Argument & Computation
Vol. 00, No. 00, Month 200x, 1–22

RESEARCH ARTICLE

Context-aware reconfiguration of large-scale surveillance systems: argumentative approach

Peter Novák* and Cees Witteveen

Algorithmics, EEMCS, Delft University of Technology, The Netherlands
E-mail: {P.Novak,C.Witteveen}@tudelft.nl

(*Received 16 June 2014; final version received 23 December 2014*)

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 like METIS typically comprise a number of heterogeneous information sources and information aggregators. Among the main problems of their deployment lies their *scalability* 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 effective and 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. Finally, we demonstrate the flexibility of the presented framework by extending the proposed algorithm to adapt to context-dependent changes in information sources availability, as well as shifts in system's focus according to its context.

Keywords: applications of argumentation technology; context-aware configuration & reconfiguration; information-aggregation agents; heterogeneous information sources; grounded semantics; maritime traffic surveillance;

1. Introduction

The METIS project (2, 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. Systems-of-systems are large-scale integrated systems that are heterogeneous and independently operable on their own, but are networked together for a common goal (7). Here, 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. Our focus here is on supporting continuous *reconfiguration*, that is, efficient 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

*Corresponding author.

ISSN: 1946-2166 print/ISSN 1946-2174 online

DOI: 10.1080/1946216YYxxxxxxxx
http://www.informaworld.com

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 similar to the one implemented and demonstrated to industrial partners of the project in spring and autumn 2013, 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, or imaging satellites can incur rather significant costs to the system's operation. Hence, using all available information sources and information fusion components is not always feasible. 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 facilitate an early detection of malicious intents in the most efficient manner?*

To answer this question we will use tools and methods derived from *argumentation theory*. Abstract argumentation theory (1) studies logical reasoning solely in terms of inter-relationships of arguments, abstract entities representing inference mechanisms, not unlike opaque information aggregators of the METIS system. In a consequence, the argumentative approach provides a solid basis for modelling information processing systems in terms of interrelated arguments which support, or attack each other.

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*. This paper builds upon and significantly extends the results presented in (12). After introducing the basic concepts (Section 2) and a preliminary analysis of evolutions of information-aggregation systems, in Section 4, we present the problems of configuration and reconfiguration of information-aggregation systems to account for changes in their environments. Subsequently, in Section 5, we show that suitable system configurations correspond to the concept of grounded extensions of an associated argumentation framework and provide an algorithm for continuous reconfiguration of information-processing systems with respect to the changes in their environment. The solution concept

viewed through the optics of abstract argumentation is closely related to standard results in logic programming, so the relationship opens the door for further study of reconfiguration in relation to standard results in logic programming. Finally, in Section 6 we extend the reconfiguration algorithm to account for context-dependent availability of information sources in the system and dynamic changes in system's focus according to context-dependent query specification. The results introduced in Section 6 comprise a major advancement over the results presented in (12). A discussion of on-going and future work along the presented line of research concludes the paper.

Throughout the discourse, in a series of example expositions indicated under the heading **Metis x.y**, we describe the relevant parts of the METIS system and identify a class of relevant solution concepts. These expositions present simplified example fragments of the delivered prototype and are aimed at supporting the discourse as a running example.

Metis 1.1 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 following course of action. Such can include for instance sending a coast-guard boat patrol to check the situation, engage additional, possibly costly, information sources like a satellite high-resolution video feed, or a human expert, or can even alert and engage police, or other entities concerned with public security. 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.

To illustrate the functionality of METIS, 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* (4) database and web-portals of *MyShip.com* (11), *MarineTraffic.com* (10) and its *Ports of Call* database storing the harbours the ship visited in the recent past. There are also two physical sensors: a receiver for *Automatic Identification System* (AIS) (3, 5) messages, the vessels transmit themselves, and *radar* providing kinematic signatures of vessel tracks. Besides cross-validation 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. Information-aggregation systems

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 aggregator* 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.

We model an abstract information-aggregation system as a tuple $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ comprising a finite set of information processing agents and a database schema, 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 *true*, *false*, and *unknown* valuations respectively.

Remark 1 : In practice, *Dom* could include an arbitrary number of distinct crisp

valuations as far as *Dom* remains finite. The actual METIS system exposure indeed assumes such an extended domain of the database schema.

Without loss of generality, we do not distinguish between different interpretations of the unknown value $\varnothing$: *no information* and *value existent, but unknown* (8). A database snapshot (*database*) $D : \mathcal{D} \to Dom$ 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*. Let $D|x$ denote the valuation of a variable x in D and $D_\varnothing$ be a database snapshot with all variables valued as unknown.

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 $A : (\mathcal{D} \to Dom) \to (\mathcal{D} \to Dom)$ for each $A \in \mathcal{A}$. Usually, an information processing agent is not interested in the complete set of D-valuations to transform it to a new set of D-valuations: only a part of the database variables and their values are of its interest and dependent upon their values will be used to change the value of some other variables. Therefore, we assume that there is a specific subset of input values that is considered by agent A, denoted by $in_A \subseteq \mathcal{D}$ and a specific set of variables (possibly) updated by A, denoted by $out_A \subseteq \mathcal{D}$. Using in_A and out_A, we can consider an agent A as a function mapping partial interpretations into partial interpretations. Formally:

$$A : (\mathcal{D} \to Dom)|in_A \to (\mathcal{D} \to Dom)|out_A$$

Let D be an arbitrary snapshot of $\mathcal{D}$ and let $D|in_A$ and $D|out_A$ denote value assignments to variables of in_A and out_A, respectively. Then A maps the snapshot D to a snapshot $D' = A(D)$ as follows:

(1) $D'|out_A = A(D|in_A)$;
(2) for every $x \notin out_A$, $D'|\{x\} = D|\{x\}$.

That means that only the bindings of the variables in out_A might have been changed by an information processing agent A. We model a special type of information processing agents, information sources, as standard agents, however, with an empty set of input variables $in_A = \emptyset$ and a non-empty set of output variables $out_A \neq \emptyset$. We denote the set of all information-source agents within a system by $\mathcal{A}_{src}$.

Metis 2.1 In the prototype scenario (Figure 1), METIS features 7 information-source agents (white), including 3 physical sensors (dotted) and 4 non-trivial information-aggregation agents (grey).

