A Universal Criteria Catalog for Evaluation of Heterogeneous Agent Development Artifacts

Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
{braubach | pokahr | lamersdorf}@informatik.uni-hamburg.de

Abstract. The research discipline of multi-agent systems is characterized by a high degree of heterogeneity. This heterogeneity leads to a vast amount of options (e.g. different architectures and languages) how to employ agent technology but is also one major source of difficulties for its adoption. People interested in using multi-agent systems depend on solid survey articles, which clarify and evaluate these different options and explain in which situations which choices should be made. A survey should also propose viable classification means for helping readers to understand which development artifacts broadly exhibit similar properties. To date, in most cases multi-agent system surveys do without classifications and only address one specific type of artifact such as agent languages or tools. Often, only the characteristics of the representatives are described without evaluating them. In this work a universal criteria catalog will be presented that has been defined abstractly enough for being usable for a wide variety of agent development artifacts. It will be shown how this abstract catalog can be further refined with respect to the chosen area of investigation. In addition to the catalog its general usage as part of a survey will be explained and a blueprint for survey conduction will be presented. To demonstrate its usefulness cutouts of extensive evaluations, performed in the areas of agent architectures, languages, methodologies, tools and platforms, will be presented.

1 Introduction

The high heterogeneity of the multi-agent systems (MAS) research field leads to many options for realizing agent applications. As many of the available solutions (be it methodologies, platforms or other things) are suitable only in specific application contexts it is very important to have guidelines at hand for the selection of the right option with respect to the given problem [9]. One viable instrument consists in studying surveys and evaluations about specific agent artifacts such as agent architectures or languages. Regrettably, most existing surveys do not contain evaluations of the described artifacts and available comparisons of artifacts suffer from ad-hoc classifications resp. selections as well as from non-standardized evaluation criteria. Especially, divergent criteria make it hard to appraise and compare evaluation results, because it remains unclear if the considered criteria are relevant and if there are others not discussed at all.

To improve this situation in this paper a *universal criteria catalog* is presented that has been deduced from established standards and is sufficiently generic for being utilized for the evaluation of arbitrary agent artifacts. The usage of the catalog fosters several important aspects. Firstly, evaluations of the same artifact type become comparable making visible the advances in the multi-agent research field, e.g. platform surveys from nowadays and from 5 years ago could show in which areas (e.g. operating ability) progress has been achieved. Secondly, evaluations of different artifact types become comparable. This will allow identifying how the state-of-the-art with respect to different artifacts is related and e.g. in which area research should be urged. Thirdly, the criteria catalog has been conceived to be usable in different scenarios. This only requires conceiving a suitable weighting scheme, which emphasizes the different criteria according to their importance with respect to the overall evaluation objective.

Besides the criteria catalog itself it is also sketched what else needs to be done to obtain significant evaluation results. Therefore, a survey blueprint is presented highlighting all important aspects that a survey (including an evaluation part) should possess and also in which order they should roughly be performed.

The rest of this paper is structured as follows. Section 2 introduces the universal criteria catalog. An abstract evaluation process is described in section 3. To illustrate the catalog usage, excerpts of evaluations, performed in the area of agent architectures and programming languages, are presented in sections 4 and 5, respectively. Section 6 concludes the paper with a summary and an outlook.

2 Criteria Catalog

In this section it is described what comprises the proposed universal criteria catalog, how it was deduced from existing standards and how it relates to other work in the field of agent surveys and evaluations.

In general there are two opposing requirements for the criteria catalog. On the one hand, the catalog should be universal and apply to all possible kinds of agent development artifacts. On the other hand, the catalog should be highly concrete to allow meaningful results, when evaluating specific types of artifacts for specific settings. The first requirement, i.e. the universality of the catalog, is essential for achieving comparability of evaluation results by providing a stable set of criteria, which can also be used in future investigations. Moreover, applying a single universal set of criteria to many different types of artifacts is the best way to ensure the completeness of the criteria catalog.

On the other hand, having concrete criteria, as mandated by the second requirement, is the precondition for obtaining profound and accurate evaluation results. Detailed and clearly defined criteria contribute to the objectiveness of an evaluation, as they prevent unconscious or ad-hoc assessments. A concrete and detailed criteria catalog also allows for adapting the evaluation to a specific setting, by weighting different criteria according to their need. When e.g. an agent platform is searched as a teaching vehicle, one would apply different ratings to criteria as one would do in an industrial software project.

