I N F S Y S

R E S E A R C H

R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

COMPARING ENVIRONMENTS FOR DEVELOPING SOFTWARE AGENTS

Thomas Eiter Viviana Mascardi

INFSYS RESEARCH REPORT 1843-01-02

MARCH 2001

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-01-02, MARCH 2001

COMPARING ENVIRONMENTS FOR DEVELOPING SOFTWARE AGENTS

Thomas Eiter[1] Viviana Mascardi[2]

Abstract. In the last years, dozens of environments for modeling, testing and finally implementing multi-agent systems have been developed. Unfortunately, no standard criteria for understanding what kind of application profile a particular development environment is good for have been individuated yet, and the question "How should I choose an existing environment which best suits the features and requirements of my application?" is still difficult to answer. This paper addresses this question, and aims at helping the multi-agent system developer to solve this problem. It provides a set of criteria for evaluating development environments, and then applies these criteria to five selected tools an multi-agent systems prototypes. Furthermore, some application-driven guidelines are described to help identifying the features of a suitable environment for developing an implementation of the given application. The features we identify can be used to find the right development framework among the frameworks we evaluate for doing the right application.

Keywords: agent frameworks, software agent development, tools, multi-agent systems, classification

[1]Institut und Ludwig Wittgenstein Labor für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria. E-mail: eiter@kr.tuwien.ac.at

[2]DISI, Università di Genova, Via Dodecaneso 35, 16146, Genova, Italy. email: mascardi@disi.unige.it

Acknowledgements: This paper has been partly written when Viviana Mascardi was visiting TU Wien. This work was in part supported by the Austrian Science Fund (FWF) under grants P13871-INF and N Z29-INF.

Contents

1 Introduction

Today's software applications are mainly characterized by their component-based structure. Components are usually heterogeneous and distributed, which makes it quite difficult to coherently glue them together for developing a final efficient application.

One of the main reasons for the successful rise of the agent-oriented software paradigm is that it provides the right level of abstraction for engineering this kind of applications [18, 34, 52], representing a valuable help for handling their complexity and developing them.

Agents and multi-agent systems (MASs) are more and more successfully deployed and used in practice, even if many theoretical issues underlying them are not fully understood at this point and subject of current investigations. The lack of rigorous formal definitions and characterizations of agenthood, and the unclarity or even disagreement about what constitutes a MAS make the gap between theory and practice quite evident. More and more environments for developing agents and agent systems become available, each one adhering more or less faithfully to some existing definition of agent or adopting one ad-hoc.

Purist researchers may argue that only a very small number of existing general-purpose environments for modeling and developing MAS, which we refer to as *MAS development kits* (MASDKs) in this paper, can be used to build true *agents*. Most of these MASKDs associate with this term a poor meaning, based on the features of the kind of software whose development is supported, rather than on some generally agreed theoretical definition of agent (which, arguably, is difficult to find).

We know and certainly agree that terminology coming from the agent field is sometimes misused or even abused, which is is one of the reasons why agent research is less received in certain communities. Nonetheless, we believe that a large share of the existing MASDKs can provide a great support in building complex societies of heterogeneous components showing some (but not necessarily all) of the features usually ascribed to agents.

In this paper, we do not address the problem of what an agent is; many researchers have been working on finding a good answer to this question; see e.g. [28, 7, 33]. Also, we do not aim at discussing which available MASDKs support the development of *agents* which really deserve this name, and which ones just provide rudimentary support. Until the notion of agent will not have a clear and finally agreed definition, this kind of discussion will be impossible. We just observe that MASDKs exist, and that a MAS developer should take advantage of them for building a MAS as it seems fit. The developer, before asking

> "How should I **build** an ad-hoc MAS which models and realizes my application?"

should try to answer the question

> "Is there any development environment available which suits my application features and requirements? How can I **determine** it?"

Because of the evolution of the field and the continuing emergence of new agent environments and applications, no standard criteria for answering such a question do exist. Moreover, to our knowledge no attempts to evaluate and compare various existing MASDKs have been made so far. This, however, leaves us in a position that while numerous tools are available, providing a wealth of different features and capabilities, it is a nontrivial and, moreover, laborious task to single out some MASDK which fits best the need of a MAS application.

