume interpretation of $\varnothing$ to equal *no information*. Finally, condition 3 ensures that there is no doubt about the computed query solution.

A solution to configuration problem does not always exist. Consider for instance a system including two agents deriving conflicting values for the same variable, or cyclically dependent agents. In such situations, the system evolution could oscillate and never stabilise. In Section 4 we identify a class of information-aggregation systems for which existence of a solution is always ensured.

Through information-source agents, a dynamic environment serves as the main driver of change within the system. Situating the configuration problem into a changing environment, repeated configuration becomes a means for continuous adaptation of the system to the updates coming from its environment.

Problem 2 (reconfiguration problem). Given a tuple $\mathfrak{R} = (\lambda_{\mathcal{E}}, \mathcal{S}, \phi)$, where $\lambda_{\mathcal{E}} = E_0, \ldots, E_k, \ldots$ is an evolution of an environment $\mathcal{E}$, $\mathcal{S} = (\mathcal{A}, \mathcal{D}, cost)$ is a information aggregation system embedded in $\mathcal{E}$, and $\phi \in \mathcal{D}$ is a query variable, the *information-aggregation reconfiguration problem* is a search for a sequence of configurations $C_0, \ldots, C_l, \ldots$, a solution to $\mathfrak{R}$, such that each C_i is a solution to the configuration problem $\mathfrak{C}_i = (\mathcal{S}, \phi, D_i)$ for $i > 0$, where $D_i = C^*_{i-1}(D_{i-1}) \oplus E_i|\mathcal{D}^{\mathcal{E}}_{in}$ and $D_0 = D_{\varnothing} \oplus E_0|\mathcal{D}^{\mathcal{E}}_{in}$. We say that a sequence of configurations $C_0, \ldots, C_l, \ldots$ is a *weak solution* to $\mathfrak{R}$, iff C_i is a solution to $\mathfrak{C}_i = (\mathcal{S}, \phi, D_i)$ if it exists and can be arbitrary otherwise.

Informally, a reconfiguration problem solution is a sequence of configurations producing a database evolution reflecting the changes of the system's environment. The sequence of configurations in a weak solution to the reconfiguration problem captures the intuition that the system tries its best to compute a query solution upon each environment update, which, however, not always exists.

METIS 4 (configuration) *Consider the* METIS *prototype scenario introduced in the previous expositions. An example configuration problem could be* $\mathfrak{C} = (\mathcal{S}_{\text{METIS}}, isSmuggling, D_3)$. *As stated, there is no solution to* $\mathfrak{C}$ *as it is not possible to determine whether the vessel is possibly spoofing it's identity (isSpoofingID) and in turn also whether it is a smuggling suspect (isSmuggling). A solution would exist for a configuration problem over a database including crisp valuations for all the variables produced by information-source agents. Furthermore, the output of the* Patrol *agent would have to match that of* CheckSpoofing *aggregator. In that case, the solution to* $\mathfrak{C}$ *would comprise all the agents of the system. There would also exist solutions for configuration problems over databases in which the* Patrol *information-source produces an unknown for the isSpoofingID variable, but* CheckSpoofing *aggregator derives a crisp valuation for it, or vice versa.*

4 Solving configuration and reconfiguration problems

The individual agents of an information-aggregation system perform inference over valuations of their input variables, premises, and thus provide support to the output variables, conclusions. In turn, Dung's theory of abstract argumentation [2] provides a natural model of computation of information-aggregation systems. Here, we propose an approach to solving (re-)configuration problems rooted in sceptical semantics of argumentation. The terminology introduced below is adapted from [2].

Let $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ be a system and D be a database snapshot of $\mathcal{D}$. We construct a *configuration argumentation framework* $CAF = \langle \mathcal{A}, \prec \rangle$ associated with $\mathcal{S}$ over $\mathcal{D}$.

Arguments correspond to information-processing agents $\mathcal{A}$ and embody a set of interrelations among variables of the schema $\mathcal{D}$. The input variables in_A provide the basis for inferring the conclusions out_A of the argument $A \in \mathcal{A}$. We say that an argument is *valid* w.r.t. a database snapshot D iff $A(D|in_A) \subseteq D$ and for all variables $x \in in_A$, we have $D|x \neq \varnothing$. Informally, a valid argument is supported by a given database snapshot in that the input/output characteristics of the internal computation of the agent is truthfully reflected in the database. From now on, we will use the notions of an argument and an agent interchangeably according to the context.

