A Layered Architecture based on Previsional Mechanisms

Francesco Fiamberti, Daniela Micucci, Marco Mobilio and Francesco Tisato
Department of Informatics, Systems and Communication
University of Milano-Bicocca, Viale Sarca 336, 20126, Milano, Italy

Keywords: Data Fusion, Software Architecture, Architectural Patterns.

Abstract: The paper presents a layered architecture that improves software modularity and reduces computational and communication overhead for systems requiring data from sensors in order to perform domain-related elaborations (e.g., tracking and surveillance systems). Each layer manages hypotheses that are abductions related to objects modeling the ”real world” at a specific abstraction level, from raw data up to domain concepts. Each layer, by analyzing hypotheses coming from the lower layer, abduces new hypotheses regarding objects at a higher level of abstraction (e.g., from image blobs to identified people) and formulates timed previsions about objects. The failure of a prevision causes a hypothesis to flow up-stream. In turn, previsions can flow down-stream, so that their verification is delegated to the lower layers. The proposed architectural patterns have been reified in a Java framework, which is being exploited in an experimental multi-camera tracking system.

1 INTRODUCTION

Nowadays, sensors are used in several application domains including surveillance, machines, medicine, and robotics. Usually such application domains rely on solutions that strongly depend on both the sensing devices and the application domain of the problem. Moreover, they usually adopt data-driven policies: a bottom-up process abstracts domain entities from raw data and the whole processing chain is traversed every time new information is available.

This results in huge and monolithic software systems presenting several drawbacks: lack of modularization and stratification (reuse is difficult or even impossible); complex configuration of the field installations (mostly implying some intervention on the software); difficult coexistence of different algorithms; computational overhead (e.g. in a surveillance system, image processing algorithms are performed from scratch for every acquired image); communication overhead (the whole chain is traversed each time); lack of timeliness (“reasoning” is performed only *after* something happened) and missing reactions (sometimes anomalous situations should be recognized when *nothing* is perceived).

To overcome such issues, this work addresses the problem from an architectural perspective, proposing ALARM (A Layered Architecture based on pRevisional Mechanisms), a layered architecture that is independent of the specific application domain and of the sensors' characteristics. This makes the architecture modular, scalable and open.

ALARM is based on the following basic concepts:

- The system is structured in layers. Each layer includes *objects* that model the “real world” at a specific abstraction level, from raw data up to domain concepts; objects are located in spatial and temporal reference systems. Each layer performs subsequent abstractions through abductive inference; the number and kind of layers is a domain-related issue.
- *Hypotheses* are timestamped abductive inferences based on objects;
- The “intelligence” of each layer relies on the formulation of *previsions* about objects and on their verification against hypotheses coming from lower layers;
- Previsions have a time-dependent validity, therefore their verification is *time-driven*;
- Failure of the verification process generates an exception that is notified to the upper layer (upstream flow). This is the only mechanism allowing information to flow to higher levels (exception-driven approach);
- A *line of thought* is a series of semantically linked previsions about the evolution of an object. Many

Fiamberti F., Micucci D., Mobilio M. and Tisato F..
A Layered Architecture based on Previsional Mechanisms.
DOI: 10.5220/0004592503540359
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-EA-2013), pages 354-359
ISBN: 978-989-8565-68-6

lines of thought related to the same object may be formulated;

- The verification of a prevision may be delegated to a lower abstraction level (downstream flow).

Since ALARM focuses on information transformation, it falls in the scope of data fusion, especially if applied in a heterogeneously instrumented environment.

Data fusion made its first appearance in the 1960s in the form of mathematical models aimed at data manipulation. In the subsequent years, it has been applied in various fields. Many generic models are described in the literature accommodating the specific applications at hand, but in each attempt the main motivation seems to to be the partitioning of the process into sub-tasks.

The model that obtained the highest accolade in the literature is the JDL Data Fusion Model (White, 1991) in its revised form made in (Steinberg et al., 1999), which has been chosen as the starting point for many data fusion applications in a wide range of domains (see for example (Bruckner et al., 2012; Jotshi et al., 2009; Hall et al., 2008)). Other widely used models are the Boyd Loop (Boyd, 1987), the Intelligence Cycle (Shulsky and Schmitt, 2002), the Behavioral Knowledge based Model (Pau, 1988) and the Waterfall Model (Harris et al., 1998), which is an approach close to the layering used here. More precisely, ALARM's layers, in a configuration as presented in the proposed case study, are addressable by the low levels of data fusion in the waterfall model. The JDL revised model, considering the further subdivision of the first level proposed in (Hall and Llinas, 1997) also finds a fine correspondence with the ALARM layering, which can then be freely used while maintaining consistence with the literature.