The aim and contribution of this paper is twofold: To provide

- some *guidelines for MASDK selection* performed on an application-driven basis, and

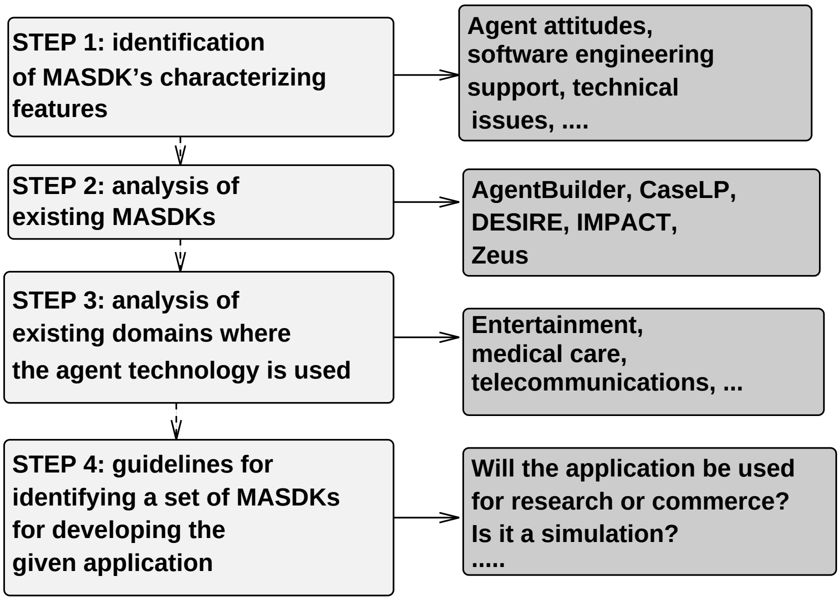

Figure 1: Our approach to develop MASDK selection guidelines.

- an *evaluation of five selected MASDKs*.

The last point practically exemplifies our approach to MASDK evaluation, and provides a MAS developer with support for understanding which MASDK (if any of them) can fit her/his needs among the evaluated ones. To our knowledge, this paper gives the first attempt to guide the MAS developer in the choice of an agent development tool, based on comparative evaluation. Other similar proposals [61, 40, 56] do not include the evaluation of existing tools.

Our approach to the development of the guidelines for MASDK selection proceeds in four steps, which are sketched in Figure 1:

Step 1 We compile a list of features that are useful for assessing MASDK properties; this is the subject of Section 2. This list is certainly not complete, as further criteria could be added. However, it takes into account many characteristics which are recognized to be relevant for defining agenthood, and includes a set of core criteria.

Step 2 According to the identified features, we analyze and compare five MASDKs: AgentBuilder, CaseLP, DESIRE, IMPACT and Zeus in Section 3. These platforms have different features and properties, and their heterogeneity is useful for testing our evaluation criteria.

Step 3 In Section 4, we analyze different scenarios where applying agent technology has already proved to be useful. For each of them, we highlight features which should be supported by a MASDK for being profitable in that scenario.

Step 4 Finally, in Section 5 we propose a set of selection guidelines based on the features of the MAS which should be developed. These guidelines aim at helping in the process of individuating a set of requirements which a MASDK should meet for being suitable to specific MAS development.

The steps 4 to 2 can be followed, in reversed order, as a *top-down approach* by the developer who wants to identify a set of MASDKs which are attractive for building his/her application. This is sketched on the following example, sketched in Figure 2, which is detailed as follows:

Step 4 The developer uses the guidelines to clarify the scenario to which the application to be developed belongs. This information will be used in Step 3. Other questions are addressed to help identifying another set of useful MASDK features which will be used in Step 2.

Step 3 Let us suppose the scenario to which the application belongs involves *information gathering*. The set of features which a MASDK for developing such an application should support are identified in this step. The "output" of Step 3 will be used in Step 2.

Step 2 Now the developer possesses a set of MASDK features known to be useful for the development of his/her application. This set derives partly from Step 4 and partly from Step 3. The developer can use these features as keys to choose a MASDK which meets his/her needs, according to the classification that we provide in Section 3. Usually, there will be more than one MASDK that approximates the ideal MASDK for the application to be developed. We do provide guidelines for restricting the set of MASDKs under consideration to a (small) set of candidates, which have almost equivalent properties; further discrimination and selection of a single candidate remains with the user.

However, we do not explicitly adopt the developer's perspective in the following, but develop the approach from our perspective of a *bottom-up approach*.

The remainder of this paper is structured as follows. After dealing with Steps 1 to 4 in Sections 2–5, respectively, we discuss in Section 6 some agent-related issues, and compare our work to related classification approaches in the literature. The final Section 7 concludes the paper and outlines directions for future work.

2 MASDK Features

Because of the different perspectives, the identification of a set of independent, orthogonal features which completely characterize a MASDK seems infeasible. In this section, we describe five groups of features which, as we believe, are significant enough to understand for what kind of applications a MASDK may be well-suited.

The aspects we consider are:

1. Agent attitudes. As well known, there is no complete agreement on which attitudes characterize agenthood yet. There are some proposals for identifying a set of "mandatory" attitudes which should discriminate between an agent and a non-agent piece of software [33], but no standard definition yet has been accepted. After all, it seems that reasonably agenthood should be a soft concept rather than a "hard" Yes/No property.

As already pointed out, we do not want to commit to some particular definition of agenthood from the literature, as this would exclude all the environments from our analysis which do not comply with it. Whatever the chosen definition, it would exclude a large part of the existing systems. We prefer to individuate a set of "interesting" features and assess, for any analyzed MASDK, whether it supports them or not[1]. We group the attitudes into *basic* attitudes, which are close to the very core of agenthood, and *advanced* attitudes, which are desirable but not of central interest.