We say that valid argument $A \in \mathcal{A}$ *attacks* another argument $A' \in \mathcal{A}$ denoted $A' \prec A$, on a variable $x \in out_A \cap out_{A'}$ w.r.t. a given database snapshot D iff $A(D|in_A)|x \neq \varnothing$ and $A(D|in_A)|x \neq A'(D|in_{A'})|x$. That is, the agent A derives a crisp valuation for x which disagrees with the one derived by the agent A'. We also say that A is a *counter-argument* to A', or that A is *controversial*. Finally, an argument $A \in \mathcal{A}$ attacks a set of arguments $C \subseteq \mathcal{A}$ iff there exists $A' \in C$ attacked by A.

Note, the attack relation is defined only for valid arguments supporting their conclusions by crisp valuation of their input. The conclusion, however, does not necessarily need to be crisp itself. Also, the attack relation is not symmetric in that a valid argument supporting a crisp conclusion can attack an argument providing unknown valuation to the same conclusion, but not *vice versa*.

Consider a fixed argumentation framework CAF associated with a system $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ over a database D. A configuration C is said to be *conflict-free* if there are no agents $A, B \in C$, such that A attacks B w.r.t. CAF. A valid argument $A \in \mathcal{A}$ (agent) is *acceptable* to C iff for each $A' \in \mathcal{A}$ in the case A' attacks A, then there exists another argument A'' in C, such that A' is attacked by A'' all w.r.t. the database snapshot D.

In security-related information-aggregation systems, such as METIS, computed assessments need to be justified in order to preserve presumption of innocence of the monitored entities. That is, the resulting crisp valuation must be traceable to and justifiable by the evidence coming from the environment. Reasoning of such a systems is sceptical in that only conclusions which the system is sure about can be inferred, given the environment evidence and the system's design. The notion of a grounded extension of an argumentation framework based on a fix-point semantics captures this intuition.

A *grounded extension* of an argumentation framework $CAF = \langle \mathcal{A}, \prec \rangle$, denoted GE_{CAF}, is the least fix-point of its *characteristic function* $F_{CAF} : 2^{\mathcal{A}} \to 2^{\mathcal{A}}$ defined as $F_{CAF}(C) = \{A \mid A \in \mathcal{A} \text{ is acceptable to } C\}$. GE_{CAF} is *admissible*, i.e., all agents in GE_{CAF} are also acceptable to GE_{CAF} over D, and *complete*, i.e., all agents which are acceptable to GE_{CAF}, also belong to it.

A grounded extension of $CAF_{\mathfrak{C}}$ always exists and F_{CAF} is monotonous with respect to set inclusion. In general, an argumentation framework can have multiple grounded extensions, a property undesirable to security-related systems, where assessments should be unambiguous. Dung in [2] shows that argumentation frameworks without infinite chains of arguments $A_1, \ldots, A_n, \ldots$, such that for each i, A_{i+1} attacks A_i, have a unique grounded extension. A way to ensure that property is to consider only *stratified* systems. That is those, for which there exists a stratification, a decomposition into a

sequence of *strata* (layers) $\mathfrak{A} = \mathfrak{A}_0, \ldots, \mathfrak{A}_k$, where $\mathfrak{A}_0 = \{A \in \mathcal{A} \mid in_A = \emptyset\}$ and $\mathfrak{A}_i = \{A \in \mathcal{A} \mid in_A \subseteq out_{\bigcup_{j=1..i-1} \mathfrak{A}_j}\}$ for all $i = 1..k$. We say that $\mathfrak{A}$ is the *most compact* stratification of $\mathcal{S}$ iff all agents belong the lowest possible layer of $\mathfrak{A}$. Formally, for all stratifications $\mathfrak{A}'$ of $\mathcal{S}$, $A \in \mathfrak{A}_i$ implies $A \in \mathfrak{A}'_j$ with $j \geq i$.

The following proposition establishes the correspondence between solutions to configuration problems for stratified systems and grounded extensions of their configuration argumentation frameworks.