The system contains a CheckDefault aggregator agent. This agent consults the local physical AIS sensor and cross-validates the self-transmitted vessel identity with those listed in the *IHS FairPlay* database. The identity of a ship is the value of a variable *aisD*, the outcome of checking an *IHS FairPlay* database is stored in the variable *fpID*. So, $in_{\mathsf{CheckDefault}} = \{aisID, fpID\}$. If there is a mismatch, the aggregator will set the variable *isSuspectID* to true, if there is a match, this variable is set to false. So, $out_{\mathsf{CheckDefault}} = \{isSuspectID\}$.

In a similar fashion, upon 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 spoof it. If that is indeed the case, the system escalates to the highest-level information-aggregator CheckSmuggling consulting the most extensive set of information sources and aggregators 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 valua-

agent	**in**	**out**
AIS	$\emptyset$	$aisID^*$, $aisType'$
FairPlay	$\emptyset$	$fpID^*$
MyShip	$\emptyset$	$myShipID^\dagger$
MarineTraffic	$\emptyset$	$mtID^\dagger$
MarineTraffPorts	$\emptyset$	$portCalls^\ddagger$
Radar	$\emptyset$	$track'$
Patrol	$\emptyset$	$isSpoofingID^\ddagger$
TrackAnalyser	marked $'$	$vesselType^\ddagger$
CheckDefault	marked $*$	$isSuspectID^\dagger$
CheckSpoofing	marked $\dagger$	$isSpoofingID^\ddagger$
checkSmuggling	marked $\ddagger$	$isSmuggling$

Table 1. Metis system agents

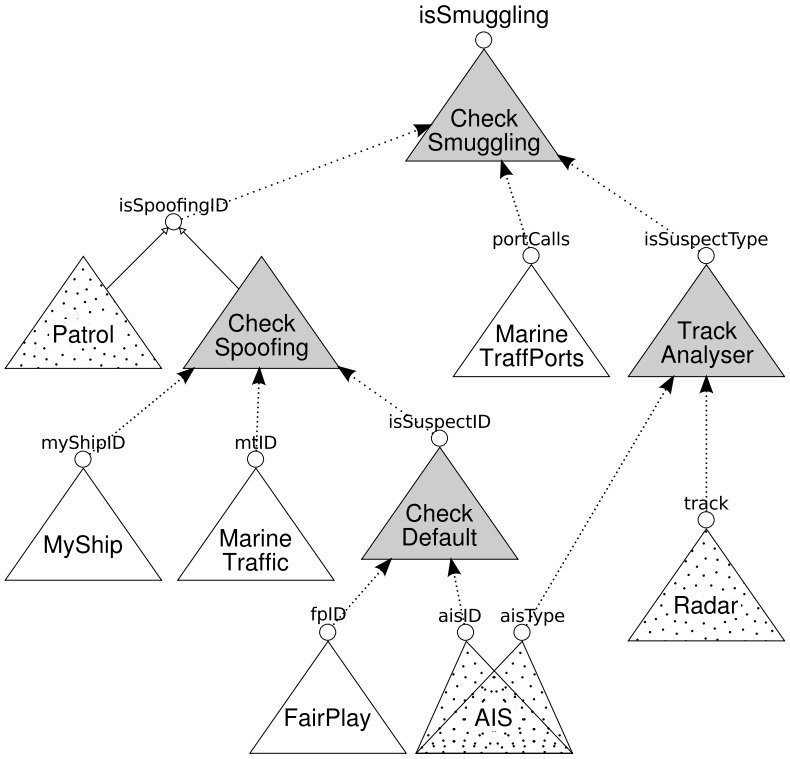

Figure 1. Metis system interdependencies.

tion 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 unknown valuation $\varnothing$ for each of their output variables.

3. Configurations and database evolution

A set of information processing agents of a system gives rise to the notion of system configuration. Formally, a configuration $C \subseteq \mathcal{A}$ of a system $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ is a set of information processing agents *active* in $\mathcal{S}$ in a given point in time. From now on, unless explicitly stated otherwise, we consider only non-trivial configurations $C \neq \emptyset$. Notation for input and output variables of an agent naturally extends to

configurations, that is $in_C = \bigcup_{A \in C} in_A$ and $out_C = \bigcup_{A \in C} out_A$. The notion of an update of a database snapshot given a configuration of agents can be easily defined as follows:

Definition 3.1: Let $C \subseteq \mathcal{A}$ be a configuration of a system $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ and D a database snapshot of the schema $\mathcal{D}$. Then the database D' is said to be an *update* of D by C, denoted by $C(D) = D'$, iff

(1) every change in D' w.r.t. D can be attributed to an information processing agent, i.e., for each $x \in \mathcal{D}$, such that $D|x \neq D'|x$, there exists an agent $A \in C$ with $x \in out_A$ and $D'|out_A = A(D|in_A)$.
(2) an update does not change the database snapshot only if no agent $A \in C$ is able to make a change, i.e., $D' = D$ only if for every agent $A \in C$, $A(D|in_A) = D|out_A$.

If $D|out_A \neq D'|out_A = A(D|in_A)$, we also say that the update was *co-induced* by the agent A.

Note that here we give priority to an agent in a configuration that is able to change the current database snapshot. Moreover, each variable x modified in the update D' w.r.t. its original value in D, is a result of a (single) computation of some agent A in the configuration C. This does not imply that an update $D' = C(D)$ is the result of at most one information processing agent $A \in C$, it only implies that there is no interference between agents in C during an update.

Instead of referring to configurations of agents, we might also refer to an update D' of D by means of a partial database D_u. The information in this partial database D_u should override existing information in D only if the information in D_u is known, i.e., the value of a variable x in D_u is either $\top$ or $\bot$:

Definition 3.2: Let D be a database and D_u be a partial database. We say that D' is an update of D by a partial database D_u, denoted by $D' = D \oplus D_u$, iff whenever $D_u|x$ is defined, we have $D'|x = D_u|x$ and $D'|x = D|x$ otherwise.

Given a configuration C of agents and an update $D' = C(D)$ of D, it might be that agents in C might be used to update D' too, resulting in a new update $D'' = C(D')$. This gives rise to the notion of an *evolution* of a system $\mathcal{S}$ under a configuration $C \subseteq \mathcal{A}$:

Definition 3.3: Let $\mathcal{S}$ be a system, D_0 be a database snapshot and $C \subseteq \mathcal{A}$ a configuration. An *evolution* of a system $\mathcal{S}$ under a configuration $C \subseteq \mathcal{A}$ from D_0 is an 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 by C, for all $i \in \mathbb{N}_0$. The evolution $\lambda_{\mathcal{D}}$ is called a *C-evolution* of $\mathcal{S}$ from D_0 on.