[1] A feature is *supported* by a MASDK, if it provides built-in constructs, libraries, or other facilities for easily modeling that feature.

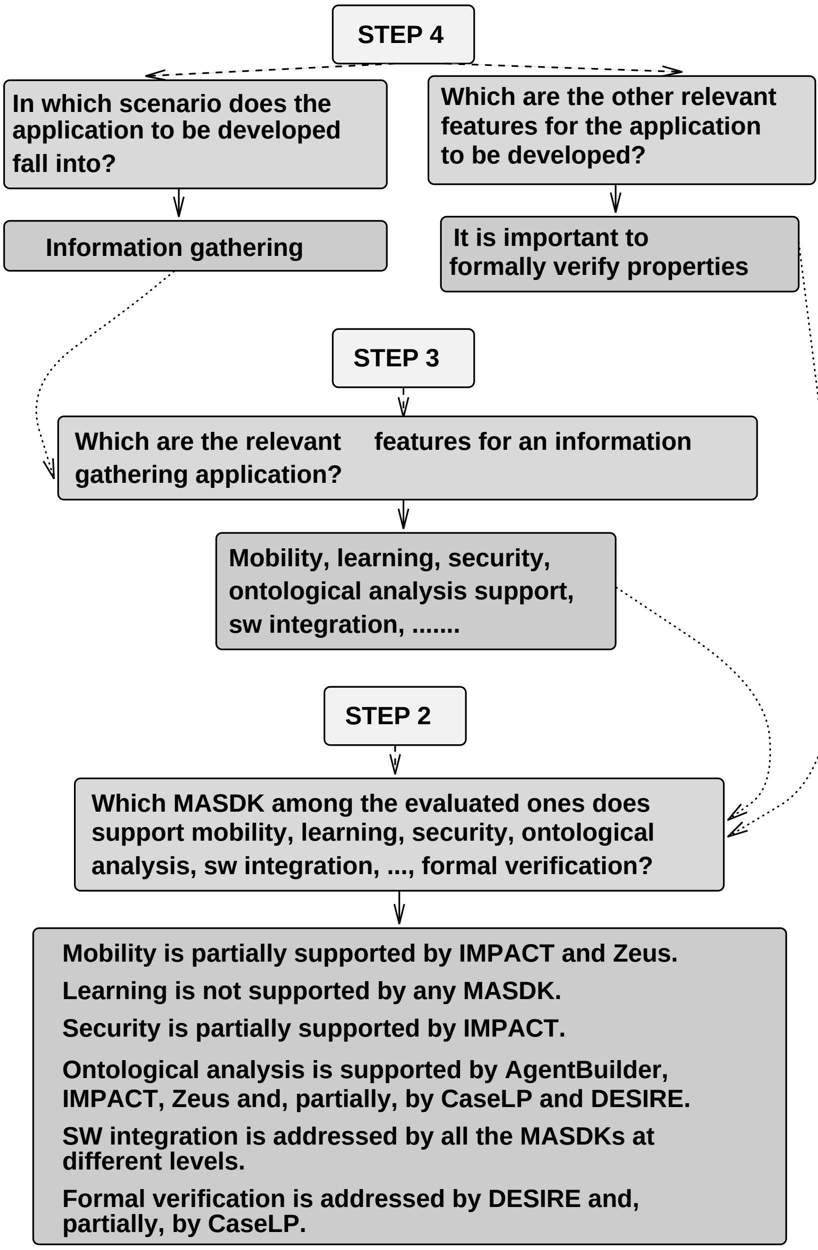

Figure 2: Application-driven selection of MASDKs: an example

2. Software engineering support. The development of a multi-agent system is a complex task which greatly benefits from the adoption of a well-defined engineering methodology. Instruments would be available for helping the developer during the modeling and development stages. Not all the MASDKs provide such kind of support, so it is interesting to understand which of them address the engineering of the MAS development and which do not. This point is expanded on in Section 2.2.

3. Implementation of agents and MAS. Once the structure of the MAS is clear and mature enough for implementation, it becomes necessary to understand in which language the agents will be finally implemented using one MASDK, which debugging facilities the MASDK provides to implement them, and which standard agent skeletons are eventually available. These features are mainly related with the way in which the MASDK helps the developer in implementing a correctly working MAS with the least effort and in the shortest possible time (Section 2.3).

4. Technical issues. The technical aspects which characterize a MASDK are more or less the same as those characterizing languages and toolkits for building large, complex software systems. Typical needs are physical distribution of agents, current agent execution, letting (some of) them roam across the network in order to minimize the time/cost of their task, and, finally all the related safety and security issues. Also the capability of integrating external and/or legacy software in the MAS is crucial for many applications. Section 2.4 expands on this point.