The paper is organized as follows. Section 2 describes the patterns at the basis of the conceptual model underlying the ALARM architecture. Section 3 discusses the results of preliminary tests of the proposed approach performed on a prototype implementation. Finally, in Section 4 some comments and possible future developments are presented.

2 CONCEPTUAL MODEL

This section introduces the conceptual patterns ALARM relies on.

2.1 The Abduction Pattern

Charles Sanders Peirce (Peirce, 1935) firstly introduced the term *abduction* as follows:

"Abduction is the process of forming an explanator *hypothesis*. It is the only logical operation which introduces any new idea."

Using Peirce words "while *deduction* proves that something *must* be and *induction* shows that something *actually is* operative, *abduction* merely suggest that something *may* be". This can be read as: deduction links premises with conclusions, induction constructs general propositions derived from specific examples, and abduction goes from data description to a hypothesis that tries to explain relevant evidence.

According to Pierce's definition, in ALARM *hypotheses* represent predicates about "real objects" at a specific level of abstraction. The abduction process translates hypotheses from a lower abstraction level to a higher one.

Such a behavior is at the basis of a *bottom-up* or *data-driven* model. It is exploited by most systems, but has several drawbacks:

- The high-level analysis of hypotheses may be computationally intensive and is performed at the last step of the abductive chain. This delays high-level decisions like the recognition of abnormal situations;
- An abnormal situation can derive from the absence of base-level hypotheses and data-driven models usually lack the capability of recognizing such an absence;
- The entire processing pipe is traversed each time a base-level hypothesis is perceived. This leads to both computational and communication overhead.

2.2 The Verification Pattern

Given the layered structure that performs abductions, the core idea in ALARM is to exploit hypotheses to produce *previsions*. A prevision is a predicate about one or more hypotheses that are expected to be valid in the future; in other words, previsions represent the expected evolution of an observed situation. While hypotheses are related to the past (they are produced when something happens), previsions are future-related facts.

The verification pattern is composed of two main activities: *forecasting* and *verification*. The former observes hypotheses and formulates previsions as an estimate about future behaviors; the latter is aimed at verifying the correctness of such previsions by comparing them against hypotheses.

The output of a verification process can be one of the following:

- **Match:** there exist a hypothesis and a prevision representing the same information;

- **Hypothesis without Prevision:** a hypothesis exists without a matching prevision;
- **Prevision without Hypothesis:** there is a prevision without a corresponding hypothesis;
- **Hypothesis / Prevision Incongruence:** there are both a prevision and a hypothesis, but they are in conflict with each other.

The key concept in handling these cases is that *the hypothesis course is driven by exceptions*, which means that hypotheses at a certain level l flow up to the level $l+1$ only if the verification at l produces a mismatch.

Mismatches are notified to the upper level as new hypotheses, thus maintaining coherence with the assumption that only hypotheses flow upward. A match does not produce any output; the case of a hypothesis without a prevision lets the same hypothesis flow upward and finally a prevision without a hypothesis produces a new hypothesis stating that the corresponding expected real object was not observed. As for the fourth possibility, it is not necessary to deal with it as an independent possibility, since it is actually a combination of the other two mismatch cases.

2.3 The Delegation Pattern

The verification pattern reduces the computational and communication overhead but still suffers from an architectural issue. In fact, in order to be able to formulate meaningful previsions, the low-level activities should embed domain-related knowledge. This would strongly affect system modularity and software re-use: in a well-structured architecture, low-level activities should not be aware of high-level issues, making it hard, or even impossible, to formulate helpful previsions at lower levels; rather, previsions at lower levels should come from higher levels, according to domain issues.

If the generation of previsions is restricted to higher levels only, the information flow between almost all the levels will increase, thus reducing one of the advantages offered by the approach. In order to solve this issue, one additional pattern is introduced: the *delegation* pattern.

