HAL
archives-ouvertes.fr

Constructing Inference-Proof Belief Mediators

Joachim Biskup, Cornelia Tadros

▶ To cite this version:

Joachim Biskup, Cornelia Tadros. Constructing Inference-Proof Belief Mediators. 29th IFIP Annual Conference on Data and Applications Security and Privacy (DBSEC), Jul 2015, Fairfax, VA, United States. pp.188-203, 10.1007/978-3-319-20810-7_12 . hal-01745830

HAL Id: hal-01745830
https://hal.inria.fr/hal-01745830

Submitted on 28 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

Constructing Inference-Proof Belief Mediators*

Joachim Biskup and Cornelia Tadros

Fakultät für Informatik, Technische Universität Dortmund, Germany
{joachim.biskup|cornelia.tadros}@cs.tu-dortmund.de

Abstract. An information owner might interact with cooperation partners regarding its belief, which is derived from a collection of heterogeneous data sources and can be changed according to perceptions of the partners' actions. While interacting, the information owner willingly shares some information with a cooperation partner but also might want to keep selected pieces of information confidential. This requirement should even be satisfied if the partner as an intelligent and only semi-honest attacker attempts to infer hidden information from accessible data, also employing background knowledge. For this problem of inference control, we outline and discuss a solution by means of a sophisticated mediator agent. Based on forming an integrated belief from the underlying data sources, the design adapts and combines known approaches to language-based information flow control and controlled interaction execution for logic-based information systems.

Keywords: attacker simulation, security policy, controlled interaction execution, declassification, inference control, information flow control, integrated belief, mediation, multiagent system, reasoning.

1 Introduction

Today's IT-security technologies provide a broad variety of effective and efficient mechanisms to prohibit unauthorized reading of any kind of raw *data*, e.g., authenticated access control and private-key or certified public-key encryption. And these technologies also offer somehow limited approaches to confine the *information* content of data made accessible to cooperation partners, e.g., language-based information flow control, information systems with controlled interaction execution, confidentiality-preserving data publishing and cryptographic multiparty computations. However, independently of its carrier and its representation, information is the fundamental *asset* of an individual pursuing self-determination or an enterprise doing business. Moreover, information arises not only from accessible data but essentially also from social *contexts* as well as from a priori *knowledge* and *intelligence* of a reasoning observer and, additionally, is *accumulated* over the time, and *revised* by new events. So, being widely unsolved so far,

* This work has been supported by the Deutsche Forschungsgemeinschaft (German Research Council) under grant SFB 876/A5 within the framework of the Collaborative Research Center "Providing Information by Resource-Constrained Data Analysis".

the challenge is to enable an individual or an enterprise, respectively, an *owner* agent for short, to exercise a *holistic control* over the information conveyed to communication partners by means of transferred data.

Notably, the direct *objects of control* are not the partners, which are seen as independent and autonomous entities, but the information owner's messages made observable to others. Thus the topic is disciplined *self-restriction* regarding a specific aspect of the owner itself. This aspect is its *belief*, as virtually derived from actual data sources by reasoning, about matters of joint interest. An unrestricted partner can watch "reality" and access other sources by himself and thus gain information about those matters (including "external" properties of the owner) in potentially many ways; but, basically, the owner's "internal" belief has to be learnt solely from the owner's behavior.

While we see the partners learning belief as an agreed primary goal of cooperation, and thus *permissions* to share data as the default, we argue that nevertheless the owner might want to hide specific pieces of belief to selected partners as an exception, declared as dedicated *prohibitions* in some security policy. Thus, regarding that policy, the owner perceives a partner as a semi-honest *attacker* agent, potentially aiming to gain more information than wanted. Having no control on the attacker, the owner has no other chance than proceeding as follows: before releasing any data, it has to explore the attacker's options to exploit the data to infer information to be kept hidden, and to block all such options by *filtering* and *modifying* the data appropriately. Clearly, any such simulation has to be based on convincing but in principle unverifiable *assumptions* about the attacker. In other words, the owner has to perform *adversarial reasoning*.

Finally, interactions between the owner and a partner might occur during a longer *period of time* and include any kind of *program execution*. These properties demand for keeping track of all information previously released and considering not only direct data flows but also implicit information flows caused by the control flow during a program execution. Thus, all reasonings about an attacker's gain of pieces of belief have to be *state-dependent*, both in long and short terms.

The discussion above points to the inherent difficulties of the challenge:

- the *targets* of protection are pieces of the holder's virtual integrated belief;
- the *requirements* of protection imply the need of an attacker assumption;
- the *application field* of protection is history-aware and procedure-oriented;
- any *mechanism* of protection has to be based on three kinds of reasoning, for the holder's integrated belief, for the attacker's inference options, and for tracing the overall history and tracking a single control flow.

Though admitting the impossibility of a general approach, in this report we will present a framework to construct solutions for instantiations of a narrower scenario. We describe their flavor and requirements in Section 2 and Section 3, respectively, leading to a mediator-based design. In Section 4 we outline the suggested kind of an inference control mechanism, which applies language-based information flow control with declassification based on controlled interaction execution, under some restrictions on the expressiveness of the programming constructs involved. The achievements are discussed in the concluding Section 5.

2 Scenario and Overall Problem

While processing *interactions* with cooperation *partners*, the information *owner* wants to employ inference control to confine the *information* content of *data* delivered by *external actions*, which might be responses to *perceptions* including explicit requests. More specifically, such an *inference control* aims at disabling a cooperating receiver to gain a piece of information that the owner wants to keep *secret*, as declared in the owner's security *policy*. The required hiding effect should be achieved solely by *filtering* and *modifying* the interaction data according to meaningful *assumptions* about the receiver which is then seen as an intelligent and only semi-honest "attacker" *reasoning* about both data perceived over the time and further *background knowledge*.