5. Economical aspects. Finally, as pointed out in [56], different issues are of particular interest from a business standpoint. The availability of on-line consulting and training, the supported platforms, the system requirements etc, will be discussed in Section 2.5 as relevant aspects in the choice among different candidate MASDKs.

2.1 Agent attitudes

One of the most general definitions of *agent* is the one proposed in [33]. It describes an agent as

> "... a computer system, *situated* in some environment, that is capable of *flexible autonomous* actions in order to meet its design objectives.
> Situatedness means that the agent receives sensory input from its environment and that it can perform actions which change the environment in some way.
>
> By autonomy we mean that the system should be able to act without the direct intervention of humans (or other agents), and that it should have control over its own actions and internal state. [...] By flexible, we mean that the system is:
>
> - responsive: agents should perceive their environment and respond in a timely fashion to changes that occur in it;
> - pro-active: agents ... should be able to exhibit opportunistic, goal-directed behavior and take the initiative when appropriate;
> - social: agents should be able to interact, when appropriate, with other artificial agents and humans."

Thus, **situatedness**, **autonomy**, **responsiveness**, **pro-activeness** and **sociality** are basic agent attitudes that we take into consideration. Sociality, which is a key feature in MAS, includes among others the following aspects:

- **Agent Communication Language.** In [29] software agents are defined as

  > "software 'components' that communicate with their peers by exchanging messages in an expressive *agent communication language*. ... The salient feature of the language used by agents is its expressiveness. It allows for the exchange of data and logical information, individual commands and scripts."

  Research in the field of Agent Communication Languages (ACLs) is very lively, and it has led to the establishment of two fully-specified ACLs: the FIPA standard [26] and KQML [44]. A MASDK supporting a standard ACL enhances communication standardization and re-usability of agents in different environments.

- **Communication protocols.** This issue can be analyzed from two points of view, reflecting different levels of abstraction. At a high level, the MAS developer may be just interested in the way agents exchange structured information. This can be achieved through different methods, e.g. via message passing (upon which ACL is built), by means of a blackboard, by method invocation, etc. Furthermore, the mode of message passing (point-to-point, broadcast, or multi-cast) is another important feature, which may be relevant for the choice of a MASDK.

  From a low level point of view, the developer must be aware of the data transmission protocol used within the MASDK, which allows him/her to use already available features in order to maintain certain communication properties. For example, if (s)he knows that data exchange between agents is based on TCP/IP, then (s)he can avoid to re-implement the basic safety mechanisms which TCP/IP already ensures.

- **Coordination protocols.** Many protocols for coordination and negotiation among agents have been proposed (see [32] and [37] for an introduction to the topic); the availability of a library of coordination protocols in a MASDK avoids to re-invent or just re-implement already developed strategies.

- **Human-agent interaction.** In order to develop agents which intelligently interact with human beings [39], advanced capabilities such as speech recognition, natural language understanding, image recognition, etc, should be provided by the MASDK.

It is desired that besides the above basic attitudes, agents should have further advanced attitudes, including the following:

- **Mental attitudes.** The agent is conceptualized in terms of mental notions, such as commitments, beliefs, desires, intentions [20].

- **Deliberative capabilities.** The agent explicitly represents its objectives and reasons about them. This issue is strictly related with the agent's planning capabilities, namely the ability to formulate and compare possible plans to solve a goal.

- **Meta-management.** The agent maintains a (nested) model of itself (*reflectivity)* and of other agents, and is able to reason about this model.

- **Emotionality.** The agent is ascribed emotional attitudes. Its behavior is guided not only by its rational goals but also by its emotions [60, 23, 54, 67].

- **Adaptivity.** The agent flexibly changes its strategies to adapt to the evolving environment and learns from previous experiences.

2.2 Software engineering support

The following issues are important from a software engineering perspective:

- **Methodology.** A well-defined methodology greatly helps the developer in passing from the initial informal ideas about the MAS to build the final, working MAS. Following a clear sequence of steps which gradually face more detailed and concrete aspects of the application design is a good approach in the development of any application. This is particularly true for the development of MASs, whose complexity can be properly faced and managed only by following a well-structured approach. For any step of the methodology, the MASDK should ideally provide languages and tools for its support.

- **Ontological analysis.** Ontologies are content theories about knowledge domains, developed to clarify knowledge structure and enhancing knowledge reuse and standardization [17]. They represent a powerful means to organize concepts and relations among concepts in an agent-based application, and to unambiguously describe features and properties of agents. In particular, ontologies are needed if heterogeneous information sources must be integrated in the same MAS.

- **Specification.** In the design of MAS various aspects must be considered: for example, it is important to provide an architectural description of the system (which types or classes of agents are involved, how many instances for each class, which services are provided/requested by the agents in a class, how provided services are linked to requested ones, etc.), as well as a description of the internal functioning of a single agent (which data structures does the agent use, which is the control flow among these data structures, in which language is the content of the data structures described, etc.). Thus, either a set of different specification languages or a single specification language capable of phrasing different views of the problem should be supported by the MASDK. The MASDK documentation should provide suggestions on how to specify the different views of the problem, and possibly how to map these specifications into the agent implementation language. Since automatic methods for correctly translating the specification into the agent implementation language are difficult to realize, some informal hints would be enough, at least for not safety critical applications. The adoption of a standard and well-known specification language for describing (part of) the MAS application turns out to be a good choice also to spread the agent technology use.