The delegation pattern consists of delegating the verification of previsions to a lower level than the one at which they have been generated. It is composed of two different activities: *delegation* and *deduction*. The former selects the previsions to be delegated, while the latter is in charge of actually translating them to the corresponding lower-level versions; thus, this activity represents the inverse of the translation performed by abduction (which in logic is known as *deduction*). If the obtained lower-level previsions match lower-level hypotheses, the latter will undergo the filtering performed by verification and will not flow up. To avoid a false mismatch at the higher level (due to the lack of matching hypotheses if delegated previsions have been matched at a lower level), delegated previsions must be considered as matched until one has proof of the contrary. In other terms, a level can delegate to the lower level the *falsification* of previsions. The lower level can in turn delegate such previsions according to the same pattern.

On one side, the approach dramatically reduces the computation and communication overhead at the topmost levels, thus supporting the development of cost-effective distributed systems. On the other side, it preserves the proper layering, thus encouraging software re-use.

Figure 1 synthesizes the ALARM concepts in a three-layer configuration. Black lines describe hypothesis flows, whereas the grey ones prevision flows. Verification, forecasting and delegation are intra-layer activities. On the contrary, abduction and deduction are inter-layer activities.

3 PRELIMINARY VALIDATION

To study the correctness of the ALARM approach, some basic tests have been performed. Since the verification activity is time-driven and both hypotheses and previsions have a relation with time, we exploited TAM (Time Aware Machine), a Java framework that simplifies the development of time aware systems (Fiamberti et al., 2012).

In order to guarantee the repeatability of tests, actual sensors have been simulated by generating predefined observations at predefined time instants and using deterministic implementations of abducers and forecasters.

The simulated environment is the corridor of a building instrumented with 12 video cameras. The test application is able to identify and track people walking in the corridor.

Three abstraction layers have been identified:

- Ground Layer, that reasons on objects represented by boxes with specific height and width, along with a coordinate location and a mean color of the box;
- Entity Layer, where boxes observed at different times are correlated (basing on position and color) in order to compute speed and direction of movement, thereby identifying persistent entities;
- Domain Layer, where the identified entities are

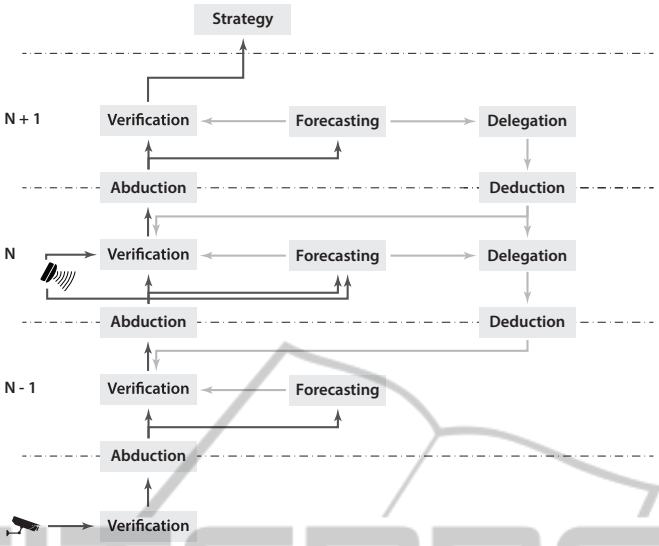

Figure 1: A three-layer ALARM configuration.

recognized as people characterized by a name and an abstract location.

On top of the layer stack, a strategy is found, which is notified when exceptions occur.

The goal of the tests is to measure the number of notifications received by the strategy and the number of data elaborations. The following scenarios have been considered:

- **a** a person is identified; a sequence of previsions about the person's future movements is generated, one for each camera; all such previsions are confirmed by hypotheses.
- **b** at first, the behavior is the same as in the previous scenario. However, after the sixth incoming hypothesis, no more hypotheses are received by the Ground Layer abducer, as if the person stopped in a blind spot.

The two described scenarios will be analyzed without delegation at first. Afterwards, for both of them delegation will be introduced, in order to highlight the corresponding benefits. Figure 2 sketches the system layering in both the situations: with and without delegation. It is possible to notice that not all the activities are present in each layer. Indeed, ALARM does not constraint the presence of each activity in each layer.

3.1 Scenario *a* without Delegation

Every time a hypothesis is received by the ground layer, it flows up to the domain layer, since in the absence of delegation no previsions are available for verification at the two lower layers. Thus, every incoming hypothesis undergoes three abductions. At the domain layer, the first hypothesis generates an exception, since no previsions have been formulated yet. Using the information of the first hypothesis, the forecaster is able to formulate several previsions about what hypotheses will be received in the future. As the time passes, the hypotheses are all matched by the previsions, therefore no more exceptions are generated.