The owner's assumptions might refer to a large variety of the attacker's capabilities, ranging from the kind and the extent of the attacker's background knowledge over the attacker's *means* and *preferences* for reasoning to the attackers's *awareness* of the owner's configuration of both the underlying *information processing system* and the employed *security mechanism* for inference control. In defending, the owner exploits these assumptions for *simulating* possible behavior of the attacker, and for then blocking a harmful one by appropriately distorting external actions.

The scenario sketched above can have diverse instantiations which might vary in several dimensions, e.g.: an information "owner" might be an individual, a group of them, or even an enterprise; "information" might refer to facts of an objective world or to somebody's belief about such facts; an owner's information "basis" might be a single relational database, a collection of diverse data sources or a dedicated virtual view derived from a federation of heterogeneous data sources; interaction "processing" might be simple generation of answer relations to database queries, more involved treatment of update and revision requests, or even execution of programs written in a general programming language; the owner's overall "system" including the control mechanism might be monolithic or based on loosely coupled components; the "interactions" with the devices of the cooperation partners might be based on shared storage or message exchanges.

Obviously, each dimension will have an impact on the general design and various more specific aspects of the anticipated inference control. Elaborating seminal proposals for inference control in information systems, and being in the spirit of many similar approaches to secrecy as concisely exposed in [14], previous work on Controlled Interaction Execution, CIE, originally only dealt with a single owner of a logic-oriented database system like a relational one solely employed for querying, and later also included updates and non-monotonic belief management, see the summaries [4,5], and the abstraction [6]. Based on that, and grossly summarized, in our vision [7] we proposed an architecture of an inference control front-end for uniformly shielding a possibly heterogeneous collection of data sources.

Further expanding on and extending that vision, in the present article we will consider an instantiation of the scenario of the following kind:

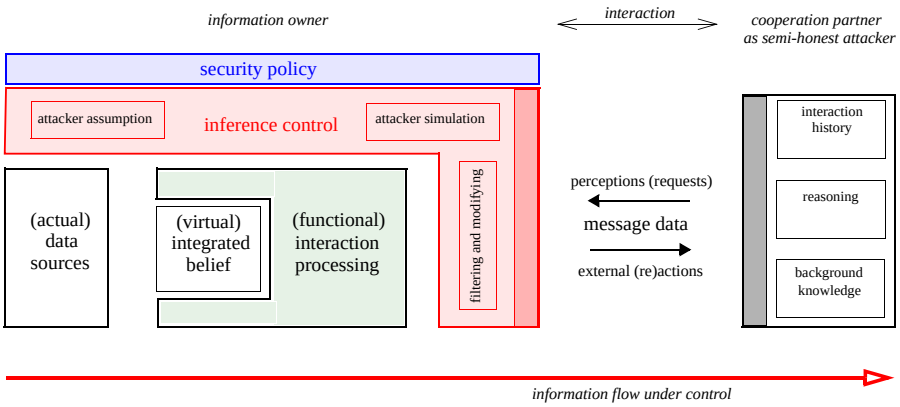

Fig. 1. Inference control for the focused scenario showing only one cooperation partner

- the owner might be *any subject* being willing and able to formulate a *coherent security policy* regarding a cooperation partner;
- data is representing information in the form of (subjective) *belief*, possibly including (objective) *knowledge* as a special case;
- the belief is derived as a dedicated *virtual integrated view* from a *federation* of existing heterogeneous data sources;
- interaction processing in terms of the virtual integrated belief might be governed by a *Java-like program* that complies with some syntactic restrictions, actually operating on the underlying heterogeneous data sources;
- the owner's system is understood as a subpart of a *multiagent system*;
- the cooperation partners' systems are also seen as *intelligent agents*;
- interactions between agents are solely based on *messages* representing *perceptions* and *actions*, including *requests* from and *reactions* to a partner's agent, respectively.

For such an instantiation we will focus on controlling the information flow

- from the owner's actual data sources
- over the owner's virtual integrated belief derived from the sources
- via the (re)action data transmitted by messages
- to a partner's system
- as caused by the owner's interaction processing.

More specifically, we consider the following problem of inference control: How to analyze and to block the partner's opportunities for successful inferences of pieces of belief the owner wants to keep confidential, as roughly visualized in Figure 1? This problem in particular includes the following subproblems: How to form the virtual integrated belief whose pieces are the main targets of protection? And how to treat not only explicit direct information flows conveyed by message data and indirect ones due to evaluating expressions but also implicit information flows caused by guarded commands in the protocols?

3 Requirements within the Agent Paradigm

Both *advanced interaction processing* alone and sophisticated *inference control* alone require autonomously performed intelligent behavior: formal reasoning in some appropriate logic, pro-active pursue of declared high-level goals, and decision making guided by an accepted norm [1]. Moreover, computational *belief management* alone demands for intelligent engineering. Consequently, for the combined task under consideration these requirements are mandatory, too.

Local autonomy and intelligence together with global cooperation based on messages are fundamental issues of the agent paradigm [17]. Accordingly, we will present our *requirements* mostly in terms of this paradigm, subsequently aiming to design the *architecture* and *selected components* of a *mediator agent* meeting our requirements. The requirements will be explained in three layers, from the underlying belief management over the functional interaction processing to the comprehensive inference control. The explanations will be informal in style.