- **Verification.** When the MAS faces safety-critical problems, it may be necessary to formally verify at least a subset of its properties. In this case, verification mechanisms provided by the MASDK would be helpful.

- **Prototyping.** Usually it is not possible to formally verify the complete behavior of the MAS. A viable way of testing the MAS design correctness is to build a working prototype and to analyze its behavior on different, relevant scenarios. If the specification language(s) previously adopted is (are) executable or animable, the prototype is obtained for free.

- **Simulation.** Finally, to get statistic information on the execution run, simulation capabilities should be provided by the MASDK.

2.3 Agent and MAS implementation

As for the implementation of a single agent or of a whole MAS, the following aspects are relevant at a macro level:

- **Agents implementation language.** After the specification and testing phases are completed and the developer is confident enough in the correctness of the MAS design, it is time to *implement* the MAS. The code for the agents may be written either in some commercial language or in an ad-hoc proprietary language. Depending on the skills and personal preferences of the MAS developer(s), on the already developed pieces of code which should be "agentized" and on other relevant issues (efficiency, portability, ...), an agent implementation language will prove more or less suitable for different applications.
- **Physical environment models.** When a simulation of some physical process is performed, the environment where the process takes place must be simulated. A library of explicitly modeled environments can be provided to the MAS developer.
- **Agents and MAS definition GUI.** A facility which would help the development of (possibly) correct applications is the presence of graphical interfaces which guide and, in some sense, constraint the definition of the agents and/or of the MAS as a whole.
- **Agents skeletons.** The availability of agents skeletons that model standard roles, into which application-dependent knowledge must be plugged in helps the agent developer in building correct applications faster and more easily, since potential sources of programming errors are reduced.
- **Utility agents.** There are different agents offering services to the MAS community which do not depend on the particular application domain. For example, *broker* agents or *yellow* and *white pages* agents can be defined and then integrated in different systems without requiring any change. For this reason, it is to be hoped that the MASDK already provides them, allowing the MAS developer to concentrate on the domain-dependent agents definition.
- **Debugging facilities.** Debugging a MAS is, because of its distributivity and concurrent nature, a difficult and complex task in general. For the development of a correct, reliable, and robust system, automated support in this direction is needed.

2.4 Technical issues

The needs of different applications have led to the following desired technical MASDK features:

- **Mobility.** When the agents need to retrieve information which is scattered over different nodes of a network, mobility may represent an efficient way for achieving this task.
- **Distribution.** If the agents do not roam across the network, they may however distributed. The MASDK must provide all the facilities for allowing the agents to reside in different physical locations and to safely inter-operate.
- **Concurrency.** Any agent must possess its own thread of control. Thus, agents find a natural implementation in concurrently executing processes.
- **Security.** When mobility and distribution come into play, mechanism are required to ensure security. The agent developer must be sure that his/her mobile agents will not be damaged by malicious entities during their roaming, and that vice versa that his/her stationary agents will not be damaged by malicious mobile agents. Furthermore, information exchanged among distributed entities must not be intercepted or corrupted.

- **Real-time control.** Depending on the application, the need for real-time control and response may arise.
- **Software integration.** It is quite difficult to imagine that a large, complex MAS is built completely from scratch. The realistic scenario is a MAS where legacy code should be seamlessly integrated with newly developed agents and external software packages. The easy integration of external software and packages within the MAS satisfies the need of software reuse, which characterizes most of the current software production.
- **Other issues.** Finally, other relevant aspects are the platforms where the MASDK can be installed, the language in which it is implemented, the system resources it requires, the availability of trial releases and its cost.

2.5 Economical aspects

From a management perspective, the following issues are important:

- **Vendor organization.** Knowing the organization providing the MASDK is relevant in the choice of a product:

  > "... a historical profile of the vendor and the product will be taken into account so that longevity of the relationship can be assured [56]."

- **Training, documentation and support.** The availability of documentation, as well as training facilities and client support on the long period are relevant in choosing a MASDK. To quote [56] again,

  > "The availability of on-site consulting and training for the initial "burn-in" period will set the stage for subsequent relationships. [...] Much knowledge is gained from a product's installation base and associated developer groups, whereby features and quirks can be formally and informally documented."

3 Evaluation of some MASDKs

In this section, we provide an evaluation of five MASDKs, according to the features identified in Section 2. Our evaluation is mainly based on our knowledge, as far as CaseLP and IMPACT are concerned, and on the available documentation and on the information that some researchers[2] kindly provided to us, as far as the other toolkits are concerned. Some of our judgments may prove not appropriate or superficial, due to a lack of deep experience with the respective tools. However, even if some debates and criticisms will arise, we think that our evaluation is useful as a starting point for finding *the right MASDK to do the right thing*, to paraphrase J.P. Müller's influential paper [47].