2.1 Foundations

Our work is based on existing standards such as ISO 9126 and ISO 9241 [16, 17] and on evaluations of agent-oriented artifacts such as development methodologies or programming environments [9, 11, 14, 30]. The ISO 9126-1 standard defines six general criteria for evaluating software product quality: *functionality*, *reliability*, *usability*, *efficiency*, *maintainability*, *portability*. These criteria (called characteristics) are subdivided into more specific subcharacteristics (e.g. *fault-tolerance* as part of *reliability*). The standard motivates, that subcharacteristics are further refined by attributes, which can be verified or measured. Those attributes are specific to concrete types of software and not defined in the standard. Moreover, the set of ISO 9241 specifications describes guidelines for the ergonomics of human system interaction that can e.g. be used for the further refinement of certain (sub)characteristics.

Besides existing standards, another source for relevant evaluation criteria are the existing surveys in the area of agent-oriented development artifacts. Unfortunately, most of these surveys focus on one specific type of artifact and therefore do not consider the universality of the applied criteria. Nevertheless, the surveys are helpful for identifying criteria that are considered relevant by agent developers. E.g. Sturm and Shehory [30] propose the criteria categories *concepts and properties*, *notations and modeling techniques*, *process*, and *pragmatics*, refined into numerous subcriteria, for evaluating agent-oriented methodologies. For evaluating agent development software, such as agent platforms or development environments, Eiter and Mascardi [14] introduce the categories *agent attitudes*, *software engineering support*, *agent and MAS implementation*, *technical issues*, and *economical aspects*, together with specific characteristics for each category.

2.2 Criteria Catalog Proposal

To meet the requirements of a universal yet concrete criteria catalog, we adopt the ISO approach of broad criteria categories, refined into more detailed, but still universal subcriteria. Basically, the catalog proposal distinguishes between *functional* and *non-functional* requirements. Functional requirements are directly related to the scope of operation, whereas non-functional requirements consider the quality of a given artifact and its provided functions. In the criteria catalog non-functional requirements have been further decomposed into the categories *usability*, *operating ability* and *pragmatics*, thereby subsuming and generalizing the six main criteria of ISO 9126-1. Similarly, the subcriteria are obtained from a unification and generalization of criteria found in standards and existing surveys. In the following the top-level and subcriteria of the catalog will be presented:

Function: The function encompasses all functional properties of an artifact. Which concrete functional requirements are important highly depends on the kind of researched artifact, e.g. in the context of programming languages other aspects are considered than in the context of architectures. The detailed criteria can be deduced asking the following typical questions:

- Which are the base concepts of the artifact?
- What does the artifact enable and which restrictions exist (power, missing concepts, capabilities)?
- Is the function of the artifact adequate in the given context (for which contexts was it conceived/is it usable)?
- Is the function of the artifact adequate with respect to the developer's knowledge and capabilities (suitable for beginners/experts only)?

Usability: The usability of an artifact refers to its suitability for the construction of agent applications. The degree of usability is evaluated according to the following aspects:

- **Simplicity/intuitivity:** How simple are the underlying artifact's mechanisms (simplicity vs. power)? Is the artifact intuitive, i.e. can the developer use the artifact in an understandable and anticipatable way?
- **Learnability/familarity:** What learning curve has the artifact? Is the developer already familiar with the artifact or its underlying mechanisms from another context?
- **Individualization:** Can the artifact be tailored towards the user and/or the context?
- **Extensibility:** Can the artifact be extended with new functionalities?
- **Software engineering principles:** Does the artifact respect well-known principles such as modularization, refactoring, verification and/or does it facilitate the reusability of elements in other contexts?

Operating ability: The operating ability of an artifact encompasses all aspects that are relevant while the artifact is executed (used), i.e. which properties does the artifact exhibit during its operation? The operating ability's quality is measured using the following aspects:

- **Performance:** How efficient is the artifact with respect to space- and/or time-critical operations?
- **Robustness:** How tolerant is the artifact with respect to (partial) breakdowns?
- **Stability:** How does the artifact behave if executed during a longer time period?
- **Scalability:** How does the artifact behave when applied to varying problem sizes?

Pragmatics: Pragmatic aspects refer to external factors that are neither related to the construction nor to the operation of the artifact. Nonetheless, pragmatic aspects can exert an important influence on the evaluation of the artifact under consideration (e.g. a software), as they determine to a high degree if an artifact can be used in practice. In detail, pragmatic aspects are subdivided into the following criteria:

- **Installation/adoption:** How easily can the artifact be installed?
- **Documentation/examples/support:** Is documentation, example code and support available for the artifact? What quality do they have?
- **Popularity:** How big is the user community? Are field reports available?
- **Maturity:** How mature is the artifact conceptually as well as technologically?
- **Technical boundary conditions:** How easily can the artifact be embedded into existing IT landscapes? How well does the artifact fit to the technological mainstream?
- **Costs:** Which costs are related with the artifact (purchase, construction, operation, working time, training, etc.)?