In general, given a configuration C and an initial snapshot D_0 there might be many different evolutions $\lambda_{\mathcal{D}}$ starting from D_0, depending upon the agents active at every update of the current snapshot. So an evolution of a system can be considered a non-deterministic process. Among these evolutions, there are special evolutions that interest us: these are the evolutions that from some point k on don't change. Such evolutions we deem *stable*:

Definition 3.4: Let $\lambda_{\mathcal{D}} = D_0, \ldots, D_k, \ldots$ be a *C-evolution* of $\mathcal{S}$. We say that $\lambda_{\mathcal{D}}$ is *stable* if there exists a constant $k \geq 0$ such that

(1) $D_0, \ldots, D_k$ is an initial segment of $\lambda_{\mathcal{D}}$;
(2) for all possible C-evolutions $\lambda'_{\mathcal{D}} = D'_0, \ldots, D'_k, D'_{k+1}, \ldots$ such that $D'_j = D_j$ for $j \leq k$, i.e., $\lambda'_{\mathcal{D}}$ and $\lambda_{\mathcal{D}}$ share the same initial segment $D_0, \ldots, D_k$, we have $D'_{k+i} = D_{k+i} = D'_k$ for all $i \geq 1$.

The state D_k is also known as the *stable state* in the evolution $\lambda_{\mathcal{D}} = D_0, \ldots, D_k, \ldots$.

Stable evolutions also can be identified with their finite initial segment $D_0, \ldots, D_k$. There is an easy characterisation of stable evolutions using the definition of a database update:

Proposition 3.5: *Let $\lambda_{\mathcal{D}} = D_0, \ldots, D_k, \ldots$ be a C-evolution. Then $\lambda_{\mathcal{D}}$ is stable iff there exists some finite k such that $C(D_k) = D_k$.*

Proof: The only-if direction is trivial, so assume that $C(D_k) = D_k$. Then, for all $A \in C$ we have $A(D_k|in_A) = D_k|out_A$. This means that for any C-evolution $\lambda''_{\mathcal{D}}$ starting from D_k we have $\lambda''_{\mathcal{D}} = D_k, D_k, \ldots, D_k, \ldots$. Hence, for every evolution $\lambda'_{\mathcal{D}} = D'_0, \ldots, D'_k, D'_{k+1}$ sharing $D'_0 = D_0, \ldots, D'_k = D_k$ with $\lambda_{\mathcal{D}}$, we have $D_{k'+1} = C(D'_k) = D_k$. Hence, $\lambda_{\mathcal{D}}$ is stable. □

The 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. So, even if a C-evolutions from a given snapshot D_0 turn out to be stable, it might be that there are several distinct stable states reached by these C-evolutions from the same initial snapshot D_0. In general, this is not what we want: given any initial situation, we want to draw a definite set of conclusions from it. Therefore, we would like to characterise C-evolutions from a certain initial snapshot that not only are stable, but at their points of stability turn out to reach the same stable state, whatever initial snapshot we take. The following definition ariculates this intuition formally:

Definition 3.6: Let $C \subseteq \mathcal{A}$ be a configuration of a system $\mathcal{S} = (\mathcal{A}, \mathcal{D})$. We say that C is *normal* iff

(1) for every database snapshot D_0 all C-evolutions of $\mathcal{S}$ from D_0 stabilise and
(2) all the stable states achieved by these stable C-evolutions from D_0 are identical.

More formally, a configuration C of agents is normal if for every initial database snapshot D_0 and every evolution $\lambda^i_{\mathcal{D}} = D_0, D^i_1, \ldots D^i_k, \ldots$ starting from D_0, there exists a finite constant k_i such that $C(D^i_{k_i}) = D^i_{k_i+1}$ and for all $i \neq j$, $D^i_{k_i+1} = D^j_{k_j+1} = D^i_{k_i}$. The unique stable state reached by such a normal configuration C from an initial snapshot D_0 is denoted by $C^*(D_0)$.

Clearly, not all configurations of any information-aggregation system are normal. To see this, consider the following example:

Example 3.7 A solution to a configuration problem does not always exist. For instance, consider three agents A_1, A_2 and A_3. Suppose that A_1 is an information source agent where $out_{A_1} = \{x\}$. A_1 is able to set x to $\top$. A_2 and A_3 are information processing agents, where $in_{A_2} = out_{A_3} = \{x\}$ and $in_{A_3} = out_{A_2} = \{y\}$. It holds that

$$A_2(x \mapsto \bot) = y \mapsto \top \tag{1}$$

$$A_2(x \mapsto \top) = y \mapsto \bot \tag{2}$$

while

$$A_3(y \mapsto \top) = x \mapsto \top \tag{3}$$

$$A_3(y \mapsto \bot) = x \mapsto \bot \tag{4}$$

Let $C = \{A_2, A_3\}$ be a configuration. Let D_0 be the snapshot after A_1 provides a crisp value to x. Then there exists no stable C-evolution starting with D_0 as the updates from D_0 oscillate between x, y being true and x, y being false. Hence, the configuration $C = \{A_2, A_3\}$ is not a normal configuration.

As we already pointed out, the precise operational semantics of application of a configuration to an information-system's database snapshot remains abstract. In particular, the configuration execution model is not precisely defined in terms of ordering of the individual agents, as well as in terms of possible effects concurrent execution might have on the underlying database. Since our study relates to design of information-aggregation systems which lend themselves to an analytical insight, the following questions are central and of natural interest to guide design of information-aggregation systems with a more transparent system evolution semantics:

What are the conditions which need to be imposed on information-aggregation systems and their underlying databases, so that existence of non-trivial normal configurations is guaranteed?

More specifically, ignoring the restrictions imposable on database snapshots, we are interested in the question:

What are the properties of information-aggregation systems which guarantee that regardless of the database snapshot of the system, there exists a non-trivial normal configuration?