Proposition 1. *Let $\mathfrak{C} = (\mathcal{S}, \phi, D)$ be a configuration problem with a stratified system $\mathcal{S}$. Let $C = GE_{\mathfrak{C}}$ be the grounded extension of $CAF_{\mathfrak{C}}$, an argumentation framework associated with $\mathcal{S}$ over the database $C^*(D)$. If $\phi \in out_C$, then C is a solution to $\mathfrak{C}$.*

Proof. Let $\mathfrak{A}$ be the most compact stratification of $\mathcal{S}$ and let F^i_{CAF} denote the i-th iteration of F_{CAF}, with $F^0_{CAF} = F_{CAF}(\emptyset)$. Firstly, we show that iteration of F_{CAF} preserves the condition 2 of Problem 1, namely that for each $x \in in_{F^i_{CAF}}$, also $x \in out_{F^i_{CAF}}$ and $C^*(D)|x \neq \varnothing$. The proof proceeds by induction on layers of $\mathcal{S}$.
Initial step: By necessity, $\mathfrak{A}_0$ includes only information-source agents. In turn, $F^0_{CAF} \subseteq \mathfrak{A}_0$ excludes only those agents, for which there exists a counter-argument (agent) in $\mathfrak{A}_0$. Since $in_{F^0_{CAF}} = \emptyset$, the property is trivially satisfied.
Induction step: Let the property be satisfied for all F^i_{CAF} with $i = 0..k$. $F^{k+1}_{CAF} = F_{CAF}(F^k_{CAF})$. Firstly, observe that $F^i_{CAF} \setminus F^{i-1}_{CAF} \subseteq \mathfrak{A}_i$ for every i. If that were not the case, there would be an agent A from a higher layer, input of which is computed by agents in the lower layers, or it would belong to a lower stratum. The former cannot be the case since $\mathfrak{A}$ is the most compact stratification of $\mathcal{S}$, since each agent is at its lowest stratum possible. Similarly, the latter can't happen either, since it would be considered for acceptance already in earlier iterations of F_{CAF}. Now either $F^k_{CAF} = F^{k-1}_{CAF}$, the fix-point, and the property is trivially satisfied, or $F^k_{CAF} = F^{k-1}_{CAF} \cup C^k$. In that case, we need to show that for each $A \in C^k$, $in_A \subseteq out_{F^{k-1}_{CAF}}$ and all valuations $C^*(D)|in_A$ are crisp. But since $C^k \subseteq \mathfrak{A}_k$, due to the definition of stratification we have $in_A \subseteq out_{F^{k-1}_{CAF}}$. Finally, each $A \in C^k$ is acceptable to F^{k-1}_{CAF}, hence it also must be valid and in turn all its input variables are crisp.

To conclude, C is a fix-point of F_{CAF}, hence the maximality condition 3 in Problem 1 is straightforwardly satisfied too. Finally, due to the antecedent of Proposition we have that the query ϕ is included in C, hence C *is a solution of* $\mathfrak{C}$.

Proposition 1 can be applied to static databases only. Note, execution of agents considered for acceptance to a candidate solution does not modify the database fragment computed in previous iterations, which also remains stable in further computation. In turn, a naive configuration algorithm utilising Proposition 1 would iteratively proceed in three steps. In the i-th iteration it would i) execute all the agents from stratum $\mathfrak{A}_i$ of the most compact stratification of $\mathcal{S}$, ii) select the non-controversial ones, and finally iii) add them to the candidate solution. To ensure non-validity of arguments from higher strata that utilise controversial inputs derived in this iteration, these should be set to $\varnothing$.

The naive algorithm, while correctly computing a solution to a given configuration problem, is rather inefficient in terms of the overall run-time cost. It targets computation of a grounded extension of the whole framework, instead of only answering the query of

the given configuration problem. Firstly, in the initial iteration the algorithm considers and executes all information-source agents. Besides that, it potentially executes also information-processing agents, which do not contribute to answering the query. In both cases it incurs unnecessary run-time cost. In fact, only arguments relevant to derivation of the configuration problem query need to be considered.

Let $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ be a stratified system and $\phi \in \mathcal{D}$ be a query. The *agents relevant to* ϕ include $\mathcal{A}_\phi(\emptyset) = \{A \in \mathcal{A} \mid \phi \in out_A\}$. Given a set of agents C relevant to ϕ, all the agents computing the input for those in C are relevant to ϕ too, i.e., $\mathcal{A}_\phi(C) = \{A \in \mathcal{A} \mid out_A \subseteq in_C\}$. The set of all agents relevant to ϕ is the (unique) fix-point of $\mathcal{A}_\phi(\emptyset)$ denoted $\mathcal{A}^*_\phi$. The following proposition formalizes the intuition.