To conclude, this section has presented a universal criteria catalog, which is based on established ISO-standards and consists of the four main categories: *function, usability, operating ability* and *pragmatics.* It is general enough to be used for the evaluation of all important agent development artifacts and provides enough detailed requirements for producing meaningful and comparable evaluations as explained in the next sections.

3 Usage of the Catalog

The criteria catalog has been used as a foundation for extensive research in the field of multi-agent systems and was specifically used by the authors to describe and evaluate agent architectures, languages, methodologies, platforms and tools [8, 21]. In this section it will be shown how the catalog can be embedded as part of a survey with respect to a specific development artifact. Therefore, a blueprint of such a survey will be sketched and later on example surveys in the selected areas of agent architectures and languages will be presented. Conducting a survey naturally requires incorporating related work as much as possible with respect to other surveys as a whole and also with respect to all important parts of a survey as explained next.

The survey blueprint we propose consists of the following logical steps: *artifact definition*, *classification*, *selection*, *catalog refinement*, *evaluation* and *summarizing results*. Starting point of each survey should be a discussion of the meaning of the selected artifact. In this respect the important definitions from literature should be discussed and it should be made clear (if the notion is not unambiguously defined) why a certain definition forms the foundation of the further explorations. Thereafter, the state-of-the-art should be described. For this purpose, in a first step classifications have to be taken from literature or newly conceived and evaluated with respect to the research objective.

A good classification serves two purposes: firstly, it should help identifying structures in the research field giving the unfamiliar reader an initial orientation and secondly it should facilitate the selection of specific representatives for a detailed evaluation as representatives within the same category have similar properties. Given that a lot of individual representatives are usually available a detailed discussion of all of them is often neither feasible nor desired and it is sufficient to select prototypical representatives from each important category. As last preparing step for the description of the state-of-the-art it is necessary to perform the already mentioned catalog refinement with respect to the given research artifact. In many cases it should be sufficient to elaborate the meaning of the functional criteria of the catalog as the non-functional aspects are quite stable and rather independent of the artifact type. Thereafter, the selected individual representatives should be explained and evaluated with respect to the criteria catalog. Finally, the individual results should be condensed by abstracting away from details and calculating key data. The formula for aggregating the results can vary according to the research objective and should be defined in beforehand of the evaluation (e.g. as part of the catalog refinement step).

The proposed survey blueprint can be tailored to a concrete evaluation scenario by introducing specific evaluation scales. In the following, as an example a generic scheme will be explained that is easy to apply and additionally fosters objective evaluation results. The proposed scheme exhibits a neutral weighting with regard to the criteria, and intends that the evaluation will be done in three steps. Firstly, each individual subcharacteristic of the criteria catalog (such as simplicity/intuitivity) should be rated on a coarse scale (+1/-1/0 standing for yes/no/undefined resp. good/bad/neutral). Such a coarse evaluation scale (e.g.

instead of values from 0 to 10) speeds up the evaluation process and also improves objectiveness, because the decision if a criterion is fulfilled or not is less prone to be influenced by personal taste. To add further details, each decision should be justified by a meaningful textual description of the relevant reasons, which allows readers of the evaluation to follow or scrutinize the decisions.

In a second step, the rating of a main characteristic (such as usability) can be calculated as a sum of the individual results. As the number of subcriteria differs for the main characteristics, these summary values should be normalized to a common scale (e.g. from -2 to +2 meaning from very weak to very strong). As final step, the overall evaluation result can be determined as mean of the main characteristics, assuming for a generic evaluation, that all main characteristics are equally important.

4 Architecture Survey

In the following a cutout of the agent architecture evaluation from [8] will be presented. The presentation here will highlight how the proposed blueprint in general and criteria catalog in particular can be used for conducting a survey.

Even though there are several surveys that target agent architectures such as [27, 33] nearly all of them focus on a detailed description and possibly classification of the individual representatives, whereas none focuses on an exhaustive evaluation and comparison of the representatives (in [27] at least some selected properties are compared). One reason for this might be that architectures are abstract and in connection with the development of MAS it seems more obvious to evaluate agent platforms, because these artifacts will directly be used. However, agent platforms employ agent architectures and therefore inherit a good deal of their properties. This means that architecture evaluations can e.g. help to identify the underlying conceptual limitations of agent platforms.

4.1 Architecture Survey: Artifact Definition

Bass et al. [2] define: "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." It therefore has the basic task of making structures of elements and their relationships visible.