3.1 Basic Belief Requests for the Virtual Integrated Belief

Assuming a collection of possibly *heterogeneous data sources*, we require that all interactions with a specific cooperation partner are based on a dedicated *unified view* on the data sources, forming an *integrated belief*, continuously maintained by the owner [10,12]. This strict requirement reflects the need to protect information independently from its data representation and its physical storage. Furthermore, information to be hidden should refer to the owner's presumably consistent belief on the matters of an interaction, rather than on the storage state of the underlying data sources or some "facts of the real world". To handle different views dedicated for specific partners, such a belief should not be fully materialized but only be *virtually derived*, as far as possible and convenient.

Leaving open the status of the underlying data sources and the channels to access them, we prefer to derive a virtual integrated belief following a *mediation* approach, as visualized in Figure 2. Each actually materialized data source is *wrapped* to be seen as a *local knowledge base*, comprising both its own *background knowledge* and *evidential knowledge* as well as *meta knowledge* regarding the mediation framework, and exporting a *local belief*. The *virtual integrated belief* is then formed by means of a *knowledge base as a mediated schema*, being equipped with background knowledge (about the intended application) and meta knowledge (regarding the mediation framework), and further components to handle a *basic belief request*, in particular a query, an update or a revision. At suitable commit points in time, this approach has to satisfy a *fundamental invariant*, namely that the virtual integrated belief "correctly reflects" the materialized sources.

A request is handled by *translating* it into subrequests directed to the underlying sources over a *communication network* via their *wrappers* and then *combining* the subreactions to an overall *basic belief reaction*. The further components for *request handling* are briefly explained by outlining the expected conceptual control flow and data flow for a basic belief request, which we suppose to be

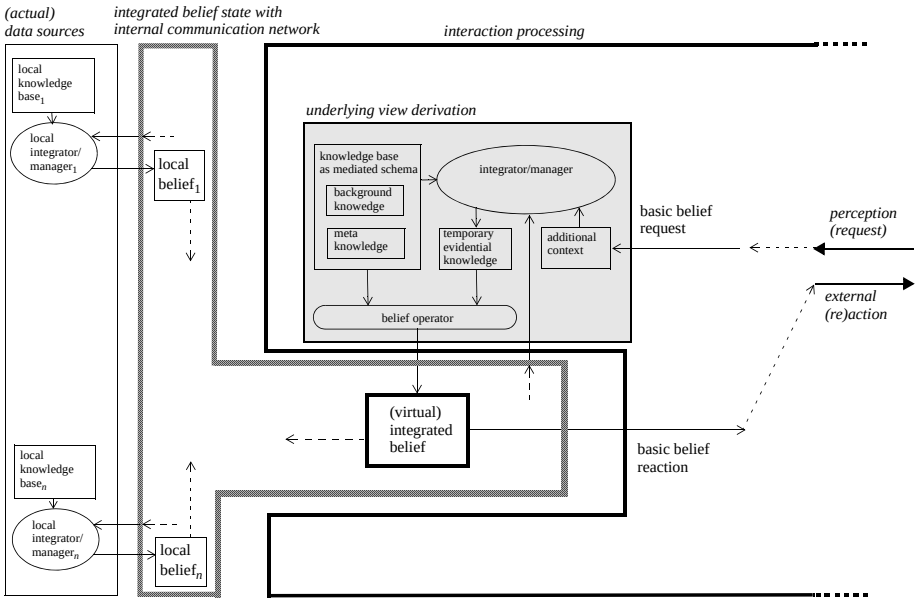

Fig. 2. Forming a virtual integrated belief and executing basic belief requests

submitted by the remaining parts of the interaction processing possibly based on perceptions. Depending on the kind of request, first some so-called *additional context* is generated that is then forwarded to the *integrator/manager* to inform it about the details of the request. The integrator/manager also has access to the integrated belief state assumed to satisfy the fundamental invariant, and to the knowledge base as mediated schema. From these three inputs, the integrator/manager generates appropriate *temporary evidential knowledge* that in turn, together with the knowledge base, is employed by the *belief operator* to form a relevant part of the virtual integrated belief, more precisely to return a pertinent *basic belief reaction* and, if applicable in case of changes, to sent appropriate *materialization requests* to the sources, in particular in order to re-establish the fundamental invariant.

3.2 Advanced Interaction Processing

Interaction processing actually performs the information owner's part of the intended cooperation with another partner and, thus, is dependent on the concrete application. Accordingly, we imagine that a *Java-like program* for some kind of a cooperation protocol is executed. Following the agent paradigm [17], we exemplify our requirements by considering the typical *looping behavior* of a rational reasoner that is *querying and maintaining belief*, *pro-actively planning* while pursuing *goals*, and *making decisions* while *respecting norms* [1], as visualized in Figure 3. Such a behavior shows the following characteristic properties:

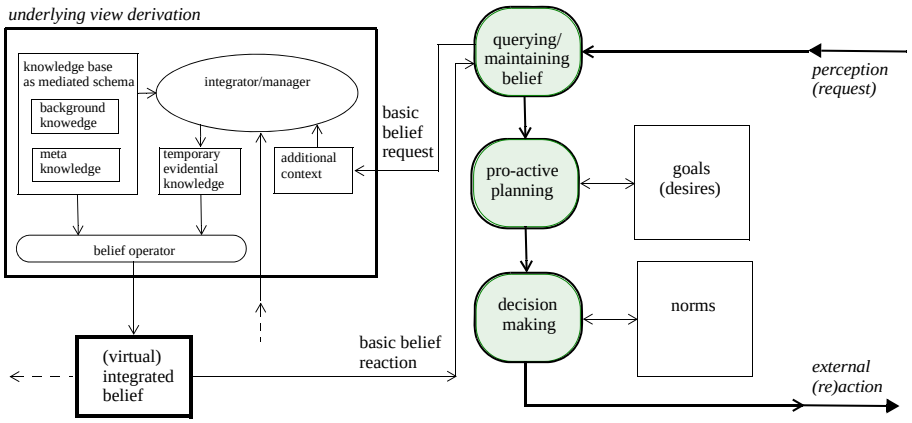

Fig. 3. Exemplified interaction processing (without inference control)

a sequence of uniformly repeated history-aware activities, branchings dependent on a possibly changing belief, submission of basic belief requests, and branchings within the execution of a basic belief request. So we require inference control of interaction processing summarized as follows:

- started with the (hidden) **input** of an *initial belief* and
- supplied with a further (open) **input** of a *sequence of perceptions*, received via a dedicated interface,
- *performing a loop*, for each of the perceptions the interaction processing **system** uniformly *executes a body* that
  - might contain *branchings* and involves calls of *basic belief requests* whose *reactions* are communicated via a *belief–program interface*, and
  - finally generates an action, sent via a dedicated interface,
- resulting in an overall (open) **output** of a *sequence of actions*.

3.3 Inference Control

For each of the information owner's cooperation partners, interaction processing (as outlined in Subsection 3.2) based on a (virtual) integrated belief (as sketched in Subsection 3.1) should be *generally permitted*, possibly somehow restricted according to granted *access rights*. However, each actual execution should also be continuously monitored in order to block the respective partner's options to infer pieces of belief the owner wants to keep secret to that partner. Accordingly, a partner-specific *security policy* expressed in terms of belief sentences should declare *prohibitions* as exceptions from the general permission.

The policy only specifies *what* should be kept confidential and leaves open *how* confidentiality will be enforced. For an enforcement we aim at a two-layered approach of employing existing concepts:

– language-based *information flow control* with declassification for the Java-like implemented interaction protocol
– based on *inference-usability confinement* with filtering and modification by CIE for the logic-oriented belief management.

Moreover, we want to apply a suitable variant of CIE-like *semantics* of a security policy for the overall system, see [5]. Informally expressed, the wanted semantics basically requires the following for each belief sentence (traditionally called a *potential secret*) in the security policy for a specific cooperation partner:

> the *actual validity* (held to be true in the owner's integrated belief) should be *kept secret* to the attacker (though being a cooperation partner), whereas an actual negated validity (held to be false in the owner's integrated belief or having an undecided status) may be learnt.

The sketched semantics can equivalently be expressed in terms of either "absence of knowledge" – the attacker should *not know* the potential secret's *validity* – or "ensurance of possibility" – the attacker should be *sure about* the *possibility of the potential secret being not valid*. We also could distinguish whether a confidentiality requirement should refer to either the *initial* status of a potential secret or its *current* status or its full status *history* or, in some extension, even to any selection of *points in time* of its history.

The meaning of phrases like "kept secret to", "not knowing", or "being sure" applied to an intelligent attacker strongly depends on its background, in particular its *a priori knowledge* about the integrated belief, its *awareness* about details of the controlled interaction system, and its *reasoning* methods [7]. However, in general the (defending) owner can only form reasonable *assumptions* about the (attacking) partner's background. So we require that the mediator for the owner should be able to *simulate* the partner agent's behavior in attempting to infer belief to be kept secret according to *partner-specific parameter* values, which are provided via an *administration interface*. This simulation should be based on a *conservative* approach to security, considering a kind of *worst case* regarding the *general part* of the attacker's background: the mediated schema and, thus, the expressive means of declaring potential secrets; the security policy actually employed in the interaction; the owner's kind of forming and changing the integrated belief, i.e., the syntax and semantics of basic belief requests; the owner's Java-like program for the overall looping of interaction processing, and, additionally, the owner's confidentiality enforcement mechanism to be designed for the required inference control.

Taken all together, the wanted protection should be achieved against an attacker that knows the overall programming code of controlled interaction processing based on CIE including its parameter values, and thus the *complete system definition* in terms of *traces* associated with the respective *inputs* and *outputs*. Under this perspective, the semantics of a security policy can be rephrased in terms of the system definition by traces like a *non-interference property* [16].

4 A Mediator Framework for Unified Inference Control

In this section we outline a mediator framework to comply with the requirements developed in Section 3. The framework applies language-based information flow control with declassification supported by controlled interaction execution. Rather than establishing a single inference control mechanism for a specific interaction protocol programmed in Java, as done in previous work, e.g., [8], we more ambitiously aim at providing a *general framework* to implement any inference-proof interaction protocol of the kind described in Subsection 3.2. The main goal of the framework is to *uniformly* and *securely relate* CIE-like inference control regarding logically expressed pieces of belief on the one hand to information flow control regarding data objects stored in containers and manipulated by means of Java commands like evaluation of expressions, assignment and procedure call, or conditionals (guarded commands) on the other hand.

For lack of space, we will present the framework by summarizing the nine main steps of the line of reasoning for its justification and by discussing an informative example. A detailed exposition of a full abstract model and a formal verification of the achievements are left to a forthcoming report, complemented by an implementation employing Paragon [11].

Paragon is a Java-based programming language supporting *static checking* of a specification of permitted information flows and offering *dynamic declassification* [16] by means of current states of *flow locks*. Roughly summarized, a programmer should associate a *flow policy* of form $\{receivers : list\_of\_flowlocks\}$ to an *object* declaration, and in the program code he might employ operations to *open* or *close* declared flow locks.

The Paragon *compiler* then checks whether (the semantics of) all commands appearing in the program code comply with the (semantics of the) declared policies in the following sense: in any program execution, the information content of an object can only flow to the intended receivers under the condition that the associated flow locks have been opened. Accordingly, the *runtime* system of Paragon only maintains and checks the lock states.