The criteria we followed in selecting the MASDKs for evaluation are subjective, and our choice does by no way mean that they are preferable or better than other MASDKs. There are at least forty MASDKs which we intended to evaluate initially. However, this enterprise soon revealed as a titanic work, and we had to drastically reduce their number. We thus had to identify a smaller set of MASDKs to start with and we decided to consider, among the MASDKs we had some knowledge about, the ones which seemed to present a range of different features, in order to test our approach on toolkits heterogeneous enough to represent a significant case-study. Part of our future work will be to extend the comparison to a large set of MASDKs.

[2] See the acknowledgments.

3.1 An introduction to the evaluated MASDKs

The MASDKs we have evaluated are: AgentBuilder; CaseLP; DESIRE; IMPACT; and Zeus.

AgentBuilder [55] is a commercial product developed by Reticular Systems, Inc. It consists of two major components: the development tools and the runtime execution environment. The first ones are used for analyzing an agent's problem domain and for creating an agent program that specifies agent behavior; the run-time system provides a high-performance agent engine that executes these agent programs.

CaseLP (Complex Application Specification Environment based on Logic Programming) [63] is a research environment developed at the Computer Science Department of Genova University (Italy). It is mainly conceived as a prototyping tool for agent-applications; it focuses on the prototype development method, the ability of integrating agents written in different specification languages, and the ability of simulating the resulting MAS behavior.

The DESIRE (DEsign and Specification of Interacting REasoning components) research program [65] is carried out at the Artificial Intelligence Department of Vrije University Amsterdam. It focuses on the study of compositional multi-agent systems for complex, distributed tasks. A framework is provided which supports all the development phases of the compositional MAS.

IMPACT (Interactive Maryland Platform for Agents Collaborating Together) [64] is a multinational project whose aim is to define a formal theory of software agents, implement it (or carefully specify fragments of it) and develop a suite of applications on top of this implementation. The integration of heterogeneous software modules within the same MAS is performed by adding a semantic wrapper to the bodies of software code, in order to "agentize" them.

Zeus [14] is an environment developed by British Telecommunications, plc. for specifying and implementing collaborative agents, following a clear methodology and using the software tools provided by the environment. The Zeus approach to MAS development consists of analysis, design, realization and runtime support. The first two stages of the methodology are described in detail in the documentation, but they are not supported by software tools. The last two stages are supported by software tools.

3.2 Feature assessment

We have organized the evaluation and comparison of the MASDKs by using tables. Our aim is to synthesize the results of our analysis and help discovering whether a MASDK supports a feature in the quickest and easiest way. The reader interested in more details may go to the Appendix, where explanations and further background information is provided.

For the assessment of features, we use in the tables the sign

- $\otimes$, if the respective feature is fully supported;
- $\oslash$, if a feature is partially supported;
- $\odot$, if a feature is not supported.

Depending on the kind of the feature, being fully, partially, or not supported may have different meanings. The more "implementative" features (for example, mobility, security, adaptivity, etc) are *fully supported* if they are integrated in the MASDK, namely, there are libraries or tools that implement them in a well-experimented and reliable way. A feature whose integration has been studied only from a theoretical point of view, or whose implementation is not completely tested, is considered *partially supported*. Since we are adopting a cautious approach, some features may appear as *not supported* even if some preliminary experiments on them have been made. As far as other kinds of features are concerned (for example, specification or documentation) the full support means that the feature has been addressed in the best possible way. Finally, some features, such the Agent Communication Language used or the MASDK price, are described by text. Where available, we have added references to papers which describe feature support.

3.3 Agent basic attitudes

As for the basic agent attitudes, we have that they are supported by all five MASDKs that we considered. Thus, in this sense they are all platforms for building agent systems.

3.4 Agent advanced attitudes

	Mental Attitudes	Deliberative	Reflective	Emotional	Adaptive
AgentBuilder	⊗	⊙	⊙	⊙	⊙
CaseLP	⊘ [6]	⊙	⊙	⊙	⊙
DESIRE	⊘ [8]	⊙	⊘ [13]	⊙	⊙
IMPACT	⊘ [22]	⊘ [21]	⊘ [22]	⊙	⊙
Zeus	⊗ [50, 42]	⊗	⊙	⊙	⊙

An observation emerging from this table is that the effort in building agents which mimic human behavior is mainly directed towards the rational aspect of humanity (mental notions, planning, reflectivity) rather than on the human emotional side[3]. From the information we got from the various MASDK teams, it was clear that most of the "rational" attributes had been (at least) taken into consideration, even if they are not integrated in the toolkit. For example, applications involving planning have been developed using all the toolkits, even if libraries of planning strategies are provided only by two of them. On the contrary, applications involving emotional agents have not been developed with any toolkit.