In the area of multi-agent systems it is broadly distinguished between *internal agent architectures* and *multi-agent (or social) architectures*. An internal agent architecture tackles the question of what comprises an agent, i.e. which building blocks can be used for its construction and how these concepts are related. On the other hand, a social agent architecture tries to describe how coordination between agents, e.g. in the sense of teamwork, can be conceptualized.

4.2 Architecture Survey: Classification

Different kinds of classifications have been proposed for internal architectures. The most influential scheme is the one of Wooldridge and Jennings [33], who introduced a distinction between *reactive*, *deliberative* and *hybrid* architectures.

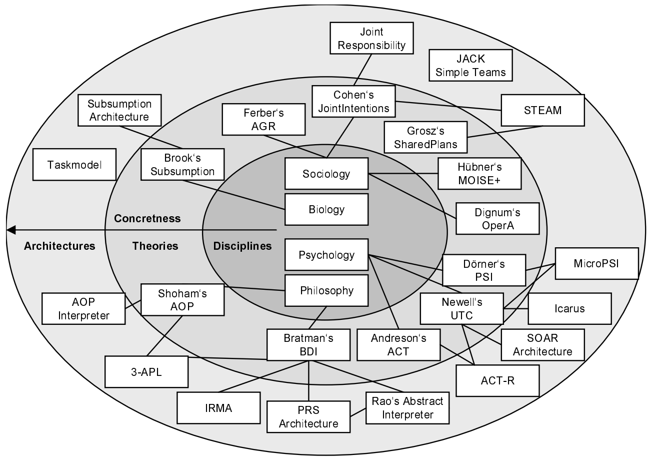


Fig. 1. Architecture overview

In our work [8] we preferred a more general classification scheme that relates architectures to the discipline and theory from which they originated (see Fig. 1). In general, architectures have not been invented per se but rely on a more abstract description in form of an agent theory. This means the scheme basically distinguishes four different categories according to the disciplines philosophy, psychology, biology and sociology from which they have been influenced. These base categories are further refined towards the relatively abstract descriptions in the form of agent theories such as the Unified Theories of Cognition (UTC) [20] and the Belief-Desire-Intention Model (BDI) [7]. Most architectures have been conceived as an interpretation and concretization of such a theory.

4.3 Architecture Survey: Selection

The selection of representatives then excluded social architectures and also those, which are not intended as foundation for application construction (e.g. psychologically inspired architectures such as ACT-R that are used for experimentation only). Concretely, the Subsumption [10], BDI (here specifically PRS) [23], Soar [18], AOP [29], 3APL [12] and Taskmodel (e.g. [13]) architectures have been evaluated in the context of the criteria catalog.

4.4 Architecture Survey: Catalog Refinement

The catalog refinement with respect to internal agent architectures needs to define what functionality is required from an agent. Therefore, it is necessary to agree upon a generally accepted agent definition. We used the *strong notion of agency* as defined in [33] for that purpose. It stresses the following agent properties: *autonomy, reactivity, proactivity, social abilities* and *mentalistic notions*. Other abilities that go beyond this definition (such as learning) are not considered as required in our evaluation and can only influence it positively.

4.5 Architecture Survey: Evaluation

The descriptions of the individual architectures and their evaluation with regard to the criteria catalog cannot be presented in full length here due to the space limitations (see [8] for the whole survey). To get an impression of how the catalog is used for the evaluation of an artifact instance the detailed evaluation of one representative with respect to one main characteristic will be further illustrated. For this purpose the *usability* of the Soar architecture will be discussed:

Simplicity/intuitivity (=): The basic principles of the Soar architecture are simple and have been formulated in terms of a few hypotheses (esp. the physical symbol system and the problem space hypothesis). Foundation of the behavior control is the search within problem spaces, which relies on the intuitive concepts of beliefs and goals. The intuitivity is yet reduced by the fact that only one solution context can be active at the same time which is not in line with the parallel goal achievement behavior of humans.

Learnability/familarity (-): The learnability of the architecture is rather low because the architecture concepts cannot directly be used for the realization of agents, i.e. instead of goals the developer has to deal with different kinds of low-level production rules. Developers with a solid background on rule-based approaches will have advantages in learning Soar.

Individualization (-): The architecture does not allow being tailored neither towards the user nor to the context.

Extensibility (-): Extensibility has not been integrated at the architectural level of Soar. Therefore, extensions have to be built at the application level, which e.g. has been successfully done for a teamwork approach in [31]. Extensions that aim at enhancing the available base mechanisms such as the knowledge representation or learning method are far more difficult to realize and require a deep understanding of the architecture.