3.2 Scenario *b* without Delegation

Everything goes on as in the previous case until the time when the seventh hypothesis is expected. The mismatch caused by the missing hypothesis generates a second exception at the domain layer. The forecaster then retracts the remaining, already generated previsions in order to avoid future uninteresting exceptions.

3.3 Scenario *a* with Delegation

In this case, delegators are present at the entity and domain layers, together with the associated deducers. Therefore, the verification of previsions generated by the domain-layer forecaster is delegated to the ground layer. Again, a single exception is generated, due to the first incoming hypothesis. The difference with respect to the case without delegation is that now the hypotheses after the first one do not traverse the complete abduction chain, since they are matched against previsions already at the ground layer.

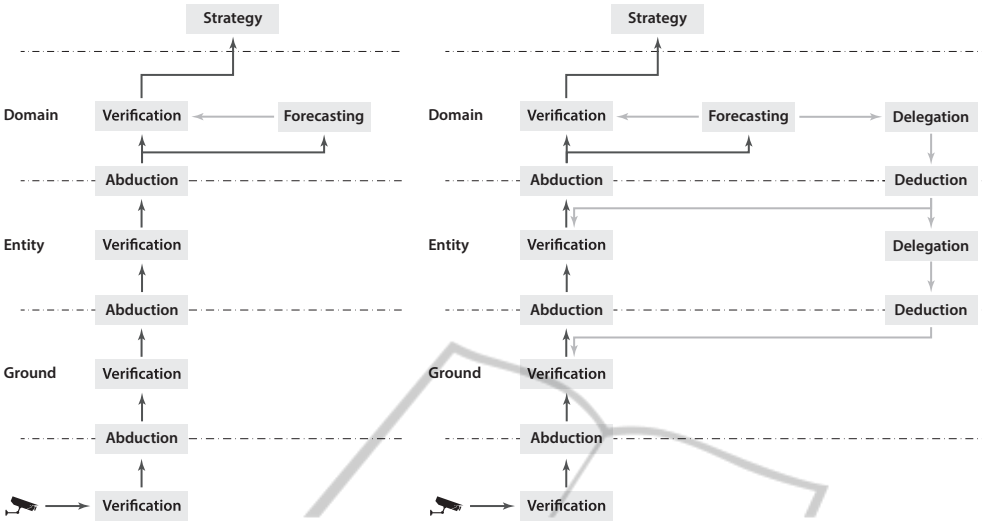

Figure 2: On the left, the configuration without delegation pattern, on the right the one with delegation pattern.

3.4 Scenario *b* with Delegation

The verification of previsions generated by the domain layer is delegated to the ground layer in this case too. When the seventh expected hypothesis is not received, a mismatch is produced and sent upwards. This is the only additional hypothesis that traverses the complete abduction chain. Once the mismatch is received at the domain layer, an exception is generated, exactly as in the case without delegation. Moreover, the forecaster retracts the remaining previsions and the delegator propagates the retractions downwards to the ground layer.

3.5 Discussion

Table 1 summarizes the results. In the table, for every test case the total number of exceptions, abductions and previsions is shown. The corresponding results for a standard (i.e., non ALARM-based) application structured according to the same layering (so that incoming data traverse the same sequence of abductions) are also shown for comparison.

As the table shows, the number of generated exceptions for a given scenario is typically sensibly reduced with respect to non ALARM-based applications.

As for the benefits of delegation, the number of exceptions is not modified when delegation is introduced. However, delegation does introduce an improvement in the form of reduction of computational overhead. In fact, the complexity of the abduction process can be expected to increase going upwards in the layer stack, so that stopping the upward hypothesis flow at the lowest possible layer should greatly reduce the amount of computations required for every incoming hypothesis.

The table does not show the number of deductions for the cases where delegation is present. It is easy to see that the sum of the numbers of delegations and abductions is always equal to the number of abductions in the absence of delegation. This can be considered as an additional advantage of the delegation pattern, as it is reasonable to assume that, from a computational point of view, a deduction is likely to be far easier than the corresponding abduction.