Tackling the above questions in their generality is beyond the scope of this paper. Instead, to give a baseline for our further analysis below we introduce a constrained class of information-aggregation systems with a property that they always enjoy normality. More specifically, these systems have the property that given an arbitrary configuration C of information-aggregation agents, whenever the information source agents have "produced" an initial database snapshot D_0, any C-evolution from D_0 will result in the same stable state. We delve into the details and rationale of such configurations below.

Definition 3.8: We say that an information-aggregation system $S = (\mathcal{A}, \mathcal{D})$ is *simple and stratified* iff

(1) there exists a *stratification* of $\mathcal{A}$, that is a partitioning of $\mathcal{A}$ into a sequence $(\mathcal{A}_0, \ldots, \mathcal{A}_k)$ of *strata* (layers) , where $\mathcal{A}_0 = \{A \in \mathcal{A} \mid in_A = \emptyset\}$ and $\mathcal{A}_i = \{A \in \mathcal{A} \mid in_A \subseteq out_{\bigcup_{j=1..i-1} \mathcal{A}_j}\}$ for all $i = 1, \ldots, k$; and
(2) S is *simple* in that for every every variable $X \in \mathcal{D}$ there is at most one agent $A \in \mathcal{A}$ such that $X \in out_A$.

As it turns out, these conditions are sufficient to guarantee normality of configurations:

Proposition 3.9: *Let $S = (\mathcal{A}, \mathcal{D})$ be a simple and stratified system. Then every configuration C of information processing agents in S is normal w.r.t. any initial database D_0 produced by the information source agents $A \in \mathcal{A}_0$.*

Proof: The proof follows by induction over the number k of layers in the stratification $(\mathcal{A}_0, \ldots, \mathcal{A}_k)$ of $\mathcal{A}$. Let $k = 0$. Then the only agents we have are information-source agents. So by assumption, every configuration comprising only information-source agents is stable as it can't update the database any more, since their outputs are already reflected in it. Hence, given any initial database snapshot D_0, $C(D_0) = D_0$, proving that every such configuration is normal.

Assume the induction hypothesis to hold for any stratification $(\mathcal{A}_0, \ldots, \mathcal{A}_j)$ with

$j \leq k$. Let S be a system with $k+1$ layers in its stratification, let C be an arbitrary configuration of agents an D_0 be an initial database snapshot. Let $C = C_{\leq k} \cup C_{k+1}$, where $C_{\leq k}$ contains all agents in C that occur in the layers 1 up to k and C_{k+1} contains all agents in the $k+1$-th layer. By induction hypothesis, we know that $C_{\leq k}$ is normal since it reduces to a configuration in a simple an stratified system $S' = (\bigcup_{i \leq k} \mathcal{A}_i, \mathcal{D})$ with k layers. Hence, $C^*_{\leq k}(D_0) = D^*$ is well-defined. Now let $|C_{k+1}| = m$ and consider an arbitrary C_{k+1}-evolution $\lambda_{\mathcal{D}} = D^*, D^*_1, \ldots D^*_m, \ldots$ starting with D^*.

First, we show that $C_{k+1}(D^*_m) = D^*_m$. We can divide D^* into two disjoint parts $D^* = D^*_{in} \cup D^*_{out}$ where $D^*_{out} = D^*|(\bigcup_{A \in \mathcal{A}_{k+1}} out_A)$ and $D^*_{in} = D^* - D^*_{out}$. Note that, by simplicity of S, all parts out_A are disjoint. Now the effect of any agent $A \in C_{k+1}$ is restricted to a possible modification of $D^*|out_A$ only, without affecting D^*_{in}. In particular, for any agent $A \in C_{k+1}$ its effect on D^* is limited to $D^*|out(A)$ which will be changed to $A(D^*|in(A))$. Therefore, after at most m updates, the cumulative effect of all updates by agents A in the configuration C_{k+1} equals $D^*_m = D^*_{in} \cup (\bigcup_{A \in C_{k+1}} A(D^*|in(A)))$ and any $A \in C_{k+1}$ applied to D^*_m will result in D^*_m again. Since, given D^*, D^*_m is uniquely defined, any C_{k+1} evolution $\lambda_{\mathcal{D}}$ is normal w.r.t. the initial database D^* reaching the unique stable state $D^{**} = D^*_m$.

Secondly, we show that D^{**} is the unique stable state any C-evolution $\lambda_{\mathcal{D}}$ will evolve to starting from an initial snapshot D_0. To prove this, we define D^{**}_j to be that part of the state D^{**} that contains only bindings for variables[1] in the layers 0 up to j, i.e., variables occurring in $in(A) \cup out(A)$ for any agent $A \in \bigcup_{i \leq j} \mathcal{A}_i$. For $j = 0$, $D^{**}_0 = D_0$ is the unique stable state any C-evolution $\lambda_{\mathcal{D}}$ will evolve to. Assume that for $j \leq k$, any C-evolution will evolve to the unique state D^{**}_j. Then for $j = k$ any C-evolution evolves to D^{**}_k. But then we must have $D^*_k = D^{**}_k$, implying that $D^*_{in} = D^{**}_k$. Now assume that there is another stable state $D^{***} \neq D^{**}$ reachable by a C-evolution $\lambda'_{\mathcal{D}}$ from D_0. By induction hypothesis, we have $D^*_{in} = D^{**}_k = D^{***}_k$. But then we must have $D^{**} = D^*_{in} \cup (\bigcup_{A \in C_{k+1}} A(D^*|in(A))) = D^{***}$, contradiction. Therefore, D^{**} is the unique state every C-evolution from D_0 will evolve to. Therefore, any C is normal w.r.t. any initial database snapshot D_0. □

While simple stratified systems represent a rather constrained and narrow class of information-aggregation systems, they are still a very useful subclass of systems. For instance most deployed sensor networks fall into this class of systems due to their uni-directional flow of information and relative simplicity of information fusion mechanisms.