This implies that a Paragon *programmer* has to take care that all declassification relevant procedures are appropriately *encapsulated* such that the wanted effects cannot unintendedly or maliciously be modified. In particular, the programmer has to safely protect any code that leads to open flow locks after checking some conditions. In our suggested framework, this code would be concentrated in a *belief–program interface* that contains a *flow tracker* as its main component. Accordingly, this interface has to be generically provided as part of the general framework such that an application-oriented programmer of an interaction protocol cannot modify or circumvent the security enforcing procedures.

4.1 Overall Design of the Framework

To provide the general framework, we have to define appropriate *structures*, to be implemented as *Java/Paragon classes*, and to describe a corresponding *programming discipline*. In the following we will introduce these items by a brief

summary of the line of reasoning that justifies our approach, complemented by a partial visualization shown in Figure 4.

1. *Isolation by typing.* The working space of an interaction protocol has to be strictly divided into a *protected realm* where potentially confidential information stemming from basic belief reactions is processed and an *open realm* where the final external reaction is prepared. We will employ *typing* with *flow lock policies* to achieve a complete (but preliminary) isolation of these realms.

2. *Sharing by declassification.* To nevertheless enable *discretionary sharing* of information about the integrated belief, i.e., a controlled information flow from a container in the protected realm to a container in the open realm, we will use *declassification* by means of temporarily *opening* the flow lock on the source container.

3. *History-aware policy compliance by FlowTracker and CIECensor. Before* temporarily opening a flow lock to permit a requested data transfer, we have to ensure that the transfer would be harmless, i.e., complying with the *security policy* under the simulated history-dependent *previous view* of the attacker on the owner's integrated belief. For this complex task, we will provide a dedicated encapsulated component called *FlowTracker* which can delegate subtasks, namely (1) evaluations of harmlessness and (2) actions of filtering and modifying, to a further component called *CIECensor*.

4. *Need of local flow tracking.* For an *evaluation of harmlessness* of a requested data transfer, the *CIECensor* needs to know both the explicit and the implicit information about the integrated belief contained in the source container as the result of the preceding processing in the protected realm. I.e., the *CIECensor* needs an additional input, namely the view on the owner's integrated belief, solely resulting from preceding processing and to be tentatively combined with the attacker's previous view. This *tentative addition* will be dynamically generated by the *FlowTracker*.

5. *Identifying implicit flows by symbolic executions.* Implicit information – as caused by guarded commands – not only depends on the actually performed execution path but also on the possibly followed (alternative) paths – which could be selected for different values of the pertinent guards. Accordingly, the *FlowTracker* will base the dynamic generation of the tentative addition on *code annotations* that stem from *symbolic program executions* performed at compile time, as inspired by [2].

More specifically, as part of the *programming discipline*, a data transfer from the protected realm to the open realm may only be requested by an explicit *assignment* command, or any equivalent like a procedure call, that is *outside the scope* of any guard. For such a command, *code analysis* will perform symbolic executions of *all* paths leading to the pertinent command, and combine their results to an annotation which contains a symbolic expression referring to the usage of a container in a path by a (still uninterpreted) symbol and denoting the information content of the source container regarding the integrated belief.

6. *Determining local flows by FlowTracker.* To prepare for manipulating the flow lock, the *FlowTracker* will evaluate the symbolic code annotation using the ac-

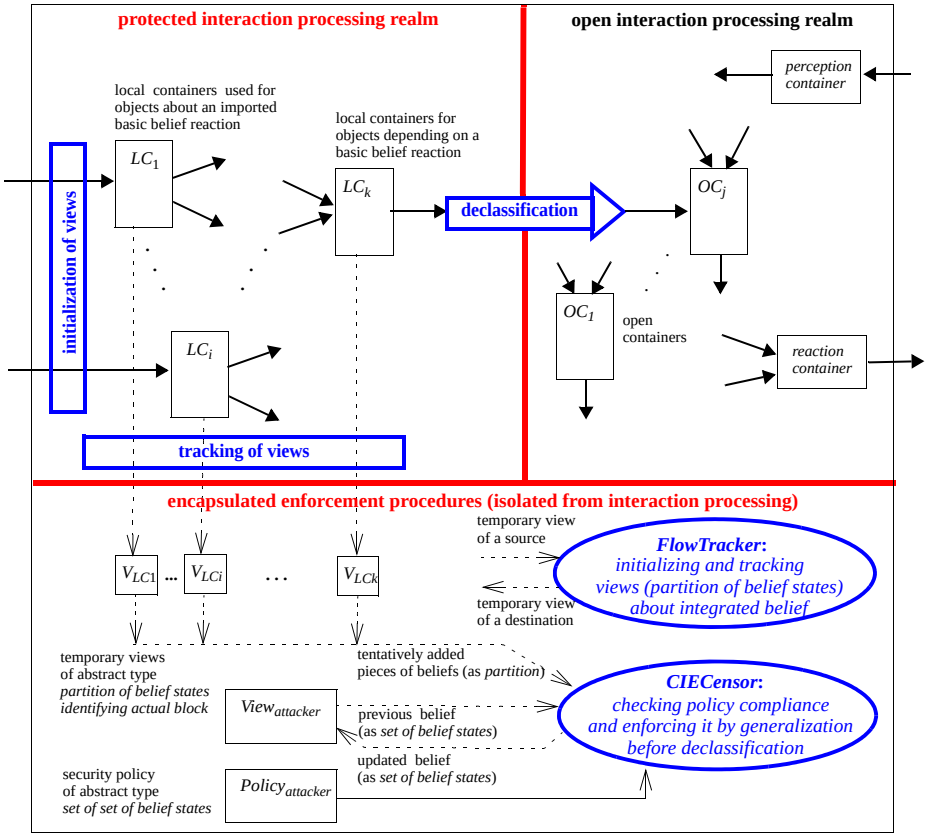

Fig. 4. Concepts of the belief–program interface underlying the mediator framework

tual information contents of the local containers involved. These information contents are dynamically determined as *temporary views* associated with the local containers. More specifically, these temporary views are *initialized* when storing a basic belief reaction as an object in a local container and for an evaluation further *tracked* according to the operators in the code annotation. In abstract terms, such an information content is represented as a mapping of possible values to sets of belief states such that all these sets together form a *partition* of the possible belief states, similarly as in [6]. Moreover, the mapping is complemented with an identification of the *actual value and block*, respectively.