Proposition 2. *Let $\mathfrak{C} = (\mathcal{S}, \phi, D)$, $CAF_\mathfrak{C}$ and $GE_\mathfrak{C}$ be as in Proposition 1. If $\phi \in out_C$, then $C \cap \mathcal{A}^*_\phi$ is the minimal optimal solution to $\mathfrak{C}$.*

Furthermore, the naive algorithm does not terminate early enough, but rather computes the grounded extension to its full extent, despite the fact that in the course of its computation it might turn out that the query is either derived in a justified manner, or that its computation is hopeless. The former is relatively easy to detect. After all the agents relevant to ϕ were considered for inclusion to the candidate solution, further computation will consider only irrelevant arguments. To detect the latter case, we need to closely inspect the current candidate solution with respect to the interdependencies among the agents of the system. Given a configuration C, let $\overline{\mathcal{A}_\phi}^*(C)$ be the fix-point of the operator $\overline{\mathcal{A}_\phi}(C) = C \cup \{A \in \mathcal{A}_\phi \mid in_A \subseteq out_C \text{ and } in_A \neq \emptyset\}$. $\overline{\mathcal{A}_\phi}^*$ is complementary to $\mathcal{A}_\phi$ in that given a configuration C, it collects all agents dependent solely on the output of C. Consequently, $\overline{\mathcal{A}_\phi}^*(F_{CAF}(C))$ contains C, together with all the arguments which can be still eventually considered for accepting to the candidate solution in future iterations of F_{CAF}. In the case $\phi \in out_{\overline{\mathcal{A}_\phi}^*(C)}$ ceases to hold during computation, the algorithm can terminate, since none of the arguments capable to compute the query solution can be added to C in the future. The following proposition formalizes the relationship between the operator and the structure of the grounded extension.

Proposition 3. *Let $\mathfrak{C} = (\mathcal{S}, \phi, D)$, $CAF_\mathfrak{C}$ and $GE_\mathfrak{C}$ be as in Proposition 1. We have, $\phi \in out_{GE_\mathfrak{C}}$ if and only if $\phi \in out_{\overline{\mathcal{A}_\phi}^*(F_{CAF}(C))}$ for every $C \subseteq GE_\mathfrak{C}$.*

Finally, the naive algorithm considers arguments for accepting to the candidate solution in sets, subsets of system layers. Considering arguments for acceptance one by one would facilitate even earlier detection of hopeless computations and thus further reduction of run-time costs. It could even consider arguments across strata, however, in that case, in line with the sceptical inference strategy, the accepted arguments can only use input variables which are a part of the already stabilised fragment of the database. An alternative definition of (safe) acceptability of an argument A a conflict-free configuration C is when all its input variables are i) crisply valued, ii) already derived by C, and iii) there are no argument outside of C which can potentially threat the valuations of its input variables. More formally, an argument A is *safely acceptable* to a conflict-free configuration C iff i) there is no $x \in in_A$ with $D|x = \varnothing$, ii) $in_A \subseteq out_C$, and iii) there is no $A' \in \mathcal{A} \setminus C$, such that $in_A \cap out_{A'} \neq \emptyset$. Evaluation of this alternative definition of acceptability does not require execution of the agent A and thus can be used in the context of an evolving database, as is the case in METIS.

Algorithm 1 Algorithm computing weak-solutions to a reconfiguration problem

Require: $\mathfrak{R} = (\lambda_\mathcal{E}, \mathcal{S}, \phi)$ with environment evolution $\lambda_\mathcal{E} = E_0, \ldots, E_k, \ldots$, a stratified system $\mathcal{S} = (\mathcal{A}, \mathcal{D}, cost)$ and a query $\phi \in \mathcal{D}$

1: $C \leftarrow \emptyset$; $D = D_\varnothing$
2: **loop** (start with $j = 0$)
3: $\quad D_\oplus \leftarrow$ the next environment update $E_j|\mathcal{D}^\mathcal{E}_{in}$
4: $\quad (C, D) \leftarrow$ CONFIGURE$(C, D \oplus D_\oplus)$
5: $\quad$ **if** $\phi \in out_C$ **then** inform operator about ϕ and $D|\phi$
6: **end loop** (increment j)