METIS is a security-related information-aggregation system. Such knowledge-intensive systems are designed by encoding human expert knowledge into the structure of the system. In practice, however, we observe that domain experts tend to articulate their knowledge in terms of hierarchically structured information flows and cascading inference and filtering processes. This provides an intuitive justification for the *stratified design* of such systems. Complementary, the requirement of *simplicity* of a system as expressed in the above definition embodies the intuition that the easiest way to resolve conflicts is by doing so explicitly. That is, in the case there might be a conflict between two operating aggregators over a valuation of some variable, this should be resolved already in the design phase by explicitly splitting the computation of the two aggregators into two separate variables and designing an independent third aggregator capable to resolve such conflicts either by fusing their outcomes, deciding which of them takes precedence, or otherwise. More formally, whenever $x \in out_{A_1} \cap out_{A_2}$ for two distinct agents A_1, A_2, we can

[1]Without loss of generality we may assume that for all variables $X \in \mathcal{D}$ it holds that there exists an agent $A \in \mathcal{A}$ such that $X \in in(A) \cup out(A)$.

rename x in out_{A_1} to x_1, similarly rename x to x_2 in out_{A_2} and introduce a new agent A_x with $in_{A_x} = \{x_1, x_2\}$ and $out_{A_x} = \{x\}$. A_x then embodies the fusion of x_1 and x_2 into a single variable without a possibility of a conflict over x in S.

In the remainder of this paper, we flesh out the above introduced intuitions about systems' evolutions and design requirements which need to be imposed on them in order to facilitate "well-behaved" information aggregation. In particular, we discuss the issues stemming from embedding of information-aggregation systems in environments which might have their own dynamics. An example of such might be a mixed physical and IT infrastructure-related context our METIS prototype system operates in. Subsequently, we introduce and apply optics of abstract argumentation on the functionality of such information-processing systems. Along the way, we re-introduce the framework of information-processing systems as laid out above, this time in terms of argumentation frameworks. We demonstrate that the parallels between the two are useful as they allow us to relate the information-aggregation processes to reasoning, inference and conflict-resolution mechanisms of argumentation approaches. As a side effect, we lift the constraint of system's simplicity introduced in Definition 3.8 and show how we can arrive to sound conclusions of the information-aggregation processes in systems like METIS. Furthermore, as we will show, argumentation optics allows us to straightforwardly capture the notion of justification of a system's conclusion regarding a variable of interest, a query.

4. Configuration and reconfiguration problems

In this section we formulate the problems of configuration and reconfiguration of information-aggregation systems. Before that, however, we introduce and illustrate the concept of an environment such a system is to be embedded in. The notion of environment provides a connection of the system to the ground reality, the source of data the system processes and upon which it infers its conclusions.

4.1. *System and its 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 $\lambda_{\mathcal{E}} = E_0, \ldots, E_k, \ldots$ of database snapshots. 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 such a distinguished timepoint, the system is embedded in the

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

Figure 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.

current state of the environment.

Metis 4.1 A configuration capable to produce the system evolution depicted in Figure 2 could include the agents AIS, FairPlay, CheckDefault, Radar and TrackAnalyser agents executed subsequently in that order up to the database snapshot D_4. That is, $\mathbf{D_0}$ is produced by execution of AIS, $\mathbf{D_1}$ by execution of FairPlay, $\mathbf{D_2}$ by CheckDefault, $\mathbf{D_3}$ by Radar and $\mathbf{D_4}$ by TrackAnalyser. Subsequently, $\mathbf{D_5}$ is produced by a re-execution of AIS, which produces an unknown valuation $\varnothing$ for the *aisID* variable which leads to derivation of $\varnothing$ also for the *isSuspectID* variable when finally $\mathbf{D_6}$ is produced by re-execution of CheckDefault. 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. Assuming all the not-mentioned variables in the environment do not change, the system is embedded in it exactly at points marked by the environment updates.

4.2. *Configuration problem*

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 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 a 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 assume that the system is embedded in an environment state reflected in its current (initial) database snapshot.

Definition 4.2: Given a tuple $\mathfrak{C} = (\mathcal{S}, \phi, D)$, with $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ 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 a 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. We also require that the resulting *query solution* is a crisp valuation $C^*(D)|\phi \neq \varnothing$ computed by the configuration C;
(2) for each variable $x \in in_C$, also $x \in out_C$ and $C^*(D)|x \neq \varnothing$; and finally
(3) there is no configuration C' with $C \subset C'$ satisfying (1) and (2), such that $C'^*(D)|\phi \neq C^*(D)|\phi$.

Condition (1) of the definition above stipulates that the solution configuration

indeed provides a valuation of the query. 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. Finally, condition (3) ensures that there is no doubt about the computed query solution. In that sense, C is a minimal sufficient support of the query answer.

Definition 4.2 is agnostic of the structure of the underlying system. In a consequence, as we already pointed out in the previous section, in general, a solution to a configuration problem does not always exist.

4.3. *Reconfiguration problem*

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.

Definition 4.3: 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 $\mathcal{E}$, $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ is an 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 $C_0, \ldots, C_l, \ldots$, 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 4.4 Consider the METIS prototype scenario introduced in the previous expositions. An example configuration problem could be $\mathfrak{C} = (\mathcal{S}_{\text{METIS}}, \mathit{isSmuggling}, D_3)$. As stated, there is no solution to $\mathfrak{C}$. This would only exist if all the information-source agents provide a reading of their output variables. Then, the solution would comprise all the agents of the system.

5. Solving configuration and reconfiguration problems using argumentation theory

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 effect, they can be treated as blackbox modules embodying a support for for their output variables, or can produce an output in conflict with outputs of other agents within the system. Thus, their interrelationships embody the flow of information within the system in terms of mutual support of conflict.

Dung's theory of abstract argumentation (1) is a formal model revolving around analysis of mutual support and the structure of conflicts between abstract arguments in favour, or against some conclusion. Hence, the framework of abstract argumentation 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 (1).

Definition 5.1: 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 $\mathcal{D}$ as follows:

- 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 a 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*.

Metis 5.2 In the case of the example system depicted in Figure 1, an attack in a particular argumentation framework associated with a METIS system in some state might arise in the case the agents Patrol and CheckSpoofing produce two different crisp valuations for the variable *isSpoofingID*. This situation could realistically arise in the case when for instance the information provided by AIS and FairPlay agents cannot be cross-validated by that retrieved from other external databases (MyShip or MarineTraffic) and thus CheckSpoofing agent concludes that the ship might be spoofing its identity. At the same time, a coast-guard patrol boat, or a UAV sent to inspect the ship would actually confirm the identity of the vessel to be that retrieved from AIS.

Definition 5.3: 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, any 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 system 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. Besides being capable to articulate their conclusions to its users, these should also be susceptible to providing insights for the structural

explanations, justifications, of the conclusions. The notion of grounded extension provides a basis for our analysis.