7. *Evaluation of harmlessness by CIECensor.* Called by the *FlowTracker* with the tentative addition, the *CIECensor* checks whether the *combined information content* of the tentative addition resulting from preceding processing and the previous view resulting from the history could possibly violate the *security policy*. In abstract terms, this combination is obtained by taking all nonempty intersections of a block in (the partition of) the tentative addition with the pre-

vious view, and a *possible violation* occurs if there is a block in the combination that is *completely contained* in an element of the security policy, assuming that a policy element is abstractly represented as a set of belief states, as described in [6]. If the check confirms harmlessness, the previous view is updated to its intersection with the actual block and the *FlowTracker* is informed accordingly, and the *FlowTracker* then temporarily opens the flow lock in turn.

8. *Filtering and modifying by generalization.* If the check indicates a possible violation, the *CIECensor* considers whether there would be an *actual violation*. Clearly, this is the case if the block in the combination that corresponds to the actual value is contained in a policy element, and thus this value may not be revealed. However, to avoid meta-inferences such a hiding has to be made *indistinguishable* from the treatment of at least one different value. Accordingly, the *CIECensor* has to apply a precomputed *distortion table*, as exemplified in [9], that (i) clusters possible values such that the union of their blocks is not contained in any policy element and (ii) determines for each cluster a suitably *generalized value*, similarly as for k-anonymity. Applying the distortion table then means that the *CIECensor* updates the previous view with the partition derived from the clustering and returns the generalized value determined by the cluster containing the actual value to the *FlowTracker*, which then replaces the actual value of the source container by the generalized one and finally temporarily opens the flow lock.

As part of the *programming discipline*, the set of possible return values of a basic belief reaction and the set of (crucial) values of external reactions (depending on the integrated belief state) has to be kept suitably small to manage the precomputation of the distortion table.

9. *Processing generalized values by overloading operators.* To enable further processing of generalized values within the *open realm*, all operators used in the pertinent program code have to be suitably *redefined* by overloading.

So far, to enable manageable procedures, we rely on two restrictions. First, only *closed queries* are accepted as basic belief requests (but no open queries, updates and revisions), since they have only two or a few more return values (e.g., corresponding to *valid*, *invalid*, *unknown*, ...) and can be handled without a sophisticated transaction management including commits and aborts. Second, only two-sided conditionals are allowed in a program for interaction processing (but no repetitions with dynamically determined number of rounds), to enable the symbolic program executions according to Item 5. Moreover, as already indicated before, the effectiveness of the proposed procedures crucially depends on the suitable protection of the *FlowTracker* and the related components, which might restrict the usage of some further programming features.

Based on an assignment of pertinent flow policies to basic belief reactions and the reaction container, type-checking the interaction program by Paragon will ensure isolation of the two realms and, by exception, sharing via declassification as permitted by the lock state. Additionally, write-access to the necessary locks can be limited to the *FlowTracker* class by using Java access level modifiers.

Regarding the CIE-like semantics of the security policy (Subsection 3.3), we will prove that the attacker's information gain is limited, locally, to explicit assignments across the realms and, in extent, to the information content of the possibly generalized value so assigned. The proof will be based on the gradual release property, to be established by Paragon, and an extension to conditional gradual release [3], to be established by symbolic execution for local flow tracking.

4.2 An Informative Example

We will discuss the following abstract, imperative interaction program:

$\underline{\mathsf{c}_1} := val_2$; $\mathsf{c}_2 := val_2$;
$\underline{\mathsf{c}_3} := \mathsf{breq}_1(\mathsf{ibs}, \mathsf{c}_{\mathsf{per}})$; $\underline{\mathsf{c}_4} := \mathsf{breq}_2(\mathsf{ibs}, \mathsf{c}_{\mathsf{per}})$; $\underline{\mathsf{c}_5} := \mathsf{breq}_3(\mathsf{ibs}, \mathsf{c}_{\mathsf{per}})$;
if $\mathsf{q}_1(c_{per})$ then
  if $\mathsf{q}_2(\underline{c_4})$ then $\underline{c_1} := \oplus_1(\underline{c_3}, \underline{c_5})$ else $\underline{c_1} := \underline{c_3}$ endif
endif
$\mathsf{ic}(\underline{c_1}, c_6)$; $\mathsf{ic}(\underline{c_3}, c_7)$;
$\mathsf{c}_{\mathsf{rea}} := \oplus_2(\mathsf{c}_6, \mathsf{c}_2, \mathsf{c}_7)$

Data from the integrated belief state *ibs* is imported into containers c_3, c_4 and c_5 via basic belief requests depending on the perception container c_{per}. Consequently, according to Item 1 (Isolation), these containers belong to the protected realm as indicated by their underlining and determined by the not represented flow policies of the respective basic belief reactions and so does the container c_1 due to its dependence on basic belief reactions. The remaining containers c_2, c_6 and c_7 belong to the open realm, contributing to the preparation of the reaction container c_{rea}, as determined by its not represented flow policy. To transfer data from containers c_1 and c_3 in the protected realm to the open realm into containers c_6 and c_7, respectively, the programmer employs the assignment ic controlled by the *FlowTracker* for declassification according to Item 2 (Sharing).