The performed tests are, of course, in no way a complete experimentation: they were simply aimed at producing a preliminary quantitative analysis of the benefits of the ALARM approach and at allowing for a first estimate of how much the actual filtering is and where it occurs. However, these simple experimentations show that using delegation leads to a sensible improvement as far as reduction of inter-layer communication and of computational overhead are concerned, as expected.

4 CONCLUSIONS

In this work, the ALARM architecture has been proposed as a means to overcome the typical drawbacks of the traditional, monolithic approaches to data-driven applications, namely computational and communication overhead, and the lack of timeliness and of the capability to recognize situations characterized by missing data.

The proposed solution is based on a layered

Table 1: Results of the preliminary tests.

Test case	Exceptions	Abductions	Previsions
Scenario *a*, without delegation	1	36	11
Scenario *b*, without delegation	2	18	11
Scenario *a*, with delegation	1	14	11
Scenario *b*, with delegation	2	10	11
Scenario *a*, standard application	12	36	-
Scenario *b*, standard application	6	18	-

structure, where the number of layers is domain-dependant. The internal structure of each layer is the same and reifies the patterns of abduction, verification and possibly delegation. This allows for the reduction of information flows and computational overhead thanks to an exception-based mechanisms which allows information to flow upwards in the layer stack only when something unexpected happens.

In order to validate the proposed approach, a simple implementation of the general structure has been created and used to build a three-layer test application modeling a virtual instrumented environment. The behavior of the test application has been observed in several simple situations in order to obtain a first, quantitative estimate of the benefits of this approach. The results show how the ALARM approach actually helps in reducing computational overhead with respect to a traditional data-driven implementation. Its stratification and modularization make it easy to reuse components and adapt the architecture behavior to fit different specific needs. The hypothesis verification pattern improves responsiveness, while the delegation pattern improves the decentralization of heavy computations.

In order to give a more thorough evaluation of the computational and informational gain introduced by ALARM, a series of more accurate and extended tests are needed. In particular, implementing real life cases will be the only way to establish the actual level of improvement.

Several open issues will be the subject of future developments. First of all, the association of a confidence level to both hypotheses and previsions must be thoroughly examined, given the unavoidable uncertainties in both the abductive and the forecasting processes. Moreover, the concept of delegation can be raised at a meta-level: in many cases, the downstream flow might consist of rules for defining previsions, rather than of already formulated previsions. Finally, matching hypotheses against previsions is a complicated issue: both are timed predicates about objects. Predicates include operators like “similar to”, “close to”, “simultaneous”, which can be hardly defined once for all. Such operators are expected to be overloaded by specific classes of objects exploiting domain-specific algorithms.

REFERENCES

Boyd, J. (1987). A discourse of winning and losing. Unpublished collection of lecture slides available via Interlibrary Loan from such sources as the Marine Corps University Library.

Bruckner, D., Zeilinger, H., and Dietrich, D. (2012). Cognitive automation - survey of novel artificial general intelligence methods for the automation of human technical environments. *Industrial Informatics, IEEE Transactions on*, 8(2):206–215.

Fiamberti, F., Micucci, D., and Tisato, F. (2012). An object-oriented application framework for the development of real-time systems. In Furia, C. and Nanz, S., editors, *Objects, Models, Components, Patterns*, volume 7304 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Hall, C., McMullen, S., Hall, D., McMullen, M., and Pursel, B. (2008). Perspectives on visualization and virtual world technologies for multi-sensor data fusion. In *Information Fusion, 2008 11th International Conference on*.

Hall, D. and Llinas, J. (1997). An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1).

Harris, C., Bailey, A., and Dodd, T. (1998). Multi-sensor data fusion in defence and aerospace. *The Aeronautical Journal*, 102(1015).

Jotshi, A., Gong, Q., and Batta, R. (2009). Dispatching and routing of emergency vehicles in disaster mitigation using data fusion. *Socio-Economic Planning Sciences*, 43(1).

Pau, L. F. (1988). Sensor data fusion. *Journal of Intelligent & Robotic Systems*, 1(2).

Peirce, C. S. (1935). *Collected papers of Charles Sanders Peirce*. Belknap Press.

Shulsky, A. and Schmitt, G. (2002). *Silent warfare: understanding the world of intelligence*. Potomac Books Inc.

Steinberg, A., Bowman, C., and White, F. (1999). Revisions to the JDL Data Fusion Model. In *Society of Photo-Optical instrumentation Engineers (SPIE) Conference Series*, volume 3719.

White, F. (1991). Data fusion lexicon. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA529661.