Definition 5.4: 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 arguments 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 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 (1) 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, consistent with our earlier observations about systems like METIS, is to consider only stratified systems. That is those, for which there exists a stratification, a decomposition into a sequence of strata (layers) $\mathcal{A} = \mathcal{A}_0, \ldots, \mathcal{A}_k$ as defined in Definition 3.8. Furthermore, we say that $\mathcal{A}$ is the most compact stratification of $\mathcal{S}$ iff all agents belong the lowest possible layer of $\mathcal{A}$. Formally, for all stratifications $\mathcal{A}'$ of $\mathcal{S}$, $A \in \mathcal{A}_i$ implies $A \in \mathcal{A}'_j$ 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 5.5: *$C \subseteq \mathcal{A}$ is a solution of a configuration problem $\mathfrak{C} = (\mathcal{S}, \phi, D)$ with a stratified system $\mathcal{S}$ if and only if C is also the grounded extension of $CAF_{\mathfrak{C}}$ (i.e., $C = GE_{CAF_{\mathfrak{C}}}$), an argumentation framework associated with $\mathcal{S}$ over the database $C^*(D)$ with $\phi \in out_C$.*

Proof:

$\Longrightarrow$: if C is a solution of $\mathfrak{C}$, by definition we have that *i*) C is normal, *ii*) each argument within C is valid and *iii*) $\phi \in out_C$. In a consequence, the computation of $GE_{CAF} = F^*_{CAF}(\emptyset)$ follows in system's layers (strata) from the information sources to ever higher ones exactly copying the inductive argument presented in the proof of Proposition 3.9. Note, there is no execution of the involved agents, their output only needs to be checked against the database snapshot $C^*(D)$, which, however, is already a result of their execution. Since C is a solution to $\mathfrak{C}$, the computation must include also computation of a valuation for the query variable.

$\Longleftarrow$: we need to show that $C = F^*_{CAF}(\emptyset)$ is *i*) normal, *ii*) $\phi \in out_C$, *iii*) all input variables of agents in C are crisply valued, and *iv*) there is no larger configuration C' producing a different valuation for ϕ than C does.

(*i*) The proof of C's normality follows the inductive argument laid down in the proof of Proposition 3.9. Thanks to the insight that there is always a unique grounded extension of a stratified argumentation framework, we have that the computation of $F^*_{CAF}(\emptyset)$ forms a stable evolution of $\mathcal{S}$ from D onwards, moreover, all possible evaluations of agents from the individual layers must lead to the same outcome;

(*ii*) $\phi \in out_C$ holds by assumption;

(*iii*) all inputs to all arguments in C are indeed crisply valued thanks to the requirement of an argument to be valid and non-controversial in order to be considered acceptable to $F^*_{CAF}(\emptyset)$; and finally

(*iv*) $C = F^*_{CAF}(\emptyset)$ is a fix-point of F_{CAF}, hence it is also the maximal set of conflict-free arguments in $CAF_{\mathfrak{C}}$.

□

Proposition 5.5 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 5.5 would iteratively proceed in three steps following the inductive argument presented in the proof of Proposition 3.9. In every i-th iteration it would *i*) execute all the agents from stratum $\mathcal{A}_i$ 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 thus 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 5.6: *Let $\mathfrak{C} = (\mathcal{S}, \phi, D)$, $CAF_\mathfrak{C}$ and $C = GE_{CAF_\mathfrak{C}}$ be as in Proposition 5.5. Then $C \cap \mathcal{A}^*_\phi$ is also a solution to $\mathfrak{C}$.*

Proof: Observe, that since C is a solution to $\mathfrak{C}$, every fragment $C' \in C$ which still satisfies $\phi \in out_{C'}$ and the condition that for each variable $x \in in_{C'}$, also $x \in out_{C'}$ with $C^*(D)|x \neq \varnothing$ must also be a solution to $\mathfrak{C}$. From the definition of $\mathcal{A}^*_\phi$ and the fact that C is a normal configuration the two conditions are satisfied in $C \cap \mathcal{A}^*_\phi$. Finally, since C is already a solution to $\mathfrak{C}$, by definition no agent from $C \setminus \mathcal{A}^*_\phi$ attacks the valuation of $C^*(D)|\phi$, hence $C \cap \mathcal{A}^*_\phi$ also satisfies the condition (3) of Definition 4.2. □

Finally, 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 *i*) either already derived in a justified manner, or that *ii*) 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 as implied by the proof of Proposition 5.6 above. 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's define $\overline{\mathcal{A}_\phi}^*(C)$ as 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 and thus works bottom-up along the system's strata, while $\mathcal{A}_\phi$ worked top-down from the query down to the relevant information source in the system's bottom layer. 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 $\phi \in out_{\overline{\mathcal{A}_\phi}^*(C)}$

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 $\mathcal{S} = (\mathcal{A}, \mathcal{D})$ and a query $\phi \in \mathcal{D}$

1: $C \leftarrow \emptyset$; $D = D_{\varnothing}$
2: **loop** (start with $j = 0$)
3: $D_{\oplus} \leftarrow$ the next environment update $E_j|\mathcal{D}^{\mathcal{E}}_{in}$
4: $(C, D) \leftarrow$ CONFIGURE$(C, D \oplus D_{\oplus})$
5: **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: $C \leftarrow C \cap F^*_{CAF}(\emptyset)$
9: **loop**
10: $C_{acc} \leftarrow \{A \in (\mathcal{A}^*_{\phi}) \setminus C \mid A \text{ is safely acceptable to } C\}$
11: **if** $C_{acc} = \emptyset$ or $\phi \notin out_{\overline{\mathcal{A}_{\phi}}^*(C \cup C_{acc})}$ **then return** (C, D)
12: select $A \in C_{acc}$
13: $D \leftarrow A(D)$ if $D|in_A$ changed since the last execution of A
14: **if** A attacks $\{A'_1, \ldots, A'_k\} \subseteq C$ **then**
15: $C \leftarrow C \setminus \{A'_1, \ldots, A'_k\}$ and set $x \mapsto \varnothing$ for all x on which A attacks C
16: **else** $C \leftarrow C \cup \{A\}$
17: **end loop**
18: **end function**

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. A straightforward corollary of this line of reasoning is the following proposition, which formalizes the relationship between the operator and the structure of the grounded extension.

