IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received November 2, 2018, accepted December 20, 2018, date of publication December 24, 2018, date of current version January 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2889557

Scoreboard Architectural Pattern and Integration of Emotion Recognition Results

AGNIESZKA LANDOWSKA AND GRZEGORZ BRODNY
Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, 80-233 Gdańsk, Poland

Corresponding author: Agnieszka Landowska (nailie@pg.edu.pl)

This work was supported in part by the Polish-Norwegian Financial Mechanism Small Grant Scheme under Contract Pol-Nor/209260/108/2015 and in part by the DS Funds of ETI Faculty, Gdańsk University of Technology.

ABSTRACT This paper proposes a new design pattern, named *Scoreboard*, dedicated for applications solving complex, multi-stage, non-deterministic problems. The pattern provides a computational framework for the design and implementation of systems that integrate a large number of diverse specialized modules that may vary in accuracy, solution level, and modality. The *Scoreboard* is an extension of *Blackboard* design pattern and comes under behavioral type. The pattern allows for an integration of multimodal results, employing early, and/or late fusion paradigms. Additionally, it provides a framework for the evaluation of the modules, dealing with inconsistency and low accuracy. In this paper, the *Scoreboard* design pattern is described with the standard meta-data model, followed by a sample implementation. This paper also provides the evaluation results based on experiments and a case study. The evaluation results confirmed the robustness, modularization, ease of integration, efficiency, and adaptability of the solutions with the *Scoreboard* pattern in comparison with the *Blackboard* pattern and "no pattern" condition. This paper provides also a case study of *Scoreboard* application in an integration of emotion recognition results. There are certain complex problems in modern software engineering which require multi-stage, multi-party, multi-modal solutions, and non-deterministic control strategies. Among those are natural language processing, image processing, and emotion recognition, to name just a few. The proposed *Scoreboard* pattern might be used in the software addressing the problems, especially in research systems that explore large solution spaces and require runtime decisions on execution order.

INDEX TERMS Affective computing, Blackboard, design pattern, emotion recognition, early fusion, integration, late fusion.

I. INTRODUCTION

Patterns have been introduced in the field of software architecture by Christopher Alexander, who documented reusable architectural solutions that provide good quality designs [1]. More patterns were proposed and catalogued in late 90's by so-called Gang of Four [2]. A design pattern documents a good practice in software engineering that provides a defined effect on quality attributes. Design patterns capture solutions that have developed and evolved over time and are simple and efficient resolutions to specific problems encountered in software design. In general, a pattern has four essential elements: the pattern name that is a handle we can use, the problem that describes when to apply the pattern, the solution that describes the elements that make up the design, their relationships, responsibilities, collaborations and the consequences as well as trade-offs of applying the pattern [2]. The consequences are critical for evaluating design alternatives and for understanding the costs and benefits of applying the pattern. The consequences for software often concern trade-offs between time and space and between re-usability/flexibility and performance. Multiple design patterns were defined in last two decades and modern research focuses on exploring the consequences and trade-offs they imply [3]–[6].

There are multiple problems in diverse areas, for which deterministic solutions are not available. Moreover, some problems, such as natural language processing [7], [8], object tracking in videos [9] or emotion recognition [10] require multifaceted solutions that combine miscellaneous modules applying diverse approaches. Not only there are several

 This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/

competitive components or algorithms to perform selected sub-tasks, but also a sequence or combination of those sub-tasks might be unknown at the design stage.

A non-deterministic application model is selected, when the control flow cannot be precisely defined at a design or implementation stage [11]. System response and control flow might be different for distinct input data and even for identical input data. The latter condition might occur, when the system state varies depending on a sequence of preceding events. Non-deterministic architectures are frequently used in applications simulating biological or physical phenomena as well as in complex (multi-phase) recognition systems [12]–[14].

Implementation of non-deterministic solutions is usually based on non-deterministic finite state automata that defines permissible states and transition conditions [15]–[17]. In non-deterministic automata, more than one transition might be available for a pair of state and condition. The transition is then triggered based on random or pseudo-random criteria and defining the ones is another challenge in system design. Sometimes during runtime the algorithms are repeatedly launched in order to search a space of possible solutions. The complexity of the process depends on the size of the solution space, number of processing stages as well as quantity of competing or supplementary algorithms per stage.

Searching large solution space is very inefficient, therefore in order to limit complexity, some solutions are proposed such as suboptimal decision criteria, reduction of stages or limitation of algorithms number per stage [18]. As a result, the key challenges faced by architecture designers of non-deterministic systems might be defined as follows:

- ease of algorithms' modification and replacement,
- continuous evaluation of the application and its individual components (modules and algorithms),
- embedding learning opportunities on the basis of the current or previous executions (adaptability),
- building confidence of the results and the components during the application runtime,
- the ability to solve problems using competition or voting approach (late fusion).

As multiple recognition problems have similar conditions and challenges, the concept of this research was to address them proposing a new design pattern that is dedicated to complex non-deterministic problems. This paper proposes a new design pattern, called *Scoreboard* that is dedicated to software systems that integrate large number of diverse specialized modules. The proposed design pattern falls into architectural category, 'mud to structure' sub-category and was called a *Scoreboard*. It is an extension of the Blackboard design pattern and the main difference is the extension with continuous evaluation of partial components regarding inconsistency and accuracy. As a result, the pattern enables fusion of the results using both early and late paradigm as well as spotting an invalid module in the runtime under current context.

The affective computing application area is the one that the *Scoreboard* pattern originated from, however when we have struggled for greater reuse and flexibility of the emotion recognition software, we realized that with time we developed a more general solution to be applied in multiple systems design. This paper first introduces the *Scoreboard* design pattern to the general public, formally defines it and provides an evaluation of the design. The paper is organized as follows. Section 2 provides background for proposing a new design pattern, the scenario that constitutes a problem to be solved and related work on both fusion in emotion recognition and evaluation of design patterns. Then the pattern is formally described in section 3. Section 4 contains thesis decomposition and research methodology of experiments and studies used for evaluation of the pattern. Section 5 provides the evaluation results, followed by discussion of results and validity of research in section 6. Section 7 concludes the paper with the suggestion of further applications and further research.

II. RELATED WORK

Works that are mostly related to this research fall into three categories: (1) research on design patterns, (2) studies on emotion recognition based on diverse input channels and multimodal fusion and (3) methods for evaluating architectures of software solutions that constitute research methodology of this study.

A. DESIGN PATTERNS

Design patterns in software development are used for reusability, clarity and maintainability in common design problems [19]. They can also help to understand others' code and document best practices [11]. Design patterns for software development were extensively explored and described by Gamma *et al.* [2], the so-called Gang of Four (GoF), in 1996. In general, solutions which apply design patterns should be more flexible, modular, reusable and understandable. The qualitative gain obtained by using a pattern depends on an addressed scenario and a pattern itself. Gamma et al. divided design patterns into three categories: creational, structural and behavioral. The creational category patterns make systems independent of how objects are created, e.g. Abstract Factory, Prototype, Singleton, Factory Method. The structural patterns concern how classes and object are composed to larger structures, e.g. Adapter, Decorator, Proxy. The behavioral patterns concentrate on communication between objects, e.g. Template Method, Observer. The GoF patterns are still popular topic in research and practice, as confirmed by a study of Ampatzoglou [19]. The literature-based study of GoF design patterns' pros and cons revealed that in general more flexibility entails higher complexity. Buschmann et al. proposed an alternative categorization of patterns, not limited to object-oriented programming, with the following categories: Architectural Patterns with subcategories: From Mud to Structure, Distributed Systems, Interactive Systems, Adaptable Systems, and Design Patterns with subcategories: Structural Decomposition, Organization of Work, Access

MOST WIEDZY Downloaded from mostwiedzy.pl

Control, Management, Communication and Idioms [11]. Patterns falling into ''From Mud to Structure'' category are designed for decomposition of tasks among multiple cooperating components. Among patterns that fall into this category one might emphasis Layers pattern, Pipes and Filters pattern and Blackboard pattern. The latter one supports non-deterministic solutions for domains that lack standard approaches. Blackboard pattern supports solutions, in which problems might be decomposed and each fraction might be solved by an independent specialized component. The Blackboard pattern was first used in HEARSAY-II system for speech recognition [20]. The system consisted of several components for word tagging, syntax analysis, vocabulary spelling check, acoustic analysis of phones, syllables, words and phrases and so on. At the time there was no consistent algorithm that combined all the necessary procedures for recognizing speech, moreover issues of ambiguities of spoken language, noisy data, and the individual peculiarities of speakers had to be addressed. The solution was non-deterministic, multifaceted system and the Blackboard pattern allowed to address the complexity, flexibility and performance of the design [20]. As the proposed Scoreboard pattern is an extension of Blackboard pattern, the latter will be used in the evaluation process for reference.

B. INTEGRATION AND FUSION IN EMOTION RECOGNITION

There are numerous emotion recognition algorithms that differ on input information channels, output labels or representation models and classification methods [21]. The most frequently used emotion recognition techniques that might be considered when designing an emotion monitoring solution include: facial expression analysis, audio (voice) signal analysis in terms of modulation, textual input analysis, physiological signals as well as behavioral patterns analysis [22]. As literature on affective computing tools is very broad and has already been summarized several times, for a more extensive bibliography on affective computing methods, one may refer to Poria *et al.* [22], Gunes and Schuller [23], or Zeng *et al.* [24].

Multichannel observation of human emotions is applied in the following domains: usability and user experience evaluation [25]–[27], educational software and resources designed for e-learning [28]–[30], affect-aware games and other intelligent personalized systems [31]–[33], and in optimization of processes [31], [34]–[36]. Although it might seem that the domain of affective computing is well established and there are even some off-the-shelf solutions, the reliability, accuracy and granularity of emotion recognition is still a challenge [37].

The best recognition results are obtained when fusing information from diverse input channels and early versus late fusion is distinguished [30]. In early fusion methods features derived from independent input channels are combined to create a common feature vector for classification [38]. Late fusion combines the classification results provided by separate classifiers for every input channel; however, this requires some mapping between emotion representation models used as classifier outputs [39]. The latter approach allows to integrate algorithms using a black-box approach.

Hupont *et al.* [39] claim that multimodal fusion improves robustness and accuracy of human emotion analysis. They observed that current solutions mostly use one input channel only and integration methods are ad-hoc designed.

Gunes and Piccardi provide a comparison of early and late fusion solutions. In their experiments they used facial expression and body gestures channel. The better recognition accuracy was achieved using the integrated solutions. The better of the two methods was early fusion, but authors suspect that the effect could be caused by too small training set. Gunes and Piccardi [38] propose to consider also a combination of the two approaches – a hybrid fusion. Another example of comparison of both methods is Poria et al. research. Both methods don't significantly differ in accuracy, but decision-level is much faster – is more parallelizable [40], can improve robustness [41] and is feature-independent [42].

Hybrid fusion combine feature-level and decision-level fusion methods, which is opt by researchers to exploit the advantages of both methods and overcome the disadvantages of each [22]. In this kind of method, often one input channel can be used by two or more classifiers, as in feature level, and results from both are integrated as in decision level [41], [43].

Some major challenges in multimodal affect analysis that were raised by different authors and need to be addressed, are as follows:

- robustness (continuous data from real noisy sensors may generate incorrect data) [22];
- accuracy and overall performance requirements, which restrict the usefulness of such systems in practical applications [22];
- effective modeling and processing of temporal information - time scale and synchronization of modalities [42], [44];
- optimal weight assignment to the different modalities [44];
- adaptability (context-dependent evaluation of temporary results, temporal unavailability of input channels) [44].

Solving selected from the above challenges was one of the reasons behind proposing the Scoreboard design patern.

C. ARCHITECTURE AND DESIGN EVALUATION METHODS AND METRICS

Each design pattern was proposed for solving challenges of certain scenario, but application of the one might compromise other quality criteria. Evaluation methods of the design and design patterns include defining quality criteria and/or a procedure of evaluation and both qualitative and quantitative approaches were proposed. The goal of the evaluation is to estimate the potential of the designed architecture to facilitate or inhibit the achievement of the required quality attributes.

MOST WIEDZY Downloaded from mostwiedzy.pl

Ampatzoglou used ISO 9126 as reference model for software quality, which contains an extensive list of quality attributes [19]:

- functionality, which includes suitability, accuracy, interoperability,
- reliability, which includes maturity, fault tolerance, recoverability,
- usability, which includes understandability, learnability, operability, attractiveness,
- efficiency, which includes time behavior, resources, utilization,
- maintainability, which includes analyzability, changeability, stability, testability,
- portability, which includes adaptability, installability, co-existence, replicability.