Software engineering principles (=): Soar supports the modular software development by separation of the overall problem into individual problem spaces. This allows for constructing the different functionalities of an application rather independently of each other. Nevertheless, it does not provide true reusability of software artifacts among different application contexts as tight coupling between functionalities exists due to direct connections of problem spaces via common beliefs.

The rating for the usability of the Soar architecture is therefore calculated as weak (-)[1] and underlines that improvements in this direction could further enhance the acceptance of the approach.

4.6 Architecture Survey: Summarizing Results

The result table is depicted in Fig. 3(A) and shows the evaluations of the four main characteristics for all analyzed architectures as well as their overall results. It reveals that none of the architectures was able to achieve very good valuations in all of the criteria. With respect to the different criteria it is conspicuous that the *operating ability* of all architectures is high or even very high meaning that all architectures can be implemented efficiently. In contrast, the other criteria have produced mixed results. In most architectures the *function* can be further improved and exhibits weaknesses with respect to the required properties of the

[1] Calculated as normalized sum: (0+(-1)+(-1)+(-1)+0)/5*2=-1.2

strong notion of agency. The same applies for the *usability* which is not as good as it is often claimed for the agent paradigm, i.e. even though the paradigm provides intuitive metaphors, it does not automatically lead to understandable and software-technically sound architectures. Pragmatic aspects mainly depend on the number and quality of available software implementations, which is best for widely used approaches such as the Taskmodel, BDI and Soar.

Considering the overall results, BDI, Soar and Taskmodel architectures have attained good evaluations, whereby the BDI architecture is the only architecture without obvious weaknesses. Nonetheless, all architectures can be improved especially with respect to the function and usability.

5 Language Survey

This section recapitulates an evaluation of agent-oriented programming languages performed in [21]. Existing surveys of agent-oriented programming languages such as [4, 5, 33] have been taken into account primarily for reviewing existing categorizations, e.g. with regard to the language type (e.g. logic-based vs. procedural) [4, 5] or aspects of the agent architecture (e.g. deductive, practical, reactive or hybrid reasoning) [33]. This review made sure that the scope of the planned survey would be broad enough to incorporate all relevant work in the area. Moreover specific evaluations, e.g. of logic based languages [19], were investigated as part of the process of finding suitable refinements of the criteria catalog (i.e. the refinement should make sure that aspects of existing evaluations are considered).

5.1 Language Survey: Artifact Definition

In general, a programming language is commonly agreed to be an artificial language that can be used to instruct machines. E.g. the ACM SIGPLAN group [1] defines programming languages as "[...] languages that permit the specification of a variety of different computations, thereby providing the user with significant control (immediate or delayed) over the computer's operation." In the area of agent-based systems, programming languages are used for a number of different purposes. We adopt the scheme proposed by Ferber [15], which introduces subgroups of agent-oriented languages, such as communication languages and knowledge representation languages. In our evaluation, we focused on languages for describing the behavior of agent systems.

5.2 Language Survey: Classification

The classification of agent languages is straightforward, as it is based on the previously presented classification of agent architectures with regard to their origin. E.g. the PRS language category is introduced for languages implementing a PRS-like architecture as originating from the philosophical BDI model. It is noteworthy that the language type "API-based approach" has been identified as the common way to implement the Taskmodel architecture, where "API-based" means that instead of conceiving a new language only a programming library for an existing language (e.g. Java) is provided.

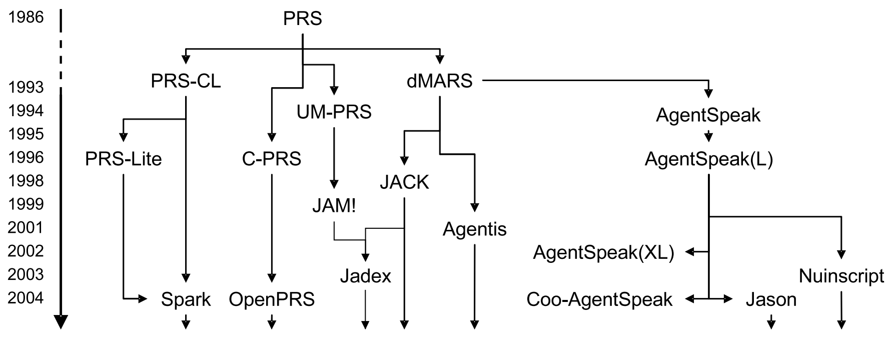


Fig. 2. PRS-like systems and relations

5.3 Language Survey: Selection

During the selection process, social languages have been excluded, as the focus of our investigation was on programming constructs for individual agents. Moreover, the Subsumption architecture category has no correspondence in the language survey. From each of the remaining five categories – AOP, 3APL, PRS (=BDI), Soar, API-based (=Taskmodel) – the most prominent representatives, based on academic as well as industrial recognition, have been selected for further investigation. For some categories such as PRS languages, where a large number of representatives exists, a pre-analysis of the field has been performed to identify existing relationships (cf. Fig. 2).