According to Item 5 (Implicit Flows), dependencies of c_1 on basic belief reactions in execution paths, which lead through the two guarded commands to the declassification command for c_1, are represented in a precomputed symbolic expression for c_1. In the precomputation, uninterpreted symbols are introduced for each basic belief reaction. Following all paths, symbolic expressions for c_1 are generated for each assignment to c_1 in the scope of the guarded commands which are $\oplus_1(\alpha, \gamma)$ (for the operation $\oplus_1$ dependent on the basic belief reactions $\mathsf{breq}_1(ibs, c_{per})$ and $\mathsf{breq}_3(ibs, c_{per})$, respectively) and α (for the basic belief reaction $\mathsf{breq}_1(ibs, c_{per})$). The visit of a branch conveys information about the validity of the guard as denoted by the symbolic expressions q_1 and $\mathsf{q}_2(\beta)$, respectively (the latter dependent on the basic belief reaction $\mathsf{breq}_2(ibs, c_{per})$). At the join of two complementary branches, the symbolic expressions assigned to c_1 in each branch are refined with the respective expression for the validity or falsity of the enclosing guard and then combined for the join. Here, leaving the branches of the inner guard, the symbolic expressions for c_1 of each branch are $\mathsf{q}_2(\beta) * \underline{\oplus_1(\alpha, \gamma)}$ and $\overline{\mathsf{q}_2(\beta)} * \alpha$, respectively, joined to the expression $(\mathsf{q}_2(\beta) * \oplus_1(\alpha, \gamma)) + (\overline{\mathsf{q}_2(\beta)} * \alpha)$.

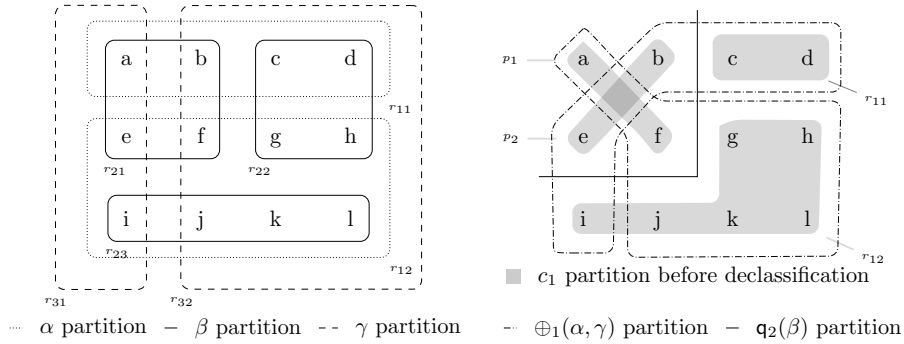

Fig. 5. Indexed partitions of the set of belief states for local flow tracking

Leaving the scope of the outer guard, the symbolic expression for c_1 becomes $(\mathsf{q}_1 * ((\mathsf{q}_2(\beta) * \oplus_1(\alpha,\gamma)) + (\overline{\mathsf{q}_2(\beta)} * \alpha))) + \overline{\mathsf{q}_1}$ as used for the declassification.

Assume that the integrated belief state *ibs* is an element from the abstract set $\mathcal{IBS} = \{a, b, \ldots, l\}$ and that the input to the perception container c_{per} is fixed. Then, dependent on the actual choice of *ibs*, the basic belief reactions may vary within small ranges R_1, R_2 and R_3 of values and partition the set $\mathcal{IBS}$ according to the value (Figure 5 on the left). According to Item 6 (Tentative Addition), the *FlowTracker* initializes the symbols for the respective belief reactions and the temporary views of the containers into which these are imported to the respective partition. A view $(w, (B_r)_{r\in R})$ for container c means the following: knowing the value of c is r enables the partner to infer to which block the actual integrated belief state belongs, namely B_r. Note that the partner is always able to determine the partition from his background and the fixed open input.

Term-wise, being synchronized with interaction processing by (not represented) code annotations, the *FlowTracker* evaluates pertinent subterms of the symbolic expression for container c_1 yielding an evaluation of the whole expression to be used as the tentative addition during declassification (Item 6), as shown in Figure 5 on the right. Being informed about the return value of the boolean function q_1, the *FlowTracker* extracts the subterm of the symbolic expression according to the validity of q_1 and discards the rest. For the subterm built with symbol $\oplus_1$, it first computes the lower bound of the two input partitions, α and γ, maps the index of each block to the result of operator $\oplus_1$, and unions all blocks with the same index. Lastly, for the boolean function q_2 dependent on the partition β, the *FlowTracker* unions all blocks B_r from β with the same value $\mathsf{q}_2(r)$, leading to two blocks, and intersects the partitions for the subterms $\oplus_1(\alpha,\gamma)$ and α with the pertinent block. This way, the *FlowTracker* obtains the tentative addition for the declassification of c_1.