Proposition 5.7: *Let $\mathfrak{C} = (\mathcal{S}, \phi, D)$, $CAF_{\mathfrak{C}}$ and $GE_{\mathfrak{C}}$ be as in Proposition 5.5. 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 the system's 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 to 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 arguments outside of C which can potentially threat the valuations of its input variables. More formally, we require *i*) $in_A \subseteq out_C$, *ii*) there is no $x \in in_A$ with $D|x = \varnothing$, 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** provides a pseudocode for continuous reconfiguration of information-aggregation systems based on the above principles. 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 one by one.

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 5.6) 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 5.7. Further, the algorithm selects a potentially acceptable information processing agent A (line 12) and executes it (line 13). In the case A does not attack the current candidate solution C (line 14), it is accepted to C (line 16). Otherwise, the arguments attacked by A 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 will be deemed non-valid in the future iterations (line 8). To further reduce the run-time 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).

For simplicity, we do not specify the particular strategy in which the potentially acceptable arguments are selected from C_{acc} (line 12). One possible heuristic strategy could be to pick the arguments which can result in a conflict with other arguments first. This would lead to an early detection of hopelessness of the computation of a solution to the given configuration problem. Another strategy could be to select the arguments in a greedy manner according to estimation of the run-time costs incurred by executing the argument agent.

Metis 5.8 Consider the example configuration problem $\mathfrak{C} = (\mathcal{S}_{\text{METIS}}, \mathit{isSmuggling}, D_{\varnothing} \oplus E_1)$. In subsequent iterations, Algorithm **1** could execute the agents as follows. The + superscript marks agents accepted to the candidate solution: AIS^+, $\mathsf{FairPlay}^+$, MyShip, $\mathsf{CheckDefault}^+$, $\mathsf{MarineTraffic}^+$, etc. However, already after execution of MyShip, it would detect hopelessness of further computation and would terminate. The valuation of *myShipID* is vital to computation of the query solution.

Let us conclude the section with a brief remark on explanations, or justifications, which can be extracted from grounded extension of configuration argumentation frameworks as solutions to configuration problems. As articulated in Algorithm **1**, provided a system infers a solution to a given configuration problem, it should inform the operator about its query answer. One of the benefits of exploiting the argumentation optics on system reconfiguration is that the grounded extension reduced to only relevant arguments directly provides also a notion of justification of the system's conclusions. In particular, it only includes the arguments supporting the query answer, while excluding all the irrelevant ones. Given a stratified system, the query answer can thus be justified by a tree of crisp valuations of the database's variables connected by the information processing agents providing the relationships between them. Assuming that from the perspective of its user, a coast guard officer in the case of METIS, the system is designed intuitively and the information-aggregation agents embody a relatively encapsulated and single-purpose computation procedure, ideally, these relationships will be comprehensible and plausible explanations of the query answer for the system's user. In effect, instead of focusing on the conflict-resolution strengths of Dung's abstract argu-

mentation approach, we exploit the mutual support relationships we enforced by requiring argument validity as a precursor for acceptability to the grounded extension.

6. Context-dependent information sources and queries

Information source agents of a system provide an interface to the system's environment. Algorithm **1** presented in the previous section assumed that all information sources are constantly available. While this assumption is plausible for automatic sources such as physical sensors (e.g., a thermometer, or a radar), other information sources might not be always available and can be only utilised depending on the system's context. As a consequence of the need to handle also such information sources with context-dependent availability, the system should be able to regularly retrieve the currently available information sources and perform its computation over these sources only.

Metis 6.1 Since Metis should be deployable also on board of a sea going ship, access to information sources such as coast guard patrol rapports, or unmanned aircraft is reasonable only with an explicit approval of a human officer and also only when in the range of utilisation of such a sensor, such as close to major harbours. Also, some sensors, such as stationary cameras are typically available at harbour entrances, but not on arbitrary locations along the coastline.

The main purpose of information processing systems is to perform continuous surveillance of their environment and attempt to infer valuation of a distinguished indicator, a query. However, queries of the system can also be a subject to change over time as the system's focus shifts according to the particular context of the monitored entities. To account for such dynamic changes of the information processing system's focus, the system needs to be able to accept changes in its currently relevant query and perform computations relevant to the actual query in a timely manner.

Metis 6.2 In the specific example of Metis, a sea-going ship continuously moves in the zone of under system's surveillance. While detection of smuggling intent of a given ship might be highly relevant while the vessel is approaching a major harbour, near a protected natural habitat, the system might focus rather on its cargo and kinematic behaviour in order to detect whether it might pose a hazard to the environment, or whether it might be involved in malicious activities, such as dumping garbage or chemical waste to the sea.

Algorithm **2** reformulates and extends Algorithm **1** to facilitate continuous reconfiguration of information-aggregation systems with information sources with context-dependent availability, as well as with dynamic, context-dependent queries. We expose Algorithm **2** as an event-driven algorithm sequentially reacting to three main types of events which can occur. Most importantly, upon an update of the system's environment (line 3), the algorithm effectively reconfigures the system by first retrieving the environment database update and subsequently invoking the function Configure from Algorithm **1**, thus updating the system's configuration. If necessary, the algorithm informs the system user about the computed answer to the currently relevant query. Note, the configuration function is, however, always invoked only on the *currently relevant fragment* of the original system $\mathcal{S}$ relevant to the currently enabled information sources $\mathcal{A}_{enabled}$ and the currently relevant query ϕ. The fragment is precisely defined as $\overline{\mathcal{A}_\phi}^*(\mathcal{A}_{enabled})$. That is, all the information

Algorithm 2 Event-driven algorithm computing weak-solutions to a reconfiguration problem with context-dependent queries and information sources availability.

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

1: $C \leftarrow \emptyset$; $D \leftarrow D_{\varnothing}$; $\mathcal{A}_{enabled} \leftarrow \mathcal{A}_{src}$;
2: **forever handle events** (start with $j = 0$)
3: **on** environment update (increment j)
4: $D_{\oplus} \leftarrow$ the next environment update $E_j|(\mathcal{D}^{\mathcal{E}}_{in} \cap in_{\mathcal{A}_{enabled}})$
5: $(C, D) \leftarrow$ CONFIGURE(C, D) over the fragment $\mathcal{S}' = (\overline{\mathcal{A}_{\phi}}^{*}(\mathcal{A}_{enabled}), \mathcal{D})$
6: **if** $\phi \in out_C$ **then** inform operator about ϕ and $D|\phi$
7: **on** info-sources update
8: $\mathcal{A}_{enabled} \leftarrow$RETRIEVECTXENABLEDSOURCES()
9: recompute $\overline{\mathcal{A}_{\phi}}^{*}(\mathcal{A}_{enabled})$
10: **on** query update
11: $\phi \leftarrow$RETRIEVECTXQUERY()
12: recompute $\overline{\mathcal{A}_{\phi}}^{*}(\mathcal{A}_{enabled})$
13: **end**
14: **end**

processing agents depending on the currently enabled information sources and at the same time relevant to the actual query.