Investigated PRS languages are AgentSpeak(L) as implemented in the academic Jason interpreter [6] and the languages of the commercial JACK agent toolkit (JAL) [32] and the academic Jadex framework [22]. In the area of AOP languages, the original work on Agent-0 [29] has been considered, as well as the languages employed in the commercial AgentBuilder (RADL) [24] and the academic Agent Factory toolkits (AF-APL) [26]. The JADE platform [3] and the Living Systems Technology Suite (LS/TS) from Whitestein Technologies [25] have been selected as representatives of API-based approaches. For 3APL and Soar, the most recent incarnations (at that time), as described in [12] and [18], have been evaluated.

5.4 Language Survey: Catalog Refinement

The refinement of the criteria catalog introduces relevant areas of functionality for programming languages. The functionality of a programming language is represented by its programming constructs. Based on general programming language literature (e.g. [28]) and existing evaluations of agent-oriented programming languages (e.g. [19]) three basic functional criteria have been identified: *concept abstraction*, *control flow abstraction*, and *safety*. These criteria are briefly explained in the following. For a more detailed explanation see [21].

Concept Abstraction: Concept abstraction means the introduction of programming constructs for some kind of high-level concept (e.g. abstract data types, procedures or – in the agent world – goals or commitments). Concept abstraction is subdivided into *data abstraction*, *behavior abstraction*, and *func-*

tionality abstraction, which respectively denote the availability of programming language constructs for representing data (e.g. objects), behavior (e.g. activities), and functionality (e.g. modules).

Control Flow Abstraction: One main requirement for an agent-oriented programming language is to provide constructs for specifying independent or interrelated sequences of activities, i.e. control flow abstraction. Subcriteria for control flow abstraction are *concurrency*, *dynamic behavior*, and *dynamic execution*. Concurrency constructs (e.g. threads and semaphores) allow to specify (quasi-) parallel activities and synchronization points between them. Dynamic behavior means the ability of an agent to dynamically select among different actions when requested to perform some task. Finally, with concepts for dynamic execution (e.g. event or exception handlers), behaviors can be activated in a dynamic (runtime-dependent) way.

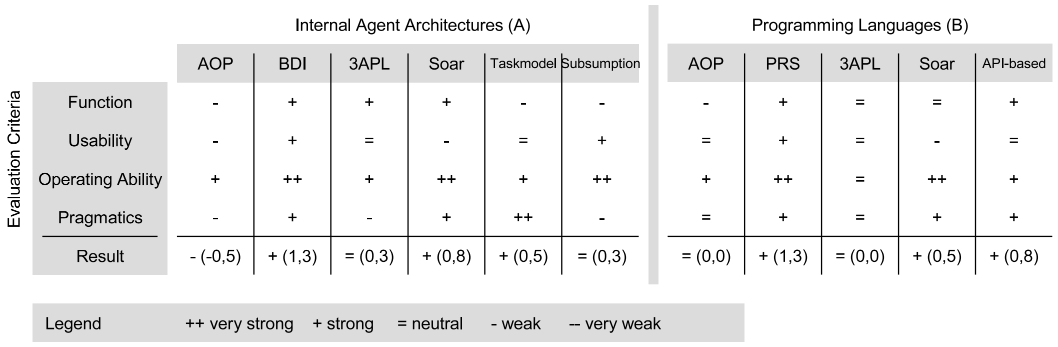
Safety: Safety subsumes built-in functionality of a programming language aimed at reducing the number of programming errors and can be subdivided into *error prevention* and *error recognition* functionality. Error prevention reduces possible sources for errors by design (e.g. local name spaces prevent name conflicts, automatic garbage collection reduces memory management errors). Error recognition functionality, on the other hand, aims at detecting errors early in the development process. E.g. with static typing, many errors can be detected at compile time, which would otherwise occur at runtime, and asserts allow to easily find errors at runtime, which might otherwise stay undetected while leading to incorrect results.