7: **function** CONFIGURE(C, D) ▷ *returns* $(Configuration, Database)$
8: $\quad C \leftarrow C \cap F^*_{CAF}(\emptyset)$
9: $\quad$ **loop**
10: $\quad\quad C_{acc} \leftarrow \{A \in \mathcal{A}^*_\phi \setminus C \mid A$ is safely acceptable to $C\}$
11: $\quad\quad$ **if** $C_{acc} = \emptyset$ or $\phi \notin out_{\overline{\mathcal{A}_\phi}^*(C \cup C_{acc})}$ **then return** (C, D)
12: $\quad\quad A_{min} \leftarrow \arg\min_{A \in C_{acc}} cost(A)$
13: $\quad\quad D \leftarrow A_{min}(D)$ if $D|in_{A_{min}}$ changed since the last execution of A_{min}
14: $\quad\quad$ **if** A_{min} attacks $\{A'_1, \ldots, A'_k\} \subseteq C$ **then**
15: $\quad\quad\quad C \leftarrow C \setminus \{A'_1, \ldots, A'_k\}$ and set all $D|x$ on which A_{min} attacks some A'_i to $\varnothing$
16: $\quad\quad$ **else** $C \leftarrow C \cup \{A_{min}\}$
17: $\quad$ **end loop**
18: **end function**

Algorithm 1 provides a pseudocode for continuous reconfiguration of information-aggregation systems based on the principles embodied in the above analysis. Upon every environment update, in a step j, the algorithm tries to compute the minimal solution to the current configuration problem. Either it succeeds and informs the operator about the query solution, or detects that a solution can't be computed and proceeds. Function CONFIGURE computes the grounded extension of the current configuration problem $\mathfrak{C}_i = (\mathcal{S}, \phi, D \oplus E_i|\mathcal{D}^\mathcal{E}_{in})$ restricted to the arguments relevant to ϕ and considers potentially acceptable arguments individually in a greedy manner according to the cost of their execution.

Given a configuration, without executing the agents, the algorithm strips C of all arguments which might need reconsideration (line 8) due to the last environment update (line 4), or because they depend on such arguments. Starting from an empty candidate solution C, in every iteration, the algorithm firstly identifies among the arguments relevant to ϕ (Proposition 2) those potentially acceptable to C (line 10). Before considering their execution, it checks whether a solution can still be computed and should this not be the case, it terminates the procedure. To detect the condition, it exploits the principles presented in Proposition 3. Further, the algorithm selects the cheapest potentially acceptable information-processing agent A_{min} (line 12) and executes it (line 13). In the case A_{min} does not attack the current candidate solution C (line 14), it is accepted to C (line 16). Otherwise, the arguments attacked by A_{min} were previously accepted to C prematurely and thus need to be removed. We also need to set the variables on which they disagree to $\varnothing$ so as to ensure that all agents dependent on controversial valuations

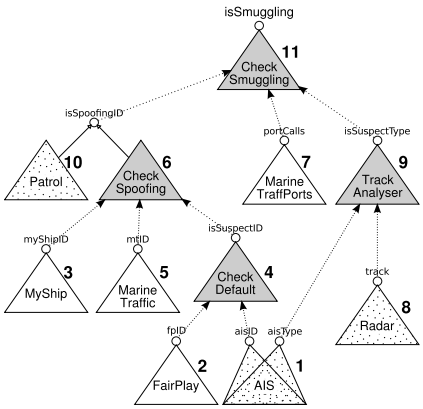

Fig. 3. Ordering of information-aggregation agents as considered by Algorithm 1.

will be deemed non-valid in the future iterations (line 8). To further reduce the costs incurred by the algorithm, we assume that each agent keeps track of changes to its input, so the algorithm executes it only in the case its re-execution is really needed (line 13).

Algorithm 1 is greedy, in that it always selects the cheapest agent to accept. Hence, although the solutions it computes are optimal it does not always incur the minimal possible run-time cost in terms of the cost of execution of the individual agents. The optimal strategy of selecting the next agent to execute is most likely application-domain dependent.

METIS 5 (configuration) *Consider the example configuration problem* $\mathfrak{C} = (\mathcal{S}_{\text{METIS}}, isSmuggling, D_{\varnothing})$. *In order to compute a solution to the problem, assuming that all the agents produce crisp valuations for their output variables upon their execution, in subsequent iterations Algorithm 1 would execute the agents as depicted in Figure 3. Noteworthy, in step 8, the cheapest agent to consider is the* TrackAnalyser*, but the algorithm is forced to choose the* Radar *agent as that is the cheapest and, unlike* TrackAnalyser*, safely acceptable at the same time. For illustration of detection of hopelessness of configuration computation, consider the agent* MarineTraffPorts *producing an unknown valuation for portCalls variable in step 7. The algorithm would immediately detect (line 11) that isSmuggling variable is not computable any more and would stop the computation.*