Apart from ISO standard, software engineering communities proposed diverse quality attributes to evaluate software systems designs. For example Losavio and Chirinos analyzed three models: ISO 9126 [45], Dromey [46] and ABAS (Attribute-Based Architectural Styles) [47], and derived the attributes of portability, extensibility, adaptability, functionality, reliability, usability, efficiency, maintainability, abstraction, robustness, performance and modifiability [48].

Meyer describes the 'good' system (in terms of design) as being fast, reliable, easy to use, readable, modular and structured [49]. He proposed a list of external quality attributes including correctness, robustness, extendibility, reusability, compatibility, efficiency, portability, ease of use, functionality and timeliness.

Modularity by Meyer, covers the combination of extendibility, reusability and flexible factors. The components need to be self-contained and organized in stable architectures [49]. This should help designers produce software, which elements are connected by a coherent, simple structure. Among criteria and metrics classified by Meyer two are commonly used: coupling and cohesion.

Coupling is a measure, which represents the strength of interconnection between two or more modules [50]. Coupling is influenced by: type of connection between modules, complexity of the interface, information flow type and connection binding time. A number of metrics were proposed for the coupling attribute: Coupling between Objects (CBO), Response for Class (RFC), Message Passing Coupling (MPC), Data abstraction Coupling (DAC), Ce and Ca (efferent and afferent coupling), Coupling factor (COF), Information flow based coupling (ICP), Inheritance based coupling As ICP (IHICP), Non Inheritance based coupling As ICP (NIHICP), Polymorphic ally invoked methods (PIM), Export coupling version of PIM (PIM_EC) and Average number of parameters per method (NPAVG) [51].

Cohesion by Yourdon and Constantine is a measure, which represents degree of functional relatedness of elements within single module [50]. In object paradigm, a class is a natural cohesive entity as all properties and operation should be packaged for one purpose only. A number of metrics were proposed for the cohesion attribute, including: Lack of Cohesion in Methods (LCOM), Tight class cohesion (TCC), Loose class cohesion (LCC), Connectivity (Co) and variation on LCOM5 (Coh) [51].

Cohesion and coupling are interrelated and the most desirable combination is a high cohesion of individual modules and a loose coupling between the modules.

Selected quality attributes are used in this paper in evaluation of the Scoreboard design pattern consequences.

Apart from quality attributes and metrics, some methods (processes) are defined for evaluation of architectural approaches. Babar et al. classified a number of methods including: Scenario-based Architecture Analysis Method (SAAM), Architecture Tradeoff Analysis Method (ATAM), Active Reviews for Intermediate Design (ARID), SAAM for Evolution and Reusability (SAAMER), Architecture-Level Modifiability Analysis (ALMA), Architecture-Level Prediction of Software Maintenance (ALPSM), Scenario-Based Architecture Reengineering (SBAR), SAAM for Complex Scenarios (SAAMCS) and integrating SAAM in domain-Centric and Reuse-based development (ISAAMCR) [52].

Some of the SA evaluation methods are being promoted as architectural design and analysis methods, e.g., ATAM [52], [53] and Quality Attribute- oriented SA design method (QASAR) [11].

The most mature methods, SBAR and ATAM, suggest considering multiple quality attributes and performing some trade-off analysis and this approach was chosen in this study. According to Babar et al. only two of the surveyed methods, ATAM and ALMA reached sufficient applicability level [52].

From the methods listed above, following Barbar's recommendation, ATAM method was applied in this study to evaluate the Scoreboard design pattern.

III. SCOREBOARD DESIGN PATTERN DESCRIPTION

The formal description of the pattern follows, according to meta-data model proposed by GoF [2]. Whenever we refer to the pattern name, it is written in standard font, while names of the classes are written in *italic*.

A. PATTERN NAME

Scoreboard

B. PATTERN CLASSIFICATION

Architectural pattern (Mud to structure subtype)

C. INTENT

Scoreboard design pattern is dedicated to non-deterministic problems with large potential solution spaces that require integration of multiple multi-faceted algorithms. Especially it might be useful in experimental systems that are developed in an iterative manner and the components are frequently optimized and updated. Application of the Scoreboard pattern not only allows for easier integration of individual results, but also enables runtime evaluation of the competing components according to chosen criteria. This feature enables automatic evaluation, algorithm optimization and adds a learning aspect (post-hoc or runtime) to the solution.

MOST WIEDZY Downloaded from mostwiedzy.pl

Adaptability and learning is possible: (1) on algorithm level (algorithms might learn and provide more accurate results), (2) on component level (components might provide a confidence estimate) and (3) on system level (by comparing alternative algorithms' hypothesis). Natural inclusion of confidence makes the Scoreboard pattern suitable also for solutions dealing with uncertainty (as most recognition systems do). Both early fusion and late fusion paradigms are supported in the integration of the results.

D. ALSO KNOWN AS

There is no other name.

E. MOTIVATION

Recognition problems share some challenges, including availability of multiple competing algorithms (classifiers) that differ in terms of input feature vectors, output model, accuracies, and performance. As a result the key requirements of recognition systems architectural design are similar: algorithms' modification and replacement, integration using early and/or late fusion, continuous evaluation of the application and its individual components/algorithms, embedding adaptability and building confidence in uncertain environments. Sometimes best results are obtained with multiple trials of diverse configurations and settings.

In such a scenario, there are at least two required features of the solution: changeability (adaptability) and dynamic evaluation. Both features are provided with the proposed Scoreboard pattern - it was designed to easily modify tested algorithms set, share data among them and evaluate them (regarding consistency or any other defined criteria) during runtime.

F. APPLICABILITY

Scoreboard design pattern might be applied in the following solutions:

- research applications and projects, aiming at evaluation of the algorithms and their optimization,
- non-deterministic applications, when the order of execution is unknown at the design stage and space of possible solutions is extensive,
- solutions that require continuous improvement and evaluation of the changes made.

Scoreboard design pattern might be not efficient in:

- projects with time pressure, as it requires additional implementation effort,
- deterministic applications, when the control flow is known at the design stage,
- solutions that have high performance requirements, as continuous evaluation of modules requires processing time.

G. STRUCTURE

Scoreboard architectural pattern is an extension of design pattern Blackboard. Both patterns hold similar classes: single instance of *Blackboard* class, single instance of *Controller* class and multiple instances of *Expert* class. *Blackboard* instance is used to share data among the *Experts* and *Controller* instance is responsible for launching *Experts* and controlling the access to *Blackboard* instance. The *Experts* read *Blackboard* entries (provided by the other *Experts*) and add their entry that might be a next-stage result or a competitive thesis to the other experts' claims. As a result, the *Blackboard* might hold multiple claims about the same solution stage. The problem of integration or evaluation of claims is not solved within Blackboard pattern and this is one of the reasons behind an extension to the *Scorboard* pattern.

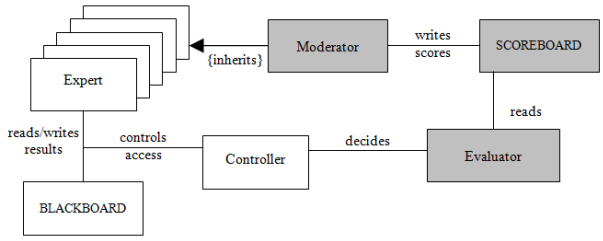

FIGURE 1. Simplified class model of Scoreboard design pattern.

The new *Scoreboard* pattern apart from instances of *Experts*, *Blackboard* and *Controller* is extended with 3 classes: *Scoreboard* (single instance), *Moderator* as a subclass of *Expert* class and an *Arbiter* (single instance). The classes and relations are shown on a simplified class model (Fig. 1). White component are derived from Blackboard pattern, while grey ones are added by Scoreboard pattern.

The *Scoreboard* class instance is used to keep evaluations of the algorithms (represented as *Expert* classes). After *Expert* instances submit their claims to *Blackboard*, *Moderator* instance is responsible to evaluate multiple claims that address the same solution stage and propose a concluding result. Additionally *Moderator* class might evaluate *Experts* by writing to *Scoreboard* instance. The evaluation might be based on: known ground-truth or approximate methods, based for example on consistency among modules. Processing of concluding result from competing claims might follow any fusion algorithm implemented as *Moderator* instance.

At this stage, with the addition of *Scoreboard* and *Moderators* instances, the solution allows to continuously evaluate any stage of the processing towards final results. However, Scoreboard pattern adds additional class of *Evaluator*, which is assigned with another task of choosing algorithms (*Experts*) to execute. In the Blackboard pattern *Experts* are called with a round-robin scheme and each round consists of execution of each *Expert*. This might be inefficient, if there are criteria to choose among *Experts*. In *Scoreboard* pattern the choice might be based on any criteria implemented within *Evaluator*, including accuracy, performance, or trade-off between them as well as criteria based on runtime evaluation, gathered in *Scoreboard* instance. As *Moderator* might have multiple instances, it is possible also to test alternative integration and evaluation strategies.

MOST WIEDZY Downloaded from mostwiedzy.pl

H. PARTICIPANTS

The Scoreboard pattern is constituted with 6 classes:

1) *BLACKBOARD* CLASS

Blackboard class is a central repository (a shared memory space) used to exchange data among *Expert* classes. It holds multiple entries varying from initially preprocessed data, through all stages of solution up to final results. Each entry, apart from content (called hypothesis), should hold at least following metadata: time, author (*Expert* class), solution level, uncertainty of hypothesis.

Time stamp might reflect the time of putting the hypothesis, or alternatively, the time of the data recording. The later approach should be considered if hypothesis are obtained for time stamped data, which makes metadata time independent of processing time.

Solution level represents conceptual distance of the hypothesis from initial data, reflecting stages of processing towards final result.

Blackboard class is extended in Scoreboard pattern with additional metadata attribute representing confidence (or uncertainty) of the provided hypothesis. The attribute calculation should be based on run-time information on circumstances that influence reliability of the provided hypothesis. It should not be dependent on overall algorithm accuracy, but rather should quantify run-time information, e.g. data partially unavailable, quality of input data, etc.

2) *EXPERT* CLASS

Expert class represents independent modules, subsystems or algorithms that solve some sub-problem of the final solution. All *Expert* instances' combined results provide complex solutions to a problem they cannot solve separately. The concept of expert set is an analogy to human expert teams that solve complex problems by iterative process of discussion and knowledge sharing. *Expert* class provides a common interface for all competitive and supplementary algorithms for multiple stages of the solution. There are two types of expert knowledge processing: providing a next stage solution (forward reasoning) or alternatively verifying a solution obtained at the current stage (backward reasoning).

3) *CONTROLLER* CLASS

Controller class is a single-instance class responsible for controlling the application execution order by granting access to *Blackboard* and *Scoreboard* instances for experts. Practically the problem could be reduced to choosing the next *Expert* to call, however as the appropriate sequence is unknown at development stage in non-deterministic systems, multiple strategies might be implemented. Rule-based strategy might be one of the options, assuming the rules might be defined. Another option is a simple round-robin scenario that calls all *Experts* in iterations in order to check whether they have anything to add to the solution. The rule-based and round-robin scenarios for application flow control are known from Blackboard design pattern. One of the reasons for extension to Scoreboard pattern was the need for more complex solution that take into account run-time efficiency of *Experts*.

4) *MODERATOR* CLASS

Moderator class is a instantiation of *Expert* class and inherits blackboard entry reading and adding functionality. *Moderator* sub-class new function is to compare and evaluate competing hypothesis (blackboard entries at the same solution level). *Moderator* is able to add an integrated result as a single blackboard entry to share as well as might implement *Expert* evaluation function, based on pre-defined labels, golden solution or consistency among hypothesis. *Moderator* writes expert evaluation scores to *Scoreboard*. Each phase of processing towards final solution might have an independent *Moderator*. As *Moderator* inherits from *Expert* class, assessment algorithms might be evaluated as well, which constitutes a double-loop learning feature.

5) *SCOREBOARD* CLASS

Scoreboard class is a central repository (a shared memory space) used to exchange control data, especially about efficiency of *Expert* classes. The data are then used by *Controller* and *Evaluator* classes to propose a control flow. Several scenarios for the *Scoreboard* organization might be proposed, from sharing an average result per *Expert* instance only to sharing all detailed run-time evaluation entries.

6) *EVALUATOR* CLASS

Evaluator class is an extension of *Controller* that might implement additional evaluation algorithms and complex criteria for choosing next experts to call. Separate *Evaluator* class enables easier modification of the algorithms, as well as a possibility to implement multiple ones. Final scores might be evaluated by: usage of the *Expert* data, consistency with "golden solution", inter-expert consistency, etc.