Upon detecting a contextual change in either the currently available information sources (line 7) or the currently relevant query (line 10), the algorithm retrieves the set of enabled information sources or the query, and subsequently recomputes the currently relevant fragment $\overline{\mathcal{A}_{\phi}}^{*}(\mathcal{A}_{enabled})$ of the system $\mathcal{S}$.

Finally, observe that the retrieval of an environment update $D_{\oplus}$ (line 4) is restricted only to the currently available information sources and in result, the system ignores the changes of the environment which it can't observe.

7. Discussion and final remarks

Above, we presented an approach for modelling information-processing systems geared towards continuous surveillance of a mixed physical and software environments largely inspired by Dung's approach to argumentation (1). We also demonstrated usefulness of modelling such systems in terms of arguments and analysing their interrelationships with respect to potential conflicts between outputs of their computations. The conceptual formal framework provides a sound and flexible basis for a rigorous formulation of (re-)configuration problems and their various extensions as also demonstrated in Section 6, where we present an approach to handle not only the dynamics of an environment, but also that of the system itself, as dictated by its changing context. We argue that sceptical semantics of argumentation frameworks is a natural fit for modelling systems like METIS and our approach paves the way for further study of their properties, as well as development of algorithms for their continuous adaptation on a the 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 further extensions of reconfiguration problems, including optimisation of run-time costs with respect to explicit costs incurred by the system computation, or reconfiguration with respect to ageing information in the system's database. In the argumentation-relevant terminology, this means studying extensions of abstract ar-

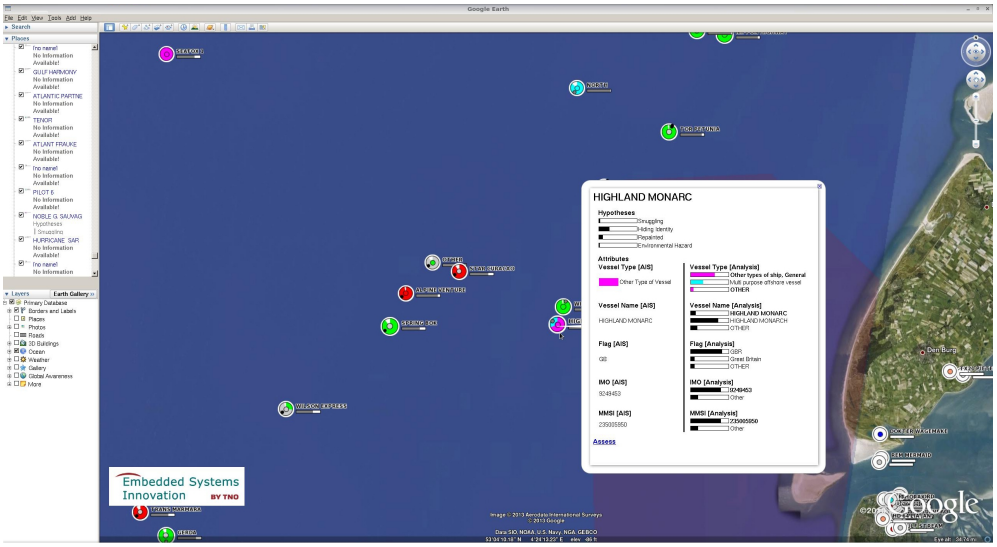

Figure 3. Metis system screenshot. The background map imagery, courtesy of © 2013 *Google*, © 2013 *Aerodata International Surveys*, *Data SIO*, *NOAA*, *U.S. Navy*, *NGA* and *GEBCO*.

gumentation to include notions of a cost of an argument and its inclusion in an extension, or time-dependent strength of argument's attacks, etc. Further inspirations stemming from the dynamic nature of such systems also invite to study links between their evolution and standard results from theories of evolving knowledge bases (e.g., (9)), logic program updates, belief revision, etc. In particular, among other challenges, we are also interested in dynamic run-time changes of the system's structure, in other words dynamic changes of the underlying configuration argumentation framework. We also aim at studying extensions and applications of the presented approach towards lifting structural constraints imposed on systems, such as stratification and inclusion of cyclic, or more involved dependencies among information aggregation agents.

Throughout of the presented paper, we introduced example fragments inspired by the actual implementation of Metis demonstrators delivered to the Metis project's industrial partners in spring and autumn 2013. Figure 3 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, such as suspicion of a smuggling intent, or environmental hazard as discussed throughout this paper. An extended account of the Metis system functionality as of 2014 and AI-related technologies employed in its implementation can be found in (13).

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] 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.

[2] Teun Hendriks and Piërre van de Laar. METIS: Dependable Cooperative Systems for Public Safety. *Procedia Computer Science*, 16(0):542–551, 2013.

[3] IEC. *International Standard IEC 62320-1: Maritime navigation and radiocommunication equipment and systems - Automatic Identification System (AIS) - Part 1: AIS Base Stations*. International Electrotechnical Commission, February 2007.

[4] IHS. IHS Fairplay Bespoke Maritime Data Services. `http://www.ihs.com/products/maritime-information/data/`, April 2013.

[5] IMO. International Convention for the Safety of Life at Sea (SOLAS), 1974.

[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.

[12] Peter Novák and Cees Witteveen. Reconfiguration of Large-Scale Surveillance Systems. In Joao Leite, Tran Cao Son, Paolo Torroni, Leon van der Torre, and Stefan Woltran, editors, *Proceedings of the 14th Workshop on Computational Logic in Multi-Agent Systems, CLIMA XIV*, 2013.

[13] Marina Velikova, Peter Novák, Bas Huijbrechts, Jan Laarhuis, Jesper Hoeksma, and Steffen Michels. An Integrated Reconfigurable System for Maritime Situational Awareness. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 1197–1202. IOS Press, 2014.