3.5 Social ability

	ACL	Communication Protocols	Coordination Protocols	Human-Agent Interaction
AgentBuilder	KQML [44]	RMI, TCP/IP	⊙	⊙
CaseLP	KQML +	int. data str.	⊙	⊙
DESIRE	FOL lang. [12]	int. data str., TCP/IP	⊗ [11, 9]	⊙
IMPACT	XML +	RMI	⊙	⊙
Zeus	FIPA [26]	TCP/IP	⊗	⊙

Note:

- The "+" after the ACL name means that the ACL usually used is the one written in the table, but it is possible to make the agents interact using any other ACL.
- FOL lang. means an ACL based on first order logic.
- int. data str. means internal data structures.

[3]Recall that in this context, by emotions we mean feelings like *anger*, *envy*, *pride*, etc.

This table shows that the five MASDKs we have considered doe not face issues of Human-Computer Interaction. A rough interpretation of this result, combined with the lack of support for emotionality evidenced in the previous section, suggests that these MASDKs are not suitable for developing applications where believable characters interact with human users using some human-like means of communication (speech, vision, ...). Even if the set of MASDKs we are considering is too small for having any statistical relevance, our feeling is that the world of "believable emotional agents" and the world of "intelligent agents" are still separate, each one facing only particular aspects of agenthood. We also have the feeling that agents showing a behavior determined by emotions are definitely less addressed than agents which show behaviors determined by rationality.

3.6 Software engineering support

	Method	**Ontology**	**Specification**	**Verification**	**Prototype**	**Simulation**
AgentBuilder	$\otimes$	$\otimes$	$\oslash$	$\odot$	$\otimes$	$\odot$
CaseLP	$\otimes$	$\oslash$	$\otimes$	$\oslash$ [5]	$\otimes$	$\otimes$
DESIRE	$\otimes$	$\oslash$	$\oslash$	$\otimes$ [24, 35]	$\otimes$	$\odot$
IMPACT	$\otimes$	$\otimes$	$\oslash$	$\odot$	$\otimes$	$\otimes$
Zeus	$\otimes$	$\otimes$	$\otimes$	$\odot$	$\otimes$	$\otimes$

All MASDKs that we considered cover most of the issues concerning Software Engineering. Due to the complexity of MASs, support in this direction is perceived as a *must* for a toolkit which aims at constructively helping the developer in his/her hard task.

3.7 Implementation of agents and MAS

	Implementation Language	**Skeletons**	**Environment model**	**GUI**	**Utility Agents**	**Debugging**
AgentBuilder	RADL	$\odot$	$\odot$	$\otimes$	$\odot$	$\otimes$
CaseLP	ProlAg	$\odot$	$\odot$	$\otimes$	$\odot$	$\otimes$
DESIRE	see Appendix	$\otimes$ [10]	$\odot$	$\otimes$	$\odot$	$\otimes$
IMPACT	see Appendix	$\oslash$	$\odot$	$\otimes$	$\otimes$	$\otimes$
Zeus	see Appendix	$\oslash$	$\odot$	$\otimes$	$\otimes$	$\otimes$

It emerges from this table that all the MASDKs considered lack an explicit model of the surrounding environment. As it will be discussed in Section 6, this aspect is mainly concerned with *Artificial Life*.

3.8 Technical issues

	Mobility	**Distributive**	**Concurrency**	**Security**	**Real-time control**	**Software integration**
AgentBuilder	$\odot$	$\otimes$	$\otimes$	$\odot$	$\oslash$	Modules meeting JNI
CaseLP	$\odot$	$\odot$	$\odot$	$\odot$	$\odot$	ECLiPSE DB, Tcl/Tk, other
DESIRE	$\odot$	$\otimes$	$\otimes$	$\odot$	$\odot$	Sockets, files, pipes
IMPACT	$\oslash$	$\otimes$	$\otimes$	$\oslash$ [4]	$\oslash$	Many packages
Zeus	$\oslash$	$\otimes$	$\otimes$	$\odot$	$\odot$	JDBC and other packages

	Platform	Implementation language	System resources	Price	Evaluation releases
AgentBuilder	Solaris, Windows, Linux, IRIX	Java	133MHz Pentium, 32 MB RAM	from $495.00	Free evaluation of Agent Builder Pro
CaseLP	Windows, Linux	SICStus Prolog, ECLiPSE	133MHz Pentium, 16 MB RAM	nca	Not yet
DESIRE	Solaris, Windows, Linux	Prolog, xpce, Java, C	500MHz Pentium, 64 MB RAM	nca	Free for course attendees
IMPACT	Solaris, Windows, Linux	Java 1.2.2	Disk Space: 19 MB	tbn	tbn
Zeus	Solaris, Windows	Java	133MHz Pentium	nca	All releases are free

nca… not commercially available
tbn… to be negotiated

Three results can be extracted from the tables above:

- Mobility is not fully supported by any MASDK; IMPACT has integrated this feature, but some issues are still under consideration, and Zeus can integrate mobile agents, but experimentation in this direction is still at the beginning. Mobility is not considered a characterizing feature of agency, but its importance is growing up quickly. We think that it will be supported by most MASDKs in a short time.