I. COLLABORATIONS

The collaboration scenario is proposed as follows:

1. *Controller* module is started and runs in iterative mode. In each iteration *Controller* module checks whether final solution (the highest level hypothesis with acceptable confidence) has been reached. If not, another iteration is started.
2. *Controller* module identifies competitive experts that might be called at the current solution level.
3. In case when multiple experts might be called, *Evaluator* algorithms are launched to choose among competitive *Experts*.
4. *Controller* module grants access to *Blackboard* to the chosen *Expert* instance.
5. *Expert* reads *Blackboard* state, processes the data and adds an entry to *Blackboard*.
6. *Expert* (if being an instance of *Moderator* subclass) might add to *Scoreboard* entries on evaluations of other *Experts*.

MOST WIEDZY Downloaded from mostwiedzy.pl

7. The iterations continue until a final solution is reached, or, alternatively, no solution was reached, but no *Expert* has anything to add to solution space on *Blackboard*.

J. CONSEQUENCES

The design pattern application allows to solve some of the challenges in non-deterministic complex and learning environments:

- it is possible to use multiple competing algorithms that are implemented in diverse technologies (using black-box approach), assuring that a wrapper code implementing *Expert* interface exists;
- it is possible to use multiple competing algorithms for the same processing phase, use them in parallel and compare results;
- multiple algorithms might be evaluated for run-time efficiency;
- support for modifiability - algorithms represented as *Experts* might be easily changed, attached and detached;
- support for re-use - verified algorithms might be used in all applications based on Blackboard or Scoreboard design patterns;
- support for evaluation and representation of confidence (or the opposite - uncertainty).

There are several consequences of using Scoreboard pattern in the design of the particular system that could be considered as drawbacks:

- difficulty in testing, especially when significant number of experts is involved - the Scoreboard inherits this feature from Blackboard predecessor;
- there is no guarantee for obtaining a satisfactory solution, which is characteristic for non-deterministic systems;
- reduced performance when there is no sufficient criteria that reduces solution space and number of experts to call;
- increased implementation time.

K. IMPLEMENTATION

Implementing the application with Scoreboard design pattern requires following steps:

1. Detailed definition of problem (analysis of input and expected result, defining of stop criteria).
2. Definition of solution space (all intermediate solution stages defined on diverse abstraction levels).
3. Decomposition of the process into stages with clear attribution of results defined as intermediate solution stages).
4. Definition of *Blackboard* entries form (how data is shared and hypothesis are represented, how confidence/uncertainty is represented).
5. Defining criteria for evaluation and selection of experts - key point would be a criteria that reduces solution space and/or competing algorithms to call. Definition of *Scoreboard* entries.
6. Decomposing the functionality to experts representing processing stages (optimally to the point of single expert being a transition between two consecutive solution levels, mixed-level approach is also possible).
7. Definition of control flow and implementation of *Controller* and *Evaluator*.
8. Implementation of *Experts* or *Expert* wrappers following the defined interfaces.

```
public interface IExpert : IComparable<IExpert>
{
    void DoExpertWork();          //reading data from Blackboard,
                                  //processing (expert's work), adding a new entry
    IWorkCondition GetExpertWorkCondition();
                                  //checking the criteria for the expert to be called
    bool ProcessingPhase(string phase);
                                  //getting information about processing stage
    string GetExpertName();       //getting Expert name
}
```

Listing 1. Sample code - definition of Expert class interface.

```
public interface IArbiter
  {
    List<IExpert> GetActiveExperts();       //returns all experts that are
                  // allowed to activate, regardless if they have anything to add
    List<IExpert> GetActiveExperts(String type, int max=int.MaxValue,
                  int min = 0);
                  //returns experts of given type that are allowed to activate,
                  //regardless if they have anything
    List<IExpert> GetActiveExperts(String type);
                  //returns all experts that are allowed to activate, taking into
                  //account current phase,  regardless if they have anything to add
  }
```

Listing 2. Sample code - definition of Evaluator class interface.

```
public interface IBlackboard
  {
    void AddBlackboardEntry(T entry);
                  //adding a new entry (of any type T) to Blackboard
    List<T> GetEntries();
                  //reading all entries (of any type) currently at Blackboard
    List<BlackboardEntry>GetEntries<T>(IBlackboardEntryType<T> type);
                  //reading all entries of specific type T
    List<BlackboardEntry> GetEntries<T>(string moduleName,
                  IBlackboardEntryType<T> type);
                  //reading all entries added by specific Expert
    int GetCurrentTurn();
                  //getting turn count (processing is divided into stages and turns)
    int GetCurrentPhase()
                  //getting stage count (processing has stages and turns)
    void ClearEntry();
                  //Blackboard clearing (removes all entries)
    void RemoveEntry(BlackboardEntry entry);
                  //removing single Blackboard entry
    void ChangeTurn();                    //increment turn count
    void ChangePhase();                   //increment stage count
    PhaseTurnNumber GetPhaseAndTurn();
                  //getting processing status (stage and turn)
  }
```

Listing 3. Sample code - definition of Blackboard class interface.

L. SAMPLE CODE

Sample code of selected key components follows. Listing 1 provides a definition of *Expert* class interface with inversion of control. Listing 2 provides a definition of *Evaluator* class interface, while Listing 3 for *Blackboard* class.

Listing 4 provides sample code of Controller class.

MOST WIEDZY Downloaded from mostwiedzy.pl

```
override public void run()
  {
    bool nothingAdded;          //logical field (flag for stage stop criteria)
    foreach (String stage in stages)          //iterate stages
      {
      do
        {
        nothingAdded = true; //setting initial stage flag value
        foreach (var expert in arbiter.GetActiveExperts
              (stage,MaxExpertsNumber,MinExpertsNumber))
                //iterate list of Experts provided by Arbiter class
                //instance (limited no), if required number of Experts
              //is less than available, Arbiter chooses Experts
              // based on algorithm informed by Scoreboard entries
          {
          tryLaunchExpert(expert, ref nothingAdded);
                //call expert - expert reads Blackboard and might
              //add entry to Blackboard (then stage flag is set to
              //false) or report that he has nothing to add
          }
        blackboard.ChangeTurn();
       //if any Expert added something change turn and repeat stage
        if (moderator.processesPhase(stage))
       //if there is a Moderator class instance for this stage
          moderator.DoWork();
       //launch Moderator (subclass of Expert)
        }
      while (!nothingAdded && blackboard.GetCurrentTurn()
                < MaxRoundInPhaseNumber);
              //repeat until turn resulted in any new Blackboard
              //entry or limit of turns was reached
      blackboard.ChangePhase();          //change stage
      }
  }
```

Listing 4. Sample controller method code.

M. KNOWN USES

Scoreboard design pattern was used in Emotion Monitor integration solution that enables integration and evaluation of multimodal emotion recognition algorithms [54].

N. RELATED PATTERNS

Scoreboard design pattern is a direct extension of Blackboard design pattern, as defined previously. It was intentionally proposed as an extension so it could be used in existing applications using Blackboard already. Progressing from Blackboard to Scoreboard requires: adding memory space (Scoreboard) and implementation of Moderator/Evaluator modules.

In the Emotion Monitor solution the Scoreboard was combined with Inversion of Control and Dependency Injection patterns to improve modifiability.

IV. EVALUATION OF SCOREBOARD DESIGN PATTERN - RESEARCH METHODOLOGY

A. THESIS

This paper proposes a new design pattern, called Scoreboard that is dedicated for multi-stage integration of results provided by solutions developed in diverse technologies. An additional feature of the pattern is to enable run-time evaluation of solutions and adjusting to certain context of use. The design pattern is dedicated to non-deterministic problems, where exact application flow is hard to define, with additional option to compare partial results. The pattern is also suitable for experimental applications design, as enables comparison and evaluation of multiple algorithms.

As stated above, there are multiple quality metrics that apply to system design evaluation and no ultimate method to select them, therefore in this study we apply a goal–question-metric (GQM) [55] approach to select criteria and metrics for the evaluation.

Goal: Analyze *the architectural design of the integration solution* in order to *characterize it* with respect to *quality of multi-faceted integration* from the point view of *developers* relative to *the multimodal recognition challenge*.

Question 1: Is the solution modular enough to incorporate available (multiple and diverse) recognition algorithms (called experts)?

Question 2: Does the solution allow for easy integration of the experts results?

Question 3: Is the solution adaptable to changing runtime circumstances (e.g. temporal unavailability of expert)?

Question 4: Is the solution robust to (temporarily or persistently) invalid expert injection?

Question 5: Does the design with focus on integration, compromise efficiency?

The questions are mapped into the following five criteria: modularization, integration, adaptability, robustness and efficiency. The chosen architectural approach should improve the first four characteristics (or at least be neutral to those criteria). As usually improving modularity might compromise efficiency, the latter criteria was added to the evaluation process and the expectancy is not to compromise performance significantly. The following definitions of the evaluation criteria were adopted in this study:

Modularization is a feature of the architectural design that allows to separate logical units (modules, components) that constitute the solution. Literature mentions high cohesion and low coupling as standard criteria for evaluation of the modularization feature [51].

Integration is a feature of the architectural design representing ability to combine results coming from diverse algorithms. The criteria should be understood as ability to integrate and to provide the results that are not worse than results from individual experts (algorithms).

Adaptability is a feature of the architectural design that allows for easy attachment/detachment/swap of algorithms providing results.

Robustness is a feature of the recognition software that benefits from proper integration algorithm. If an invalid expert provides the results for the integration, the final result should be compromised as little as possible and the invalid algorithm should be spotted.

Efficiency is the criteria that represents the overhead of the integration solution. In this study extra lines of

MOST WIEDZY Downloaded from mostwiedzy.pl

code (LOC) were provided along with time-based and memory-based measures.

The main thesis of the paper might be formulated as follows:

> *Scoreboard design pattern provides modularization, integration, robustness, adaptability and efficiency for non-deterministic applications integrating multiple and diverse algorithms with large solution space to explore.*

According to authors' knowledge, there are multiple problems that require multi-faceted multi-technological algorithms and there is no design pattern dedicated to solving such challenges. If the above thesis is confirmed, the proposed Scoreboard pattern might be a solution to some multimodal integration challenges. Detailed operationalization of the criteria as specific metrics is provided with the description of the experimental design that follows.

B. RESEARCH METHODOLOGY OF THE EVALUATION STUDY

To verify the research hypothesis that Scoreboard design pattern provides modularization, integration, robustness, adaptability and efficiency for non-deterministic applications, we used a number of techniques: laboratory experiments, simulations and a case study.

Two experiments were planned, performed and analyzed. The experiments were followed by two simulations and a real case study of Scoreboard pattern application in emotion monitoring solution.

The difference between simulation and experiment is usually defined by the use of real objects or real data, however the border between the two is fuzzy. In this study we use expression 'experiment' whenever the integrated algorithms are actually processing real data. As experimental setups did not allowed us to test all scenarios, additional simulations were performed. By 'simulations' in this study we mean that mock-ups of integrated algorithms were used in order to test some specific integration scenarios.

As a result, the following components constitute the evaluation study:

- Experiment 1. Integration of multiple text-processing algorithms;
- Experiment 2. Integration of multiple sentiment analysis algorithms;
- Simulation 1. Specific robustness scenario with mocked-up algorithms;
- Simulation 2. Specific scalability scenario with mocked-up algorithms;
- Case study of Emotion Monitor software layer.

Experiments and simulations allowed to obtain metrics (quantitative approach), while the case study is evaluated with ATAM method, which represents qualitative approach to architecture evaluation.

C. EXPERIMENT 1. INTEGRATION OF MULTIPLE TEXT-PROCESSING ALGORITHMS

Goal of the experiment was to compare architectural system construction using Scoreboard design pattern versus Blackboard-based and no-pattern architecture.

For the experiment three versions of software system of the same functionality were implemented. The versions differed in architecture, while performing the same text processing function, i.e. coding text with multiple letter-to-letter transformations, and using 20 text processing algorithms, each solving a sub-transformation of the textual input, while the order of their execution was unknown at design stage. Each of the algorithms was implemented as an independent *Expert* class instance and single execution of each one was necessary to perform text transformation that passes the stop criteria.

Independent variable in the experiment is architectural approach with values {No-pattern; Blackboard-based, Scoreboard-based}.

In the experiment modularization and efficiency criteria were addressed.

For modularization the following metrics were chosen as dependent variables: (M1) number of distinctive classes; (M2) methods per class ratio; (M3) LOC per method ratio; (M4) cyclomatic complexity of methods; (M5) depth of inheritance; (M6) class coupling; (M7) methods coupling.