5.5 Language Survey: Evaluation

The evaluation process for languages follows the generic evaluation scheme already applied to the architectures, i.e. coming to yes / no decisions about individual subcriteria, which are then summarized to produce the overall score. The results for each of the subcriteria have been obtained in analytical and empirical evaluations. To evaluate criteria such as simplicity/intuitivity, example applications have been implemented with all of the languages. Among these applications, some simple benchmarks, initially implemented for evaluating platform performance (e.g. memory consumption and time needed for creating/destroying a given number of agents) were also useful for evaluating the language constructs available for these common tasks. As an example, the evaluation of the functionality of the JACK agent language (JAL) is presented next:

Concept Abstraction (+): JAL is based on Java and therefore allows data to be represented in an object-oriented fashion. A slight drawback is the fact, that the internal reasoning of JACK requires some data to be represented in a simple relational model, which does not fit well with the object-oriented abstraction. Behavior abstraction is nicely supported by the notion of plans, as well as functionality abstraction is provided by the so called capability concept.

Control Flow Abstraction (=): The control flow of a BDI agent is driven by the practical reasoning process consisting of the steps goal deliberation and means-end reasoning. The PRS approach, as implemented in JACK, focuses on means-end reasoning, i.e. selecting plans for achieving goals or reacting to events.



	Internal Agent Architectures (A)						Programming Languages (B)				
Evaluation Criteria	AOP	BDI	3APL	Soar	Taskmodel	Subsumption	AOP	PRS	3APL	Soar	API-based
Function	-	+	+	+	-	-	-	+	=	=	+
Usability	-	+	=	-	=	+	=	+	=	-	=
Operating Ability	+	++	+	++	+	++	+	++	=	++	+
Pragmatics	-	+	-	+	++	-	=	+	=	+	+
Result	- (-0,5)	+ (1,3)	= (0,3)	+ (0,8)	+ (0,5)	= (0,3)	= (0,0)	+ (1,3)	= (0,0)	+ (0,5)	+ (0,8)

Legend	++ very strong	+ strong	= neutral	- weak	-- very weak

Fig. 3. Evaluation results for architectures and languages

While the language therefore provides very good control flow abstraction in this area, the first step of practical reasoning, i.e. the selection of goals to pursue, is not directly supported. This part of the agent behavior therefore needs to be implemented in a manual, ad-hoc fashion by the agent developer.

Safety (+): The new constructs that JAL introduces as extensions to the Java language have been designed to allow for compile-time consistency checks and therefore support the safety of the language. Moreover, Java constructs for runtime checks, such as asserts, can be used in JAL as well.

As a result, JAL is attested a good (+) functionality score,[2] while identifying also some areas for improvement.

5.6 Language Survey: Summarizing Results

One aim of the survey was comparing programming approaches abstracting away from concrete language implementations. Therefore, the results of the individual representatives, such as AgentSpeak or JAL, have been used to compose a unified result for each approach. To avoid that deficiencies of a single representative weaken the score of an approach as a whole, for each main criteria such as function or usability, the best value has been selected instead of the average.

The accumulated scores of the evaluation are shown in Fig. 3(B). PRS languages are rated best, followed by API-based approaches and Soar. This result might also be influenced by the fact that these are the language families, where ongoing commercial development takes place. Another indication for the importance of commercial development is that the winners achieve their best scores in the area of operating ability. Nevertheless, it can be seen that still many areas for improvement exist, e.g. the continued integration of agent-oriented concepts into useful programming language constructs to further improve the functionality of agent-oriented programming languages.

6 Conclusion

This paper has tackled the question how a developer can choose among the many development options when implementing an agent application. One key

[2] Calculated as normalized sum: (1+0+1)/3*2=+1.3

aspect here is to understand that agent technology currently offers many problem-specific solutions that address only certain types of application domains. We argue that one important foundation for making accurate choices is the availability of well-defined and comparable surveys and evaluations of artifacts such as platforms or methodologies. Therefore, we have proposed a new universal criteria catalog for evaluating many different kinds of agent artifacts. This is possible because the criteria catalog is two-staged, consisting of an abstract artifact-agnostic stage and additionally an artifact-specific stage, which needs to be refined with respect to the concrete artifact type. The general applicability of the criteria catalog has been proven by cutouts of two extensive evaluations in the area of agent architectures and programming languages, but was also successfully employed for methodologies, platforms and tools [8, 21].

Besides the criteria catalog it has also been shown how it can be used as one part in a complete survey. For this purpose the integral ingredients of a survey have been identified and their coarse ordering has been defined. In a first step the *definition of the artifact* has to be explained for establishing a common discussion basis. Thereafter an overview of the available representatives should be given and a *selection* of artifact instances should be performed by applying a meaningful *classification*. The main part of the survey should then present the selected representatives and *evaluate* them against the in beforehand *refined criteria catalog*. Finally, the detailed results should be *summarized* and coarsened to deduce also globally valid statements.