5 Final remarks and outlook

As of spring 2013, the METIS prototype, fragment of which is described here, was implemented and delivered. Figure 4 provides a screenshot of the operator's view in the prototype. It shows several vessels (circular glyphs) in a selected monitored coastal area with indication of the most likely values of their selected attributes. The pop-up inspection window shows the likelihoods of the vessel satisfying the target indicators,

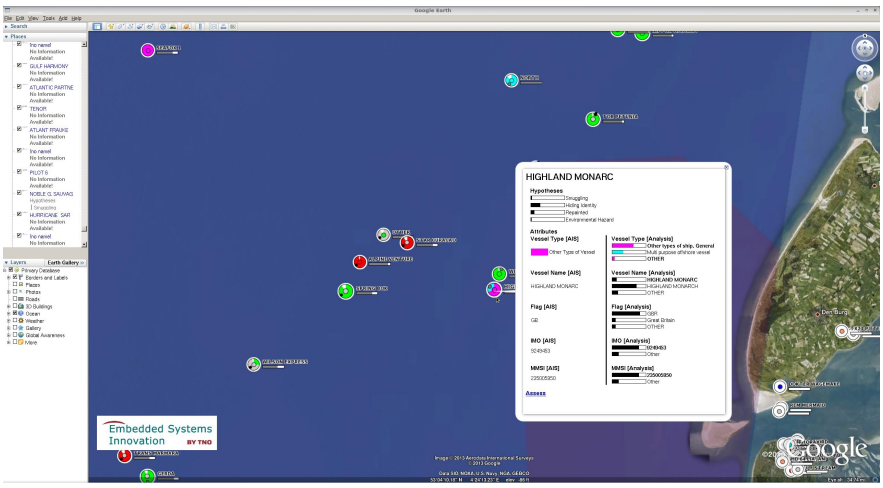

Fig. 4. METIS system screenshot. The background map imagery, courtesy of © 2013 *Google*, © 2013 *Aerodata International Surveys*, *Data SIO*, *NOAA*, *U.S. Navy*, *NGA* and *GEBCO*.

such as suspicion of a smuggling intent. In the system, the relative size of the vessel glyph corresponds to the cost of the system configuration instantiated for the vessel.

Above, we introduced a formal framework for modelling information-aggregation systems, such as METIS, providing a basis for a rigorous formulation of (re-)configuration problems. We argue that sceptical semantics of argumentation is a natural fit for modelling such systems and paves the way for further study of their properties, as well as development of algorithms for their continuous adaptation on a solid basis of the existing body of research in argumentation theory and logic programming. In our future work we intend to explore these relationships, specifically to study cost- and information-age-constrained reconfiguration of METIS, as well as the relationship of the introduced approach to computation of well-founded models for logic programs [3]. The dynamic nature of the system also invites to study links between their evolution and standard results from theories of evolving knowledge bases (e.g., [9]), logic program updates, belief revision, etc.

Acknowledgements This work was supported by the Dutch national program COMMIT. The research was carried out as a part of the METIS project under the responsibility of the *TNO-Embedded Systems Innovation*, with *Thales Nederland B.V.* as the carrying industrial partner.

References

1. Automatic Identification System. http://en.wikipedia.org/wiki/Automatic_Identification_System, April 2013.
2. Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
3. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.
4. Teun Hendriks and Piërre van de Laar. Metis: Dependable cooperative systems for public safety. *Procedia Computer Science*, 16(0):542 – 551, 2013.
5. IHS. IHS Fairplay Bespoke Maritime Data Services. http://www.ihs.com/products/maritime-information/data/, April 2013.
6. TNO Embedded Systems Innovation. METIS project. http://www.esi.nl/research/applied-research/current-projects/metis/, April 2013.
7. M. Jamshidi. System of systems engineering - New challenges for the 21st century. *Aerospace and Electronic Systems Magazine, IEEE*, 23(5):4–19, May 2008.
8. Hans-Joachim Klein. Null values in relational databases and sure information answers. In Leopoldo E. Bertossi, Gyula O. H. Katona, Klaus-Dieter Schewe, and Bernhard Thalheim, editors, *Semantics in Databases*, volume 2582 of *Lecture Notes in Computer Science*, pages 119–138. Springer, 2001.
9. J. A. Leite. *Evolving Knowledge Bases*, volume 81 of *Frontiers of Artificial Intelligence and Applications*. IOS Press, 2003.
10. Maltenoz Limited. MarineTraffic.com. http://www.marinetraffic.com/, April 2013.
11. MyShip.com. MyShip.com – Mates, Ships, Agencies. http://myship.com/, April 2013.