The metrics were calculated automatically by the Visual Studio 2013. The metrics include cyclomatic complexity measured at method level. The metric was proposed by McCabe in 1976 [56] and could be calculated for different languages. At method level cyclomatic complexity should be less than 10 for each class (classes with values higher than 20 are prone to errors and their refactorisation should be considered). Another measure was depth of inheritance (generally the lower, the better, however a value of 1 indicates no abstraction paradigm was applied, which might be considered wrong). We calculated also coupling at class and method level (generally the lower, the better). Coupling at class level should be lower than 10-20 [57], [58]. According [59] measures should be in relative thresholds – for some percent of classes e.g. 10 for 70% of classes, 22 for 80% [58] etc. Another research gives some reference statistics for coupling measure: mean 13.4 and median – 8 [60].

Efficiency might be measured with: theoretical complexity, execution count or execution time. While increase of complexity depends on specific algorithm and data size, and in this study we evaluate integration function only, the execution count and execution time were chosen as metrics. Additionally initialization time was calculated along with execution time. Moreover, as time efficiency sometimes compromises memory usage, also additional metrics were defined for memory.

As a result for efficiency the following metrics were chosen as dependent variables: (E1) no of expert executions (count);

MOST WIEDZY Downloaded from mostwiedzy.pl

(E2) execution time [s]; (E3) initialisation time [ms]; (E4) paged and non-paged memory used [kB].

The input was standardized: a file with one thousand lines, each containing up to 2 million characters (about 963 MB) with each line being an independent task and exactly the same file used for all architectural approaches. Tests were run on CPU Intel Pentium C860 3GHz 2 cores, 16 GB RAM, operating system Windows 10. Test were run during low system load. Test was repeated 10 times for each architectural approach. Mean execution time and memory usage is reported along with standard deviation and maximum values.

To sum up, the thesis of the experiment might be formulated as follows: Scoreboard-based architectural approach provides higher efficiency not compromising modularization features compared to Blackboard-based and no-pattern approaches.

D. EXPERIMENT 2. INTEGRATION OF MULTIPLE SENTIMENT ANALYSIS ALGORITHMS

In order to measure the integration and robustness factors, an experiment was held that simulated how the integration solution performs (in terms of emotion recognition accuracy), with a number of algorithms of diverse accuracy and performance. The experiment used 7 versions of opinion mining algorithms based on textual inputs (6 sets of 40 sentences) and compared the integrated result with the individual algorithms' accuracy. The sentiment analysis algorithms were implemented in C++ as *Expert* classes and are named Sentiment with version from 2.0 up to 2.6. The algorithms were rule-based and dictionary-based (occurrence of words), however differed in lexicons they used. Lexicons were lists of affect-annotated words and we used both home-made lexicons [61], [62] as well as referential ones like ANEW [63]. An algorithm providing correct annotations was used as a ''ground truth'' reference for estimation of accuracies.

Independent variable in the experiment is architectural approach with values {Blackboard-based, Scoreboard-based}.

Adaptability was measured with the following metrics: (A1) LOC changed to add algorithm to integration; (A2) LOC changed to remove algorithm from integration; (A3) LOC changed to modify algorithm integration interface; (A4) no of classes to override to change control flow; (A5) no of classes to override to change evaluation function; (A6) LOC required to adjust the new algorithm for integration, (A7) control overhead - LOC for control mechanisms only/LOC total (percentage), (A8) communication overhead - LOC for implementing communication mechanisms/LOC total (percentage).

Another dependent variable we use in this study for evaluation of integration and robustness criteria is emotion recognition accuracy (after integration) denoted as metric (R1). In Blackboard-based approach we combine results from all experts with equal weights, while in Scoreboard-based scenario the integration algorithm takes into account the confidence of specific algorithm that was scored during runtime (by weights of partial results).

Three scenarios of the test were performed: (1) all algorithms scenario (2) three good and one weak algorithms scenario (3) one good and three weak algorithms scenario. The first scenario allows to evaluate integration factor, while the second and third scenario aimed at evaluation of solution robustness. The aim of the latter two scenarios was to check, how an integration algorithm would deal with discrepancies among algorithms. The algorithm preferred consistency among experts, and one might propose a number of alternative approaches. However changes to the algorithm do not influence the comparison between Blackboard-based and Scoreboard-based architecture.

To sum up, the thesis of the experiment might be formulated as follows: Scoreboard-based architectural approach provides higher accuracy of integrated result compared to Blackboard-based approach.

E. SIMULATION 1

As sentiment mining algorithms tested in experiment 2 did not differed in accuracy significantly, the scenarios of strong-weak algorithms combination were inconclusive. Therefore a simulation was implemented and performed. Simulation used mocked-up algorithms as Experts that on delay-based basis returned appropriate (correct/incorrect) sentiment analysis result. As the algorithm prefers consistency (in a real-case scenario when the ''ground truth'' remains unknown) 'correct' algorithms generated random quasi-consistent output from (0.2;0.8) range, while 'incorrect' algorithm provided an inconsistent value from $(-1; 0)$ range. The independent variable was architectural approach with values {Blackboard-based, Scoreboard-based}, while dependent variables represented robustness. In such simulation accuracy (R1) could not be measured (mocked-up algorithms), therefore other two metrics for robustness were used: (R2) execution count - number of executions of specific algorithm (the one that was incorrect, after a number of turns gets a lower and lower score and should be called less frequently); (R3) score obtained by an algorithm after all stages have passed.

F. SIMULATION 2

The second simulation was performed in order to evaluate scalability criteria as a sub-criteria for efficiency. 20 *Expert* class instances were generated with a randomized delays and were grouped into 4 groups of 5 *Experts* each, with a comparative mean and sum of delays. Then the integration algorithm was performed using group 1, group 1+2, group 1+2+3 and group 1+2+3+4. Expert delays are provided in Table 1.

The independent variable was architectural approach with values {Blackboard-based, Scoreboard-based}, while dependent variables was execution time (E2 metric) in milliseconds. Tests were executed on CPU Intel Core i5-4200M 2,5GHz 2 cores (with hyper-threading), 4 GB RAM, operating system Windows 7 Professional during low system load.

MOST WIEDZY Downloaded from mostwiedzy.pl

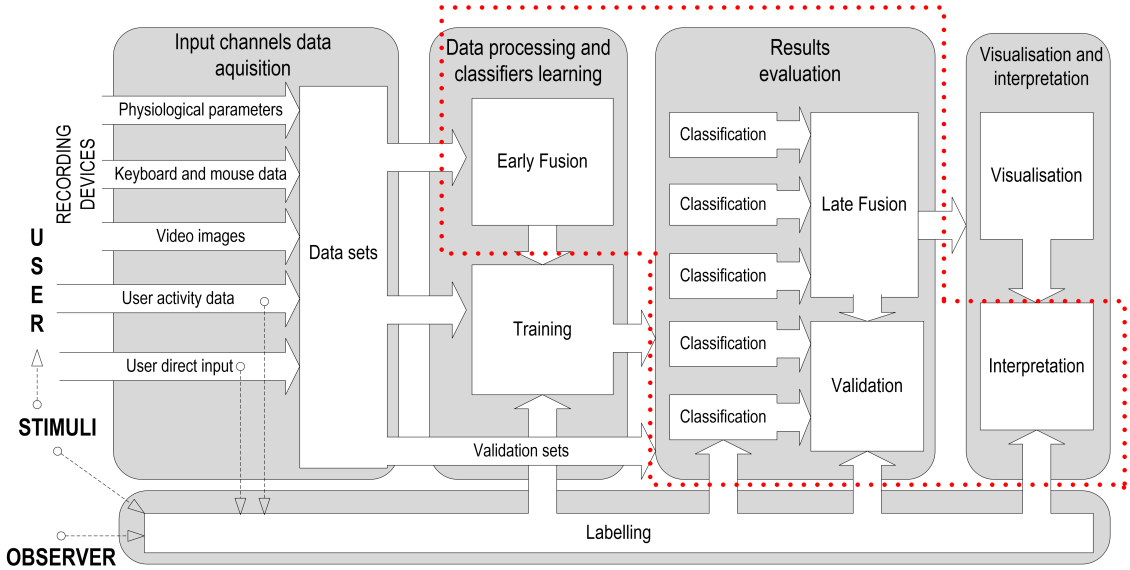

FIGURE 2. Conceptual model of integration solution for Emotion Monitor stand [64].

TABLE 1. Delays for mocked-up algorithms in Simulation 2 (scalability scenario).

Expert no	Delays Group 1 [ms]	Delays Group 2 [ms]	Delays Group 3 [ms]	Delays Group 4 [ms]
1	839	2199	992	1299
2	2305	839	1754	839
3	2048	1254	1706	1554
4	591	950	948	950
5	950	1486	1349	2086
Mean delay per group	1346,6	1345,6	1349,8	1345,6
Sum of delays per group	6733	6728	6749	6728

G. CASE STUDY OF EMOTION MONITOR

The Scoreboard design pattern was applied as a foundation of architecture in a Emotion Monitor stand and this implementation was used as a case study in this paper. We used ATAM qualitative approach for evaluation of architectural design of the Emotion Monitor solution.

In 2013 a project was started at Gdansk University of Technology (GUT) to build an emotion monitoring stand that uses existing technologies in order to extend human-systems interaction with emotion recognition and affective intervention. The Emotion Monitor stand objective was to conduct experiments on computer users affective states retrieval and analysis. The stand is equipped with computers, cameras and a set of biosensors, which allow to monitor user activities and record multiple user observation channels at the same time. The concept of the stand assumed combining multiple modalities used in emotion recognition in order to improve the accuracy of affect classification [54].

Software layer of Emotion Monitor includes an application to store and track biometric data, tools for observation and recording of video images, keyboard and mouse usage tracker and user activity logger. Apart from the applications recording input channels, the main Emotion Monitor's application is the one that combines input channels and multiple classifiers in order to provide an affective state estimate. The result might be displayed with diverse visualization tools (general or dedicated for emotion representations).

Integration of the existing technologies, input channels and solutions turned out to be very challenging, due to the following reasons:

(1) The available set of emotion recognition solutions fluctuates due to the upgrades, limited-time licensing or acquisition of new solutions;

(2) The emotion recognition solutions differ significantly in terms of technologies, APIs and execution efficiency;

(3) The solutions differ in emotion representation model used (output is not compatible and frequently requires additional mapping);

(4) The solutions differ in terms of reliability of the recognized emotional state, especially while some of the input channels they use are temporarily unavailable.

As a result it turned out that the first challenge we had to face, was proposing the integration layer and proper architecture to address practical application requirements. The conceptual model of the Emotion Monitor software layer is visualized in Fig. 2. Components that use Scoreboard design pattern classes are marked with dotted line.

The main purpose of the integration solution is to perform the integration of emotional activation information from multiple input channels. One might consider early fusion, in which the data is combined to create common feature vectors. This approach has at least two important drawbacks:

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE 2. Coverage of evaluation criteria in experiments and simulations.

Criteria and metrics	Research method					
	Experiment 1	Experiment 2	Simulation 1	Simulation 2	Case study (quantitative)	Case study (qualitative)
Modularization	+ (Table III)				+ (Table IV)	+ (Table XIV, XV, XVI)
Integration		+ (Table V)				+ (Table XIV, XV, XVI)
Adaptability		+ (Table VI, VII)			+ (Table VIII)	+ (Table XIV, XV, XVI)
Robustness		+ (Table IX,X)	+ (Table XI)			+ (Table XIV, XV, XVI)
Efficiency	+ (Table XII)			+ (Table XIII)		+ (Table XIV, XV, XVI)

timing synchronization and temporary unavailability of input channels. Late fusion that is based on the integration of the recognized emotional states from different classifiers suffers from diverse emotion representation models and lack of mapping between them. Facial expression analysis algorithms for emotion recognition use Ekman's Facial Coding System and provide six basic emotions as a result. The biosignals are best in recognition of the arousal dimension of emotional state, and not the valence (positive and negative experience might cause the same activation of the nervous system). The integration solution of the stand supports mainly the late fusion approach [64].

In Emotion Monitor software layer Scoreboard design pattern is implemented as follows:

- Emotion recognition algorithms and of-the-shelf solutions are implemented using *Experts* class interface. Implementation supports algorithms written in Python, Java, C++ or C# programming language and additionally any external program, communicating by command line might be attached. We have already successfully integrated Java, C++ and C# classifiers as well as external batch program for emotion recognition. As the integration tool is written in C#, the C# algorithms (experts) don't require a wrapper class. For the other technologies wrappers were implemented. Only wrapper implementation for Python language was not verified, because we lack emotion recognition algorithms in this technology.
- Blackboard and Scoreboard are implemented using a RabbitMQ component and potentially each solution that is able to communicate with the queue system might be integrated. *Experts* are expected to work on input channels data and send their hypothesis on a human emotional state along with a time stamp and evaluation of confidence. The data provided by the *Experts* are shared. *Experts* might share a final outcome as well as some intermediate results.
- *Moderator* class instance uses a number of techniques and design approaches to provide an agreed, reliable result. One of the aspects of the integration is the evaluation of *Experts* results consistency and accuracy in a certain context.
- The application directly supports late fusion to integrate hypothesis on an emotional state, however early fusion and hybrid fusion approaches are possible, although were not used in this study.
- The final emotion states and experts scores are accessible via API, which enables multiple presentation forms.