In future work we want to employ the criteria catalog to perform individualized surveys that try to reason about the degree of usefulness of artifact instances with respect to specific application scenarios (e.g. which agent platforms are specifically suited for the transportation domain and why). Additionally, we plan to investigate the applicability of the presented criteria catalog regarding other non agent-related artifact types (such as component-based approaches). Finally, a main objective and hope of the paper is that other researchers pick-up the universal criteria catalog for their planned investigations. This would lead to many positive effects especially with respect to the completeness of the used criteria and the comparability with other evaluations.

References

1. ACM SIGPLAN. Bylaws of the special interest group on programming languages of the association for computing machinery. ACM, 2003.
2. L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley, 2005.
3. F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE - A Java Agent Development Framework. [5], pages 125–147.
4. R. Bordini, L. Braubach, M. Dastani, A. El Fallah Seghrouchni, J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30:33–44, 2006.
5. R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, 2005.
6. R. Bordini, J. F. Hübner, and R. Vieira. Jason and the Golden Fleece of Agent-Oriented Programming. [5], pages 3–37.

7. M. Bratman. *Intention, Plans, and Practical Reason*. Harvard Univ. Press, 1987.
8. L. Braubach. *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme*. PhD thesis, Univ. Hamburg, 2007.
9. L. Braubach, A. Pokahr, and W. Lamersdorf. Tools and Standards. In S. Kirn, O. Herzog, P. Lockemann, and O. Spaniol, editors, *Multiagent Systems. Intelligent Applications and Flexible Solutions*, pages 503–530. Springer, 2006.
10. R. A. Brooks. How to build complete creatures rather than isolated cognitive simulators. In *Architectures for Intelligence*. Lawrence Erlbaum Associates, 1989.
11. M. Casagni and M. Lyell. Comparison of two component frameworks. In *Proc. of ICSE'03*, pages 341–351. IEEE Computer Society, 2003.
12. M. Dastani, B. van Riemsdijk, and J. J. Meyer. Programming Multi-Agent Systems in 3APL. [5], pages 39–67.
13. S. A. DeLoach. Specifying agent behavior as concurrent tasks. In *AGENTS'01*. ACM Press, 2001.
14. T. Eiter and V. Mascardi. Comparing environments for developing software agents. *The European Journal on Artificial Intelligence*, pages 169–197, 2002.
15. J. Ferber. *Multi-Agents Systems - An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.
16. International Organization for Standadization (ISO). *Software engineering – Product quality – Part 1: Quality model*, ISO/IEC 9126-1:2001 edition, 2001.
17. International Organization for Standadization (ISO). *Ergonomics of Human-System Interaction-Part 110: Dialogue Principles*, ISO 9241-110:2006 edition, 2006.
18. J. F. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to Soar, an architecture for human cognition. Technical report, University of Michigan, 2006.
19. V. Mascardi, M. Martelli, and L. Sterling. Logic-based specification languages for intelligent software agents. *CoRR*, cs.AI/0311024, 2003.
20. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
21. A. Pokahr. *Programmiersprachen und Werkzeuge zur Entwicklung verteilter agentenorientierter Softwaresysteme*. PhD thesis, Univ. Hamburg, 2007.
22. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. [5].
23. A. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proc. of ICMAS'95*, pages 312–319. MIT Press, 1995.
24. Reticular Systems. *AgentBuilder User's Guide*, version 1.3 edition, 2000.
25. G. Rimassa, D. Greenwood, and M. E. Kernland. The Living Systems Technology Suite. In *In Proc. ICAS'06*, 2006.
26. R. Ross, R. Collier, and G. O'Hare. AF-APL - Bridging Principles and Practice in Agent Oriented Languages. In *Proc. of ProMAS'04*, pages 66–88. Springer, 2005.
27. M. Scheutz and V. Andronache. The apoc framework for the comparison and evaluation of agent architectures. In *AAAI Workshop*. AAAI Press, 2004.
28. R. Sebesta. *Concepts of Programming Languages*. Addison Wesley, 2005.
29. Y. Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1), 1993.
30. A. Sturm and O. Shehory. A Framework for Evaluating Agent-Oriented Methodologies. In *Proc. of AOIS'03*, pages 94–109. Springer, 2004.
31. M. Tambe. Towards Flexible Teamwork. *Art. Intelligence Research*, 7, 1997.
32. M. Winikoff. JACK Intelligent Agents: An Industrial Strength Platform. [5].
33. M. Wooldridge and N. Jennings. Agent theories, architectures, and languages: A survey. In *Proc. of ATAL 1994*, pages 1–39. Springer Verlag, 1995.