Consider now the security policy $\{\{a\}, \{g, h\}\}$, the actual integrated belief state $e \in \mathcal{IBS}$ and the history-dependent previous view containing $\mathcal{IBS} \setminus \{c, d, f\}$. Regarding Item 3 (Compliance), before the transfer from c_1 to c_6

via the declassification command ic, the *CIECensor* checks whether a set in the security policy comprises a block from the tentative addition for c_1 intersected with the previous view according to Item 7 (Harmlessness). The check is positive for policy element $\{a\}$ so that policy violation is possible and the *CIECensor* proceeds with generalization according to Item 8 (Generalization). In the distortion table, it first looks up the highlighted row as determined by the view of c_1 intersected with the previous view and the security policy and then it looks up the column for the actual value p_2 of c_1:

sets of indices per harmful union of blocks	actual container value			
	p_1	p_2	r_{11}	r_{12}
$\{p_1\}$	p	p	r_{11}	r_{12}
$\{p_2\}$	p	p	r_{11}	r_{12}
$\{p_1, p_2\}$	#	#	#	#
...				

The so looked up cell contains the value to be transferred to container c_6. Accordingly, the *CIECensor* updates the view for the partner to the set $\{a, b, e\}$ and then forwards the value p to the *FlowTracker*.

5 Conclusions

As already mentioned in the introductory Section 1, we started a highly ambitious project to construct solutions for the inherently difficult challenges of a uniform information confinement of protocols for interaction processing based on an integrated belief. Briefly summarized, we have had to combine adapted technologies from various fields of expertise, including logic-based belief forming [10], adversarial reasoning [15], information system integration and mediation [12], data–program interfacing [13], logic-based inference control [4], and language-based information flow control with declassification [16], for each of which alone the state of science already provides impressive achievements.

So, the conjecture underlying our project is that time is come to construct a novel kind of systems to manage and share information while discretionarily protecting dedicated pieces of it. Our main contribution, a framework for designing and implementing such a system by employing available technologies from the various fields, supports our conjecture by a technical argument. The technicalities employed consist of interfacing data sources and programming code not only at the syntactic layer of the data's bytes but on the semantic layer of the data's information content in terms of the owner's integrated belief, actually even more striving in terms of (a simulation of) the attackers's view on that belief.

Obviously, we have only exhibited one possible approach for the indicated scenario; in other scenarios and under other circumstances alternative approaches might be possible as well. However, we believe that bridging the gap between the underlying raw data and the effectively released information will be at the core of any attempt. Moreover, we still have to experimentally evaluate the practicality of several mediators constructed by employing a future prototype implementation of the framework. As usual, acceptable computational complexity and scalability will demand appropriate approximations and some simplification.

References

1. G. Andrighetto, G. Governatori, P. Noriega, and L. W. N. van der Torre, editors. *Normative Multi-Agent Systems*, volume 4 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
2. M. Balliu, M. Dam, and G. L. Guernic. Encover: Symbolic exploration for information flow security. In S. Chong, editor, *IEEE Computer Security Foundations Symposium – CSF 2012*, pages 30–44. IEEE Computer Society, Los Alamitos, 2012.
3. A. Banerjee, D. A. Naumann, and S. Rosenberg. Expressive declassification policies and modular static enforcement. In *IEEE Symposium on Security and Privacy – S & P 2008*, pages 339–353. IEEE Computer Society, Los Alamitos, 2008.
4. J. Biskup. Inference-usability confinement by maintaining inference-proof views of an information system. *International Journal of Computational Science and Engineering*, 7(1):17–37, 2012.
5. J. Biskup. Logic-oriented confidentiality policies for controlled interaction execution. In A. Madaan, S. Kikuchi, and S. Bhalla, editors, *Databases in Networked Information Systems – DNIS 2013*, volume 7813 of *LNCS*, pages 1–22. Springer, Berlin Heidelberg, 2013.
6. J. Biskup, P. A. Bonatti, C. Galdi, and L. Sauro. Optimality and complexity of inference-proof data filtering and CQE. In M. Kutylowski and J. Vaidya, editors, *Computer Security – ESORICS 2014*, volume 8713 of *LNCS*, pages 165–181. Springer International Publishing, 2014.
7. J. Biskup and C. Tadros. Idea: Towards a vision of engineering controlled interaction execution for information services. In J. Jürjens, F. Piessens, and N. Bielova, editors, *Engineering Secure Software and Systems – ESSoS 2014*, volume 8364 of *LNCS*, pages 35–44. Springer International Publishing, 2014.
8. J. Biskup and C. Tadros. Preserving confidentiality while reacting on iterated queries and belief revisions. *Ann. Math. Artif. Intell.*, 73(1-2):75–123, 2015.
9. J. Biskup and T. Weibert. Keeping secrets in incomplete databases. *Int. J. Inf. Sec.*, 7(3):199–217, 2008.
10. G. Brewka. Multi-context systems: Specifying the interaction of knowledge bases declaratively. In M. Krötzsch and U. Straccia, editors, *Web Reasoning and Rule Systems – RR 2012*, volume 7497 of *LNCS*, pages 1–4. Springer Berlin Heidelberg, 2012.
11. N. Broberg, B. van Delft, and D. Sands. Paragon for practical programming with information-flow control. In C. Shan, editor, *Programming Languages and Systems – APLAS 2013*, volume 8301 of *LNCS*, pages 217–232. Springer International Publishing, 2013.
12. A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, San Francisco, 2012.
13. M. Fowler. *Patterns of Enterprise Application Architecture*. Pearson, Boston, 2003.
14. J. Y. Halpern and K. R. O'Neill. Secrecy in multiagent systems. *ACM Trans. Inf. Syst. Secur.*, 12(1):5.1–5.47, 2008.
15. A. Kott and W. M. McEneaney, editors. *Adversarial Reasoning: Computational Approaches to Reading the Opponent's Mind*. Chapman & Hall/CRC, London, 2007.
16. A. Sabelfeld and D. Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.
17. M. J. Wooldridge. *An Introduction to MultiAgent Systems (2. ed.)*. Wiley, Chichester, 2009.