Evaluation of the Emotion Monitor architecture was performed. First, the modularization and adaptability metrics were provided. We also applied ATAM qualitative method for multifaceted evaluation of architectural design. We have chosen ATAM method based on the literature review (Section 2.3) as the one that allows to consider multiple quality attributes and architectural approaches and to perform some trade-off analysis.

H. COVERAGE OF CRITERIA

Only selected criteria are covered by specific experiment or simulation. Table 2 presents coverage (with + indicating that this criteria was covered). Additionally tables with results were referenced. Please note that experiments' and simulations' results are reported with quantitative metrics, while case study results are reported also with descriptive scenarios and architectural approaches of qualitative nature.

V. RESULTS

Results are organized under criteria, while all tables are clearly named with experiment or simulation number that were the sources of the data. Please refer to Table 2 while looking for specific experiment/simulation results.

A. MODULARIZATION

In the first experiment Scoreboard-based architecture was compared with Blackboard-based one and no-pattern solution. The modularization metrics results obtained in Experiment 1 are provided in Table 3. Metrics are not evenly distributed throughout the code, therefore mean, standard deviation and maximum values were provided for the metrics

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE 3. Modularization metrics in Experiment 1 with text processing algorithms.

Modularization metric	Statistic	Architectural approach		
		Scoreboard	Blackboard	No pattern
(M1) number of distinctive classes	count	26	24	23
(M2) methods per class ratio	mean (SD)	2,88 (0,65)	3 (0,51)	2,13 (0,46)
	max	5	5	4
(M3) LOC per method ratio	mean (SD)	22,73 (8,98)	15,63 (7,49)	11,96 (15,55)
	max	47	45	88
(M4) Cyclomatic complexity of methods	mean (SD)	2,49 (1,85)	2,07 (1,14)	1,67 (0,59)
	max	15	5	4
(M5) Depth of inheritance	mean (SD)	1,77 (0,43)	1,83 (0,38)	1,87 (0,34)
	max	2	2	2
(M6) Class coupling	mean (SD)	4,08 (5,85)	3,17 (5,77)	4,43 (5,41)
	max	32	30	29
(M7) Method coupling	mean (SD)	2,27 (3,69)	1,65 (3,47)	2,53 (4,09)
	max	32	30	29

apart from M1 (class count). The lower values are better for all metrics M1- M7.

No-pattern architecture is the simplest one. It has the lowest values of class count (M1), LOC per method ratio (M3) as well as cyclomatic complexity of methods (M4). Blackboard-based architecture has one more class, while Scoreboard-based architecture has 3 more classes and the solutions are more complex with regard to no-pattern solution. The solution based on Blackboard pattern scores lower for class and method coupling (M6-M7). The solution based on Scoreboard pattern has the lowest depth of inheritance (M5), while class and method coupling is better with regard to no-pattern condition.

Modular architectures should exhibit low coupling. The modularization metrics from the experiment do not show a preferred solution - some trade-offs between architectures are identified instead. More complexity is obtained with lower coupling between classes and methods.

The same modularization metrics were measured for the case study (implementation of emotion recognition solution) and are provided in Table 4. Please note that the distribution of metrics values is not even nor close to normal among classes and methods. Means and standard deviations in that condition might be misleading, however were provided for comparability with experiment 1 architectures.

TABLE 4. Modularization metrics of emotion monitor implementation (Case study).

Metric of code	Statistic	Emotion Monitor
(M1) number of distinctive classes	count	25,00
(M2) methods per class ratio	mean (SD)	5,52 (6,86)
	max	27,00
(M3) LOC per method ratio	mean (SD)	22,16 (40,00)
	max	188,00
(M4) Cyclomatic complexity of methods	mean (SD)	2,01 (3,55)
	max	32,00
(M5) Depth of inheritance	mean (SD)	2,64 (1,2)
	max	4,00
(M6) Class coupling	mean (SD)	6,2 (6,97)
	max	28
(M7) Method coupling	mean (SD)	1,88 (3,15)
	max	21

Case study of Emotion Monitor allows to evaluate modularization in a more complex, real-life solution. The chosen architectural approach allowed for:

- separation of emotion recognition algorithms from communication and control;
- independence of emotion recognition algorithms from each other (no knowledge nor communication is required for the modules to operate);
- separation of emotion estimates integration function (in Moderator class);
- independent evaluation of modules (in Arbiter class).

Although in Emotion monitor class count (M1) is similar to the solutions in experiment 1, the other metrics exhibit some differences. Class and method coupling (M6-M7) preferred values should be lower than 10, which is true on the average, however there are 2 classes with a coupling metrics higher than 20.

Average methods per class ratio (M2) in the application is equal to 5.52, while LOC per method ratio (M3) has a mean of 22 and a maximum of 188. The depth of inheritance is not higher than 4, while coupling of classes and methods acceptable. The maximum value of coupling (M6) occurred in the communication class, which uses a number of external libraries. 19 out of 25 classes have one or none dependencies.

To sum up, the metrics show that Scoreboard-based solution is modular and comparable in terms of modularization to Blackboard-based and no-pattern condition.

B. INTEGRATION

Integration is a quality feature that represents: the ability of the system to integrate diverse solutions. The diversity might result from algorithms, modalities, technologies etc. Integration feature could be evaluated in terms of correctness, performance and ease-of-use. Integration is connected with other quality features and in experiment 2 with sentiment analysis algorithms multiple metrics were measured, but not

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE 5. Accuracy of integrated and partial results in Experiment 2 with sentiment analysis algorithms.

Algorithm	Accuracy per set of sentences [In1]					
	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
S 2.0	72,16	71,64	65,70	72,24	71,49	70,98
S 2.0.1	71,63	73,36	68,42	72,65	72,04	70,10
S 2.1	71,87	69,03	68,09	74,00	67,66	70,82
S2.2	71,87	74,06	68,42	72,64	72,04	70,10
S 2.2.1	73,42	73,39	68,42	74,28	69,12	70,10
S 2.2.2	70,23	na	75,00	64,27	84,58	70,04
S 2.3 AD	67,61	63,95	74,33	68,64	70,00	67,63
Blended (Blackboard-based)	**84,39**	**82,91**	**79,10**	**80,92**	**89,04**	**74,09**
Blended (Scoreboard-based)	**87,05**	**88,43**	**81,06**	**81,48**	**82,60**	**81,52**

all are presented in this section. Ease of use is mostly related to adaptability and is presented in section 5.3, while performance in section 5.5. In table 5 we present results of the integration measured in terms of correctness - accuracies in this case. Accuracies are provided for each of the integrated algorithms as well as for Blackboard-based integration results and Scoreboard-based integration results.

Integration feature was tested on multiple versions of sentiment analysis algorithm from text and multiple versions of affect-annotated lexicons, as described in section 4.4.

Integrated results in most cases, both for Scoreboard-based and Blackboard-based architectures, were better than individual algorithms, which confirms observations from previous research. In the case of Blackboard-based solution all algorithms results were taken into account, as no information is available to differentiate between them. In Scoreboard-based solution selection of algorithms' results to be blended depends on the algorithm score, as gathered in *Scoreboard* class. The scores are results of the previous executions (probability of inclusion depending on the score).

When algorithms are dynamically selected by score, the accuracies are better than when blending all algorithms results. This allows the integration function for run-time adaptation to changing circumstances.

The data allows to accept the thesis that the Scoreboard-based architecture provides better integration results in terms of accuracy. More adaptability and performance metrics from the experiment are provided in the following sections.

C. ADAPTABILITY

Adaptability in this study is understood as a feature of the architectural design that allows for easy attachment/detachment/swap of algorithms providing results. Table 6 presents adaptability metrics from experiment 2 with sentiment analysis algorithms.

TABLE 6. Adaptability metrics (A1-A6) in Experiment 2 with sentiment analysis algorithms.

Metric	Blackboard	Scoreboard	Additional information
(A1) LOC changed to add algorithm	1	1	No need for recompilation
(A2) LOC changed to remove algorithm	1	1	No need for recompilation
(A3) LOC changed to swap algorithm	1	1	No need for recompilation
(A4) no of classes to change control flow	1	1	Override Control module (one class)
(A5) no of classes to change evaluation function	1	1	Override evaluation module (one class)
(A6) LOC required to adjust the new algorithm for integration	87-158	87-158	Communication with Message Broker – once per each technology (reuse possible)
	33-111	33-111	Interface for control flow – once per each technology (reuse possible)
	162	162	Expert wrapper class – once per each technology (reuse possible)
	18	18	Change of parameters – once per each technology (reuse possible)

Adaptability is measured with metrics representing size of code (lines of code, class count) that needs to be changed/added in order to adapt the software to a desired change. The lower the values of metrics, the better adaptability of the solution. Experiment 2 reveals no difference between Blackboard-based and Scoreboard-based architecture. For both solutions metric values are low (1 line of code or one class to be changed) due to the chosen approach of 'inversion of control' - algorithms are defined in external configuration file and launched by injection. All modules must be defined in configuration file with the following parameters: *services* – determines function of a module: expert, moderator or arbiter module, *to* – indicates class of module for execution, *type* – type of inclusion, used only in experts modules. Each configuration file must define at least one expert for Blackboard-based approach and at least one expert, one moderator, one arbiter module for Scoreboard-based approach. This approach is representing late binding paradigm and is preferable to adaptability criteria.

If an algorithm is an external one (emotion recognition algorithm from other technology, launched by API or launched in command line), it requires adaptation for integration and A6 metric provides the estimated LOC count for the algorithm adjustment. Metric A6 might be not informative enough - it is an absolute value with no reference to total

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE 7. Adaptability metrics (overheads A7-A8) in Experiment 2 with sentiment analysis algorithms.

Algorithm	Code size [LOC]			(A7) Control overhead [%]	(A8) Communication overhead [%]
	Total	Com	Control		
S 2.0 (C++)	608	158	55	9,1	26,0
S 2.0.1(C++)	800	158	55	6,9	19,8
S 2.1 (C++)	713	158	55	7,7	22,2
S 2.2 (C++)	799	158	55	6,9	19,8
S 2.2.1 (C++)	806	158	55	6,8	19,6
S 2.2.2 (C++)	841	158	55	6,5	18,8
S 2.3 (C++)	908	158	55	6,1	17,4

TABLE 8. Adaptability metrics (overheads for diverse technologies) in Emotion Monitor case study.

Algorithm	Code size [LOC]			(A7) Control overhead [%]	(A8) Communication overhead [%]
	Total	Com	Control		
S 2.3 (C++)	908	158	55	6,1	17,4
Sam1 (Java)	424	92	96	22,6	21,7
Sam2 (Java)	269	92	96	35,7	34,2
Sam3 (C#)	330	87	111	33,6	26,4
FERT (C#)	242	87	33	13,6	36,0
FERT2 (C#)	207	87	33	15,9	42,0
FR (C#)	413	87	45	10,9	21,1
XE (C#)	321	87	33	10,3	27,1

code size, therefore additional metrics A7 and A8 were counted for the sentiment analysis algorithms in experiment 2 that represented control and communication overhead as a relative value (to total code size) - see Table 7.

The values provided in Table 7 are valid for both Scoreboard-based and Blackboard-based architecture. as experts were treated with black-box approach, the adaptation requires an implementation of a wrapper, which is exactly the same for both architectural approaches. At this layer of the solution there is no difference between the two approaches.

The algorithms in experiment 2 were all implemented with C++. In case study of Emotion Monitor more emotion recognition solutions were taken into account, including solutions in Java and C#, as well as external software that required cross-technology wrappers. Table 8 presents adaptability metrics A7 and A8 for the algorithms in Emotion Monitor: with S 2.3 sentiment analysis algorithm, Sam1 and Sam2 emotion recognition algorithms written in Java, FERT and FERT 2 algorithms based on facial expression analysis and two external solutions: FR - Noldus Face Reader off-the-shelf software merged by API and XE - external software launched from command line.

Please note that all of the different emotion recognition solution technologies were merged successfully in Scoreboard-based architecture of Emotion Monitor.

The additional goal of Emotion Monitor study was to evaluate the overhead of the integration and evaluation solution above the isolated emotion recognition code. Algorithms in following technologies: C++, Java and C# were measured in order to provide an estimate of the overhead.

The overhead on control flow varies from 6% up to 35%, while the overhead on communication – from 23% op to 49%. The algorithms used in this study are rather simple ones and therefore in some cases the overhead is quite significant. However, the overhead measured in LOC is (almost) constant for certain technology, which might be a good prerequisite for scalability of the solution. Moreover, the integration code developed for one algorithm might be re-used for another one in the same technology.

As a result, the observations concerning adaptability might be formulated as follows:

- emotion recognition algorithms are easily attached, detached or modified without influencing others;
- complexity of algorithms adaptation for integration with the solution depends mainly on technology;
- integration function and evaluation function are replaceable without influencing emotion recognition solutions;

The quantitative data and qualitative observations allow to accept the thesis that the chosen architecture is adaptable and Scoreboard-based architecture is at the same adaptability level as Blackboard-based one, which was used as a reference in this study.

D. ROBUSTNESS

In section 5.2 we reported results of integration in experiment 2, showing benefits of Scoreboard-based architectural approach. This section reports robustness, which is understood in this study as a feature of the recognition software that benefits from proper integration algorithm by limited influence of "faulty" expert on the final solution. The feature was evaluated within experiment 2 by choosing specific subsets of emotion recognition software. Results of scenario 1 (integration of results from all algorithms) was already shown in Table 5. Table 9 and 10 report results of two other scenarios that combine a limited number (4) of "strong" and "weak" algorithms.

Please note that in the case of experiment 2 with sentiment analysis algorithms the algorithms have similar accuracies. The one "weaker" for the first set of sentences (S2.3) was actually "stronger" for the third set. Robustness feature is however represented with a better accuracy score for blended result. For sentence sets: 1, 2, 5 and 6 the integrated result is better or as good as the best algorithm result (in accuracy). For the sentence set 3 and 4 the integrated result is worse than the best of the algorithms, but still better than the other two.

In the second combination of sentiment analysis algorithms for the sentence sets: 1, 3 and 6 the integrated result is better or as good as the best algorithm result. For the sentence

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE 9. Accuracy of emotion recognition from text as an estimate of robustness characteristics (Experiment 2 Scenario 1).

Algorithm	Accuracy per set of sentences (R1)					
	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
S 2.0	72,16	71,64	65,70	72,24	71,49	70,98
S 2.1	71,87	69,03	68,09	74,00	67,66	70,82
S 2.2	71,87	74,06	68,42	72,64	72,04	70,10
S 2.3	67,61	63,95	74,33	68,64	70,00	67,63
Blended (Scoreboard -based)	**73,09**	**74,22**	**72,17**	**73,10**	**72,20**	**70,89**

TABLE 10. Accuracy of emotion recognition from text as an estimate of robustness characteristics (Experiment 2 Scenario 2).

Algorithm	Accuracy per set of sentences (R1)					
	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
S 2.2	71,87	74,06	68,42	72,64	72,04	70,10
S 2.2.1	73,42	73,39	68,42	74,28	69,12	70,10
S 2.2.2	70,23	Na	75,00	64,27	84,58	70,04
S 2.3	67,61	63,95	74,33	68,64	70,00	67,63
Blended (Scoreboard-based)	73,27	73,22	74,99	73,67	77,80	70,42

TABLE 11. Robustness metrics in Simulation 1 with mocked-up algorithms.

Name	Condition	(R2) Execution count (out of 100)	(R3) Score achieved
Algorithm 1	Strong	98	0,989
Algorithm 2	Strong	99	0,993
Algorithm 3	Strong	98	0,993
Algorithm 4	Reference	100	0,990
Algorithm 5	Weak	42	0,465

set 2, 4 and 5 the integrated result is worse than the best of the algorithms, but still better than the other two. Please note that even if one of the algorithms was unable to provide a result (algorithm S2.2.2 for Set 2), the solution was still able to provide a result combining the ones that were available. This shows that robustness feature might be also understood as ability to provide a score despite temporal unavailability of some results/some algorithms/some observation channels.

Because the sentiment analysis algorithms in experiment 2 were close in terms of accuracy, we have performed a simulation that aimed at showing the scenario of larger discrepancies between the accuracy of integrated algorithms, as described in section 4.5. Table 11 presents execution count (R2 metric) as well as score achieved by an algorithm (R3 metric) from simulation 1.

TABLE 12. Efficiency metrics in Experiment 1 with text processing algorithms.

Efficiency metric	Statistic	Architectural approach		
		Scoreboard	Blackboard	No pattern
(E1) no of expert executions [count]	Mean (SD)	1 295,5 (320,4)	2000*+1000 (0)	6 209,6 (349,3)
	Max	1582	2000*+1000	7 013
(E2) execution time [s]	Mean (SD)	401,1 (118,5)	570,2 (200,2)	559,5 (168,1)
	Max	611,5	984,8	954,5
(E3) initialisation time [ms]	Mean (SD)	39,2 (48,6)	18,6 (28,2)	23 (25,9)
	Max	125,0	94,0	78,0
(E4) paged and non-paged memory used [kB]	Mean (SD)	13,6 (0,9)	12,8 (0,3)	16,7 (1,9)
	Max	16,0	13,2	19,3

In the simulation execution was performed 100 times to get more significant observations, as the choice is not set, but randomized with probabilities based on the scores of specific experts. The defective expert was quickly spot and after 100th run got it score of 0,46. As a result, it was chosen for execution only 42 times, while the others were executed more intensively.

The integration function is not perfect, as it was mainly based on the consistency of the provided results. Therefore one strong algorithm vote weights less than three votes of weak algorithms. One might consider better evaluation and integration functions and in the solution of Emotion Monitor integration it requires changing one class only.

As a result, the observations concerning integration and robustness might be formulated as follows:

- integrated results are slightly more accurate than the ones of individual algorithms, assuming, the proportion of 'strong' and 'weak' algorithms is balanced;
- it is possible to automatically spot the low-accuracy algorithm on the run and to adapt the frequency of its execution.

The observations allow to accept the thesis that the chosen architecture provides integration and some robustness to low accuracies in emotion recognition.

E. EFFICIENCY

Optimization of software architecture according to quality criteria, such as adaptability or modularity, might compromise performance. Therefore in experiment 1 with text processing algorithms performance metrics were gathered. The results for Scoreboard-based, Blackboard-based and no-pattern architectures are presented in Table 12.

There were 20 algorithms in each solution, which should work in a certain order, which was unknown a priori. The best (optimal) solution required exactly one execution on each algorithm. 1000 different inputs were taken as test cases.

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE 13. Efficiency metrics in Simulation 2 with mocked-up algorithms (scalability scenario).

Efficiency metric (E2) execution time [s]	Statistic	Architectural approach	
		Blackboard-based	Scoreboard-based
with 5 experts	Mean (SD)	5,902 (0,229)	4,695 (1,053)
	Max	7,418	7,603
with 10 experts	Mean (SD)	16,561 (0,184)	5,433 (0,864)
	Max	17,457	8,490
with 15 experts	Mean (SD)	22,668 (0,372)	5,144 (1,020)
	Max	24,667	8,323
with 20 experts	Mean (SD)	44,933 (0,385)	7,718 (1,604)
	Max	46,694	12,041

Execution times (E2 metric), both average and maximal, are the lowest for Scoreboard-based architectural approach with regard to both Blackboard and no pattern solution. This might be explicitly assigned to the number of expert algorithms execution (see E1 metric).

The Scoreboard-based solution learns the order by giving scores for previous algorithm. After training period (100 inputs processed - the number is configurable), each algorithm has already a score stored in *Scoreboard* instance and the score is used for determining the preferred order.

In Blackboard-based approach, each algorithm is launched at least once. In the approach, the first round is necessary to determine the order of experts (partial data - 5% if input - are processed for evaluation), and then specific order is executed. One might propose another algorithm for determining the order, however, still is shall be executed each time, as there is no place to store the evaluation data for the following inputs. In no-pattern reference condition, algorithms were executed in randomized order. Execution count marked with $*$ denotes partial executions, while the other number reflects execution on entire input.

The Blackboard-based solution had the lowest execution count only for entire input execution. For both, partial and entire input processing, count is preferable for Scoreboard-based approach. The difference is also reflected in execution times (E2 metric). Please note that initialization time (E3 metric) is the longest for Scoreboard-based approach, as additional classes must be initialized. Memory used is the highest for no-pattern approach, the lowest for Blackboard-based approach and medium for Scoreboard-based architecture.

Another simulation was performed in order to evaluate scalability criteria as a sub-criteria for efficiency (see section 4.6 for simulation description). The Scoreboard-based and Blackboard-based solutions were executed with mocked-up algorithms. The results are presented in Table 13 and Fig. 3.

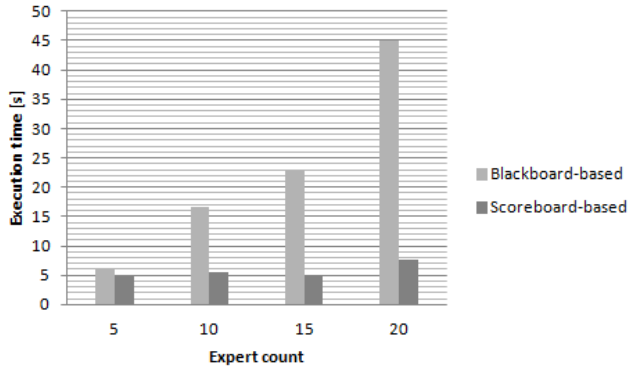

FIGURE 3. Scalability - growth of execution time as a function of expert count.

Execution time grows with expert count for both architectural solutions, however the growth rate is lower for Scoreboard-based solution. For Blackboard-based architecture it is quasi-linear (nlogn), while for Scoreboard-based it is sub-linear (loglogn). Please note that this metrics does not reflect computational complexity, as n is not an input size, but experts count instead.

The observations allow to accept the thesis that the Scoreboard-based architecture provides some improvement to performance in terms of execution time and scalability.

F. CASE STUDY QUALITATIVE EVALUATION WITH ATAM METHOD

Apart from quantitative experiments and simulations, a case study of Emotion Monitor was analyzed in detail with qualitative ATAM method. The method requires three steps: (1) defining scenarios to fulfill by the solution, (2) defining architectural approaches that constitute a solution and (3) mapping the approaches to the scenarios in order to identify non-risks (supported scenarios), risks (scenarios that might be compromised) and sensitivity points (trade-offs). The scenarios for Emotion Monitor are provided in Table 14. The scenarios are attributed to the defined criteria, for compatibility with quantitative analysis.

The next step of ATAM method was to identify key architectural approaches that constitute a solution under evaluation. Fourteen architectural approaches were defined, out of which five (AA1-AA5) characterize both Blackboard-based and Scoreboard-based approach, additional two (AA6, AA7) are results of Blackboard pattern only and another seven (AA8-AA14) might be attributed to Scoreboard-based solution only. Table 15 describes architectural approaches in detail.

Next step of ATAM method is to map scenarios to architectural approaches. The mapping results are summarized in Table 16 with plus assigned to non-risks (scenario supported by an architectural approach), minus assigned to risks (scenario is compromised by an architectural approach) and asterix assigned to sensitivity points. If sensitivity point involves more than one scenario, it identifies a trade-off.

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE 14. ATAM method scenarios descriptions.

Criteria	Scenario	Description
Modularization	Sc 1. Expert separation from control	Emotion recognition algorithms are separated from the control and integration.
	Sc 2. Control and integration separation	Control and integration algorithms are separated from each other.
	Sc 3. Expert separation	Emotion recognition algorithms are separated from each other and do not communicate directly to perform.
Integration	Sc 1. Integration	It is possible to integrate any reasonable amount of emotion recognition solutions that differ in modalities and algorithms.
	Sc 2. Technologies	It is possible to integrate solutions implemented in diverse technologies.
	Sc 3. Off-the-shelf	It is possible to integrate off-the-shelf solutions provided they are loadable from command line or API is available.
	Sc 4. Early fusion in multimodal emotion recognition	All input channels are available for all experts and early fusion is possible in multimodal emotion recognition.
	Sc 5. Late fusion in multimodal emotion recognition	All partial results from all experts are available for all other experts and late fusion might be performed.
	Sc 6. Multistage	Multistage emotion recognition challenge solving (it is possible to define separate set of experts for observational data pre-processing, emotion recognition, emotion integration and emotion models mapping.
	Sc 7. Inconsistency	Inconsistency of emotion recognition results is addressed.
Adaptability	Sc 1. New algorithm	New Expert algorithm is easily attached, not influencing run-time. No need for re-compilation of the entire solution.
	Sc 2. Deleting algorithm	Any chosen Expert algorithm is easily detached, not influencing run-time. No need for re-compilation of the entire solution.
	Sc 3. Change of Expert algorithm	Change of internal implementation of Expert algorithms does not influence run-time. No need for re-compilation of the entire solution.
	Sc 4. Integration function change	Integration algorithm in might be easily changed. No need for re-compilation of the entire solution.
	Sc 5. Sequence of execution is easily modifiable	Sequence of execution of algorithms might be changed and influenced during run-time with no need for recompilation of the entire solution.
Robustness	Sc 1. Low accuracy	It is possible to spot an algorithm of low accuracy
	Sc 2. Impact	Algorithms of low accuracies have a limited influence on final score
	Sc 3. Confidence	Confidence of result is built along with emotion recognition process.
Efficiency	Sc 1. Execution time	Execution time of the entire solution might be adjusted with no limits on the number of emotion recognition algorithms integrated.
	Sc 2. Scalability	The solution is scalable with regard to number of emotion recognition algorithms used.
	Sc3. Efficiency	The most efficient subset of emotion recognition algorithms is performed.

Analysis of the ATAM method map in Table 16 led us to identify a number of consequences of Scoreboard-based architecture. First of all, all modularity and adaptability scenarios are fulfilled supported by the solution due to both Blackboard-based and Scoreboard-based architectural approaches. Among integration scenarios two benefit from Blackboard-to-Scoreboard extension, i.e. late fusion and dealing with inconsistency. The main benefits are identified within robustness and efficiency scenario sets - Scoreboard-based approaches allow to support those scenarios, which Blackboard-based approaches compromises.

Qualitative analysis of Emotion Monitor case study mostly supports the observations from quantitative experiments and simulations.

TABLE 15. ATAM method architectural approaches descriptions.

Architectural approach	Black board	Score board	Description
AA1	X	X	Single separate structure for sharing results among Expert class instances (Blackboard class).
AA2	X	X	Single Controller class instance separated from Expert classes.
AA3	X	X	Multiple Expert classes execute the same task (emotion recognition) with diverse algorithms. Black-box approach to those algorithms is applied.
AA4	X	X	Experts are communicating with Blackboard and Controller, experts are not communicating among each other.
AA5	X	X	Algorithms provide results as a pair of emotion and certainty values.
AA6	X		All Experts are executed per each round.
AA7	X		Randomized or rule-based selection of next Expert to execute.
AA8		X	Experts are extended with possibility to evaluate results of other experts.
AA9		X	Separate structure for storing evaluations of experts (Scoreboard class).
AA10		X	Grouping competitive experts available for launch and execution of a subset of those based on evaluations.
AA11		X	Moderator class responsible for evaluation and resolving inconsistencies among results for a current stage.
AA12		X	Using inversion of control and dependency injection in Controller class.
AA13		X	Using external configuration files for definition of Experts list.
AA14		X	Wrappers are used for integration of external programs and modules in other technologies.

VI. SUMMARY OF RESULTS AND DISCUSSION

A. MAIN FINDINGS

Scoreboard architectural pattern is an extension of design pattern Blackboard. Both patterns hold similar classes: single

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE 16. ATAM method map of scenarios supported by architectural approaches.

Criteria	Scenario	Architectural approaches													
		01	02	03	04	05	06	07	08	09	10	11	12	13	14
Modularization	Sc 1. Expert separation from control		+							+					
	Sc 2. Control and integration separation		+							+					
	Sc 3. Expert separation	+		+	+					+		+			
Integration	Sc 1. Integration	+		+		+	+	+							+
	Sc 2. Technologies	*		+	+										+
	Sc 3. Off-the-shelf	*		+	+										+
	Sc 4. Early fusion in multimodal emotion recognition	+		*	*										
	Sc 5. Late fusion in multimodal emotion recognition				-	+						+			
	Sc 6. Multistage	+		*	*			+							
	Sc 7. Inconsistency					+	-	-		+		+			
Adaptability	Sc 1. New algorithm		+										+	+	
	Sc 2. Deleting algorithm		+										+	+	
	Sc 3. Change of Expert algorithm		+	+									+	+	+
	Sc 4. Integration function change	+											+	+	
	Sc 5. Sequence of execution is easily modifiable		+		+			+					+	+	
Robustness	Sc 1. Low accuracy						-	-	+			+			
	Sc 2. Impact		+				-	-	+			+			
	Sc 3. Confidence			*				-	+	+	+	+			
Efficiency	Sc 1. Execution time	*	*	*			-	-	*		+				
	Sc 2. Scalability			*			-	-			+				
	Sc3. Efficiency		*				-		+	+	+	+			

instance of *Blackboard* class, single instance of *Controller* class and multiple instances of *Expert* class. The problem of integration or evaluation of claims was not solved within Blackboard pattern and this is one of the reasons behind an extension to the *Scorboard* pattern. The new *Scoreboard* pattern apart from instances of *Experts*, *Blackboard* and *Controller* is extended with 3 classes: *Scoreboard* (single instance), *Moderator* as a sub-class of *Expert* class and an *Arbiter* (single instance).

In order to verify the integration solution based on the proposed Scoreboard design pattern a number of studies were performed: two experiments, two simulations and a qualitative analysis of a case study. The results of Scoreboard design pattern evaluation might be summarized as follows:

1. Scoreboard-based solution is modular and comparable in terms of modularization to Blackboard-based condition, which was used as a reference in this study. It is possible to use multiple competing algorithms for the same processing phase, use them in parallel and compare results.
2. Scoreboard-based architecture provides better integration results in terms of accuracy. Integrated results are slightly more accurate than the ones of individual algorithms and Blackboard-based integration, assuming the proportion of 'strong' and 'weak' algorithms is balanced. The results blend into the observations made in the related research so far: the late fusion provides better results than individual algorithms. Both early fusion and late fusion are directly supported by Scoreboard design pattern.
3. The Scoreboard-based architecture is adaptable and in terms of adaptability comparable to Blackboard-based one. Algorithms are easily attached, detached or modified without influencing others and complexity of the algorithms adaptation for integration with the solution depends mainly on technology. Moreover, both integration function and evaluation function are replaceable without influencing the rest of the solution.
4. Multiple algorithms (for integration functions, evaluation, emotion recognition or other processing task) might be evaluated for run-time efficiency. Learning and adaptability to changing circumstances (channel availability, emotion recognition depending on context and environment, etc.) are built into Scoreboard pattern. It directly supports continuous evaluation of the application and its individual components (modules and algorithms) and building

MOST WIEDZY Downloaded from mostwiedzy.pl

confidence of the results and the components during the application runtime.

5. Scoreboard-based approach provides some robustness to invalid (permanently or temporarily) algorithms, temporarily unavailable input channels, temporarily missing algorithm availability. it is possible to automatically spot the low-accuracy algorithm on the run and to adapt the frequency of its execution.
6. There are some trade-offs regarding Scoreboard design pattern - qualitative attributes such as adaptability or robustness are obtained with some cost on performance. The efficiency of the Scoreboard-based solution is lower than sequential execution of algorithms, as communication and control flow overhead exists. The efficiency might be improved by parallel execution or balancing the number of experts to execute.

The quantitative data and qualitative observations allow to accept the thesis that Scoreboard design pattern provides modularization, integration, robustness, adaptability and efficiency for non-deterministic applications integrating multiple and diverse algorithms with large solution space to explore.

B. THEORETICAL AND PRACTICAL IMPLICATIONS

Scoreboard design pattern is dedicated for integration in experimental systems, in systems with a large solution space to explore, systems performing under uncertainty or the ones that aim at evaluation of the algorithms in use.

Scoreboard design pattern might be applied in the following solutions:

- research applications and projects, aiming at evaluation of the algorithms and their optimization,
- non-deterministic applications, when the order of execution is unknown at the design stage and space of possible solutions is extensive,
- solutions that require continuous improvement and evaluation of the changes made.

Scoreboard design pattern might be not efficient in:

- projects with time pressure, as it requires additional implementation effort,
- deterministic applications, when the control flow is known at the design stage,
- solutions that have high performance requirements, as continuous evaluation of modules requires processing time.

Practical implications of this study on multimodal emotion recognition:

- it is possible to use (almost) any algorithm or of-the-shelf solution for emotion recognition, assuming existence of wrapper or API;
- the algorithms might be easily attached and detached, allowing for an individualized setting of the solution per each study and experiment;
- it is possible to integrate the results using late fusion approach, assuming, there is some mapping to the emotion representation model used in the integration layer;
- as algorithms have diverse accuracies and execution time, the integration might be continuous - the information on recognized emotional states might be updated with new data; on-line continuous integration requires solving the issue of diverse latency in processing input channels by different algorithms. Streams of emotion estimations must be somehow synchronized.

There are several consequences of using Scoreboard pattern in the design of the particular system that could be considered as drawbacks:

- difficulty in testing, especially when significant number of experts is involved - the Scoreboard inherits this feature from Blackboard predecessor;
- there is no guarantee for obtaining a satisfactory solution, which is characteristic for non-deterministic systems;
- reduced performance when there is no sufficient criteria that reduces solution space and no of experts to call;
- increased implementation time.

The proposed Scoreboard design pattern provided one of the solutions that address robustness in multimodal recognition, which was considered as a major issue by [22]. Also the challenge of adaptability and weight assignment to the different modalities [44] might be directly supported by the pattern.

There is a challenge that remains unsolved - effective modeling and processing of temporal information [42], which both require further studies. Also accuracy versus performance trade-off, which restricts the usefulness of such systems in practical applications [22], remains in Scoreboard-based solutions.

C. VALIDITY OF THE STUDY

The authors are aware of the fact that this study is not free of some limitations. First of all, from all the quality criteria listed in the related work section, only 5 were addressed in this study. Although we have used the GQM approach to choose them, one might consider other criteria (e.g. scalability) as more important. Metrics for the criteria were chosen based on availability in particular environment and experiment and do not follow any literature-based framework. The choice was justified by expected outcome of the Scoreboard pattern on the resulting architecture. One might propose another set of metrics for the chosen criteria.

Moreover, in this study we have treated the integration solution as a black-box and no details were provided on the used integration nor evaluation algorithms. No detailed information was provided on which emotion recognition algorithms were used in the study. This approach was chosen intentionally, as the algorithms might be changed and adjusted and the goal of this study was to evaluate integration and not individual algorithms.

Not all criteria were covered by all experiments (see Table 2). In experiment 1 text processing algorithms did not differ in accuracy, therefore robustness and availability criteria was not evaluated. Experiment 2 depended on

MOST WIEDZY Downloaded from mostwiedzy.pl

sentiment analysis algorithms used and they did not differed significantly enough for evaluation of some integration scenarios. Therefore additional simulations were performed in order to address specific scenarios for the criteria.

Case study was evaluated mostly based on qualitative approach, although some metrics were provided (Table 4 and 8). In the case study only Scoreboard-based architecture was explored in detail.

Although some limitations exits, we are convinced that the study hypothesis could be accepted. Moreover some of the shared observations and lessons learned might be interesting for the other researchers and practitioners in affect recognition and other multimodal recognition challenges.

VII. CONCLUSIONS

The study proposed a new design pattern, called Scoreboard, which being an extension of Blackboard pattern, addresses adaptability and robustness in multimodal recognition systems.

The reliability and the accuracy of an estimate of an emotional state depends on many conditions: availability and quality of the input channels, environmental variables (temperature, light) as well as on the expressivity of an individual.

The proposed Scoreboard design pattern allows to address the issues that arise in emotion recognition, as well as in some other experimental solutions.

An open sample implementation of the Scoreboard pattern is available in the following Bitbucket repository: https://bitbucket.org/pgscoreboard/scoreboard.

ACKNOWLEDGMENT

The authors would like to thank following MSc and Eng. students of Gdansk University of Technology: Adam Polak, Maciej Marszałek and Paulina Barczyńska for implementation of software used in the study.

REFERENCES

[1] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language*. Berkeley, CA, USA: Oxford Univ. Press, 1977.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns CD: Elements of Reusable Object-Oriented Software* (Addison-Wesley Professional Computing). Reading, MA, USA: Addison-Wesley, 1996.

[3] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: A System of Patterns*, vol. 2. Hoboken, NJ, USA: Wiley, 2000.

[4] J. Hannemann and G. Kiczales, "Design pattern implementation in Java and aspectJ," *ACM SIGPLAN Notices*, vol. 37, no. 11, pp. 161–173, 2002.

[5] T. Erl, *SOA Design Patterns*. Upper Saddle River, NJ, USA: Prentice-Hall, 2009.

[6] J. Tidwell, *Designing Interfaces: Patterns for Effective Interaction Design*. Newton, MA, USA: O'Reilly, 2005.

[7] E. Cambria, Y. Song, H. Wang, and N. Howard, "Semantic multidimensional scaling for open-domain sentiment analysis," *IEEE Intell. Syst.*, vol. 29, no. 2, pp. 44–51, Mar./Apr. 2014.

[8] S. Dublin *et al.*, "Natural language processing to identify pneumonia from radiology reports," *Pharmacoepidemiol. Drug Saf.*, vol. 22, no. 8, pp. 834–841, Aug. 2013.

[9] R. Lienhart, L. Liang, and A. Kuranov, "A detector tree of boosted classifiers for real-time object detection and tracking," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2003, p. II-277.

[10] A. Landowska, "Emotion monitor—Concept, construction and lessons learned," in *Proc. Federated Conf. Comput. Sci. Inf. Syst. (FedCSIS)*, 2015, pp. 75–80.

[11] F. Buschmann, R. Meunier, and H. Rohnert, *Pattern-Oriented Software Architecture: A System of Patterns*. Hoboken, NJ, USA: Wiley, 1996.

[12] T. S. Levitt, "Choosing uncertainty representations in artificial intelligence," *Int. J. Approx. Reason.*, vol. 2, no. 3, pp. 217–232, Jul. 1988.

[13] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Comput. Speech Lang.*, vol. 16, no. 1, pp. 69–88, Jan. 2002.

[14] A. Norouzian and R. Rose, "An approach for efficient open vocabulary spoken term detection," *Speech Commun.*, vol. 57, pp. 50–62, Feb. 2014.

[15] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM J. Res. Develop.*, vol. 3, no. 2, pp. 114–125, 1959.

[16] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural Comput.*, vol. 1, no. 3, pp. 372–381, Sep. 1989.

[17] J. Daciuk, S. Mihov, B. W. Watson, and R. E. Watson, "Incremental construction of minimal acyclic finite-state automata," *Comput. Linguistics*, vol. 26, no. 1, pp. 3–16, Mar. 2000.

[18] J. F. Ackermann and M. S. Landy, "Suboptimal decision criteria are predicted by subjectively weighted probabilities and rewards," *Attention, Perception, Psychophys.*, vol. 77, no. 2, pp. 638–658, Feb. 2015.

[19] A. Ampatzoglou, S. Charalampidou, and I. Stamelos, "Research state of the art on GoF design patterns: A mapping study," *J. Syst. Softw.*, vol. 86, no. 7, pp. 1945–1964, 2013.

[20] V. Lesser, R. Fennell, L. Erman, and D. Reddy, "Organization of the hearsay II speech understanding system," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-23, no. 1, pp. 11–24, Feb. 1975.

[21] A. Landowska, G. Brodny, and M. R. Wrobel, "Limitations of emotion recognition from facial expressions in e-learning context," in *Proc. 9th Int. Conf. Comput. Support. Educ. (CSEDU)*, vol. 2, 2017, pp. 383–389.

[22] S. Poria, E. Cambria, R. Bajpai, and A. Hussain, "A review of affective computing: From unimodal analysis to multimodal fusion," *Inf. Fusion*, vol. 37, pp. 98–125, Sep. 2017.

[23] H. Gunes and B. Schuller, "Categorical and dimensional affect analysis in continuous input: Current trends and future directions," *Image Vis. Comput.*, vol. 31, no. 2, pp. 120–136, Feb. 2013.

[24] Z. Zeng, M. Pantic, G. I. Roisman, and T. S. Huang, "A survey of affect recognition methods: Audio, visual, and spontaneous expressions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 1, pp. 39–58, Jan. 2009.

[25] T. Partala and A. Kallinen, "Understanding the most satisfying and unsatisfying user experiences: Emotions, psychological needs, and context," *Interact. Comput.*, vol. 24, no. 1, pp. 25–34, Jan. 2012.

[26] R. L. Hazlett and J. Benedek, "Measuring emotional valence to understand the user's experience of software," *Int. J. Hum.-Comput. Stud.*, vol. 65, no. 4, pp. 306–314, Apr. 2007.

[27] P. G. Zimmermann, P. Gomez, B. Danuser, and S. G. Schär, "Extending usability: Putting affect into the user-experience," in *Proc. NordiCHI*, 2006, pp. 27–32.

[28] H. H. Binali, C. Wu, and V. Potdar, "A new significant area: Emotion detection in E-learning using opinion mining techniques," in *Proc. IEEE Int. Conf. Digit. Ecosyst. Technol.*, Jun. 2009, pp. 259–264.

[29] A. Landowska, "Affect-awareness framework for intelligent tutoring systems," in *Proc. 6th Int. Conf. Hum. Syst. Interact. (HSI)*, 2013, pp. 540–547.

[30] A. Landowska, M. Szwoch, and W. Szwoch, "Methodology of affective intervention design for intelligent systems," *Interact. Comput.*, vol. 28, no. 6, pp. 737–759, 2016.

[31] K. Hone, "Empathic agents to reduce user frustration: The effects of varying agent characteristics," *Interact. Comput.*, vol. 18, no. 2, pp. 227–245, 2006.

[32] A. Kolakowska, A. Landowska, M. Szwoch, W. Szwoch, and M. R. Wrobel, "Emotion recognition and its applications," in *Human-Computer Systems Interaction: Backgrounds and Applications*. Cham, Switzerland: Springer, 2014, pp. 51–62.

[33] L. Chittaro and R. Sioni, "Affective computing vs. Affective placebo: Study of a biofeedback-controlled game for relaxation training," *Int. J. Hum.-Comput. Stud.*, vol. 72, pp. 663–673, Aug./Sep. 2014.

[34] T. Partala and V. Surakka, "The effects of affective interventions in human-computer interaction," *Interact. Comput.*, vol. 16, no. 2, pp. 295–309, 2004.

MOST WIEDZY Downloaded from mostwiedzy.pl

[35] A. Kołakowska, A. Landowska, W. Szwoch, W. R. Wróbel, and M. Szwoch, "Emotion recognition and its application in software engineering," in *Proc. 6th Int. Conf. Hum. Syst. Interact. (HSI)*, 2013, pp. 532–539.
[36] M. R. Wrobel, "Emotions in the software development process," in *Proc. 6th Int. Conf. Hum. Syst. Interact.*, Apr. 2013, pp. 518–523.
[37] A. Landowska, "Emotion monitoring—Verification of physiological characteristics measurement procedures," *Metrol. Meas. Syst.*, vol. 21, no. 4, pp. 719–732, 2014.
[38] H. Gunes and M. Piccardi, "Affect recognition from face and body: Early fusion vs. Late fusion," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, vol. 4, Oct. 2005, pp. 3437–3443.
[39] I. Hupont, S. Ballano, S. Baldassarri, and E. Cerezo, "Scalable multimodal fusion for continuous affect sensing," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI), Workshop Affect. Comput. Intell. (WACI)*, Apr. 2011, pp. 1–8.
[40] S. Poria, E. Cambria, and A. Gelbukh, "Deep convolutional neural network textual features and multiple kernel learning for utterance-level multimodal sentiment analysis," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 2539–2544.
[41] M. Mansoorizadeh and N. M. Charkari, "Multimodal information fusion application to human emotion recognition from face and speech," *Multimed. Tools Appl.*, vol. 49, no. 2, pp. 277–297, Aug. 2010.
[42] S. Poria, E. Cambria, N. Howard, G.-Bin Huang, and A. Hussain, "Fusing audio, visual and textual clues for sentiment analysis from multimodal content," *Neurocomputing*, vol. 174, pp. 50–59, Jan. 2016.
[43] M. Wollmer *et al.*, "YouTube movie reviews: Sentiment analysis in an audio-visual context," *IEEE Intell. Syst.*, vol. 28, no. 3, pp. 46–53, May 2013.
[44] P. K. Atrey, M. A. Hossain, A. El Saddik, and M. S. Kankanhalli, "Multimodal fusion for multimedia analysis: A survey," *Multimedia Syst.*, vol. 16, no. 6, pp. 345–379, 2010.
[45] *Software Engineering—Product Quality—Part 1: Quality Model*, Standard ISO/IEC 9126:2001, ISO, 2001.
[46] R. G. Dromey, "A model for software product quality," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 146–162, Feb. 1995.
[47] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, and H. Lipson, "Attribute-based architecture styles," in *Proc. Work. Conf. Softw. Archit.*, 1999, pp. 225–243.
[48] F. Losavio, L. Chirinos, and M. A. Perez, "Quality models to design software architectures," in *Proc. Technol. Object-Oriented Lang. Syst. (TOOLS)*, 2001, pp. 123–135.
[49] B. Meyer, *Object-Oriented Software Construction*. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.
[50] Y. Press and L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 1979.
[51] A. Shaik, B. Manda, C. Prakashini, C. R. K. Reddy, and K. Deepthi, "Metrics for object oriented design software systems: A survey," *J. Emerg. Trends Eng. Appl. Sci.*, vol. 1, no. 2, pp. 190–198, 2010.
[52] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *Proc. Softw. Eng. Conf.*, 2004, pp. 309–318.
[53] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for architecture evaluation," Softw. Eng. Inst., Carnegie-Mellon Univ, Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2000-TR-004, 2000.
[54] A. Landowska, "Emotion monitor-concept, construction and lessons learned," in *Proc. Federated Conf. Comput. Sci. Inf. Syst. (FedCSIS)*, Oct. 2015, pp. 75–80.
[55] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, "Goal question metric (GQM) approach," in *Encyclopedia of Software Engineering*. Hoboken, NJ, USA: Wiley, 2002.
[56] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.
[57] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *J. Syst. Softw.*, vol. 85, no. 2, pp. 244–257, Feb. 2012.
[58] T. L. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2010, pp. 1–10.
[59] P. Oliveira, M. T. Valente, and F. P. Lima, "Extracting relative thresholds for source code metrics," in *Proc. IEEE Conf. Softw. Maintenance, Reeng., Reverse Eng. Softw. Evol. Week (CSMR-WCRE)*, Feb. 2014, pp. 254–263.
[60] S. Herbold, J. Grabowski, and S. Waack, "Calculation and optimization of thresholds for sets of software metrics," *Empir. Softw. Eng.*, vol. 16, no. 6, pp. 812–841, Dec. 2011.
[61] A. Landowska, "Web questionnaire as construction method of affect-annotated lexicon—Risks reduction strategy," in *Proc. Int. Conf. Affect. Comput. Intell. Interact. (ACII)*, Sep. 2015, pp. 421–427.
[62] A. Kolakowska, A. Landowska, M. Szwoch, W. Szwoch, and M. R. Wrobel, "Evaluation criteria for affect-annotated databases," in *Beyond Databases, Architectures and Structures* (Communications in Computer and Information Science). Cham, Switzerland: Springer, 2015.
[63] M. Bradley and P. Lang, "Affective norms for English words (ANEW): Instruction manual and affective ratings," Center Res. Psychophysiol., Univ. Florida, Gainesville, FL, USA, Tech. Rep. C-1, 1999.
[64] G. Brodny and A. Landowska, "Integration in multichannel emotion recognition," in *Proc. 11th Int. Conf. Hum. Syst. Interact. (HSI)*, 2018, pp. 35–41.

AGNIESZKA LANDOWSKA was born in Gdańsk, Poland, in 1976. She received the M.S. and engineering degrees in computer science from the Gdańsk University of Technology, Poland, in 2001, and the Ph.D. degree in computer science from the Gdańsk University of Technology, in 2006.

Since 2000, she has been with the Faculty of Electronics, Telecommunications and Informatics, Department of Software Engineering, Gdańsk University of Technology. She is currently the Leader of Emotions in HCI Research Group, where she conducts research on software usability, affective computing methods, and e-learning systems designs. Recently, she managed the research project on methods and tools for affect-aware intelligent tutoring systems financed by Polish-Norwegian Financial Mechanism. She is the leader of the project developing mobile therapeutic applications for children with autism spectrum disorder.

Dr. Landowska is a member of the Management Board of the Polish Scientific Society on E-Learning (PTNEI) and a member of the scientific organizations AAAC and EDEN. She was a recipient of a Gold Medal in KIDE International Design Fairs, Taiwan, in 2016, and a Bronze Medal in IENA International Innovations Fairs, Nuremberg, Germany, in 2016, for a method of automated therapy monitoring in children with autism.

GRZEGORZ BRODNY was born in Gdynia, Poland, in 1991. He received the B.S. and M.S. degrees in computer science, with specialization in software engineering and databases, from the Gdańsk University of Technology, in 2015, where he is currently pursuing the Ph.D. degree in computer science.

His Ph.D. thesis concerns the uncertainty and integration of emotional states in affect-aware software. He took part in the methods and tools for affect-aware intelligent tutoring systems project. Since 2015, he has been a member of Emotions in HCI Research Group, Gdańsk University of Technology. His scientific studies include affective computing, software integration, late and early fusion in classification problems, and the reliability of measurements of affect symptoms.

MOST WIEDZY Downloaded from mostwiedzy.pl