- Security is addressed only by IMPACT, even if it is not implemented in the toolkit. Since, in our opinion, mobility requires good security mechanisms for preventing damages caused or suffered by the mobile agent, we think that the lack of security mechanisms in the MASDKs we are considering is due to the lack of support of mobility.

- No MASDK can be labeled as completely suitable for building real-time systems. There are many reasons for this: agents are often described declaratively, as rule-based systems; the rules may be executed by an interpreter (like in CaseLP) or compiled into some implementation language. In any case, the efficiency of their execution cannot be comparable with some procedure written ad-hoc for the task. Moreover, the implementation language is often Prolog or Java, which are not suitable for real-time applications. Finally, even if this is not stated anywhere, we think that the MASDKs we are considering have been (at least initially) conceived to *simulate* real applications, rather than to *implement* them. The attention that most MASDKs deserve to prototyping and simulation issues, despite to mobility and security issues, confirms our opinion.

3.9 Business issues

	Organization	Training	Documentation	Support
AgentBuilder	Reticular Systems Inc.	$\otimes$	$\oslash$	$\otimes$
CaseLP	CS Dept., Genova Univ. (Italy)	$\odot$	$\oslash$	$\odot$
DESIRE	AI Dept., Vrije Univ. (Nederlands)	$\otimes$	$\oslash$	$\oslash$
IMPACT	Multinational Project	$\otimes$	$\otimes$	$\odot$
Zeus	British Telecommunications plc.	$\odot$	$\otimes$	$\otimes$

We have observed that universities provide scarce support, documentation, and training on the toolkits they develop, while commercial institutions tend to better advertise their products (even if they are free) and provide an easier-to-read documentation. Documentation is clearly oriented towards people having no or little experience with agents, while university research teams presumably seem to expect that people interested in their toolkits are at an advanced level, and thus share some common knowledge on basic concepts.

4 Scenarios

In order to provide some guidelines for deciding which MASDK is best-suited for a particular application (dealt with in Section 5), we first need to identify the characterizing features of some potential application domains for MAS technology.

Since it almost impossible to consider all the areas where the MAS technology could be applied, we confine here to analyze some fields where it has already been successfully applied. In particular, we consider the application domains from Section 4 of [33], thus complementing the considerations there.

Our aim is to outline, for any application domain considered, the most useful features among those we identified in Section 2 that agents in this domain should possess. These features can be used as keys to search the most suitable MASDKs in Section 3.

The set of *useful features* we identify for any scenario derives from our experience. Since we are limiting ourselves to the features introduced in Section 2, this set is certainly not exhaustive, and other features may prove relevant for the scenarios which which we do not consider in this paper. Furthermore, even under this restriction, we may analyze the quite large domains only superficially. The analysis in this section is at a general level which needs to be refined for particular scenarios, and thus only draw a rough picture. Nonetheless, it serves as a *starting point* for helping a system developer in choosing a MASDK in an application-driven fashion.

4.1 Industrial applications

4.1.1 Manufacturing.

The adoption of the MAS technology in manufacturing applications has found a large consensus, since manufacturing enterprise entities may be modeled as interacting agents. The **simulation** of the real system is then adopted as a decision support system, or to forecast future situations and automatically issue some optimal sequence of actions. The agents may be **distributed**, but **mobility** is usually not needed. The large amount of variables characterizing a manufacturing process may lead to very different situations the agents have to cope with. For this reason, an agent should **dynamically plan** its behavior according to the current situation, since providing the agents with pre-defined strategies for all the possible situations may not be feasible in many situations. **Negotiation** and **coordination protocols** are often adopted, and thus support in this respect from a MASDK would be of great help. Since the final goal of the MAS is to simulate the behavior of a real system, **real-time control** may be dispensable.

4.1.2 Process control.

The adoption of **autonomous, reactive** agents for process control has led to successful results thanks to their similarity to process controllers which are autonomous reactive systems. The major advantages of applying agent technology to process control come from the easiness of reconfiguration and maintenance of **distributed** and **decentralized** control architectures. The complexity of the process task can be mastered by **distributed planning** and **execution**. The ability of agents to take decisions in a **timely fashion**, when necessary, and of developing **long-term plans** for less time-critical tasks demonstrates all its potential in this kind of applications.

4.1.3 Telecommunications.

The current trend in telecommunications is influenced by a great number of factors. Convergence of traditional telephony and data network worlds, blurring of boundaries between public and private networks, and the emergence of the "information super-highway," all contribute to the continuous evolution of the telecommunications field. To address the emerging need for a new technology dealing with all these issues, the agent approach seems very promising. In order to cope with the issues in telecommunication, agents must be **reliable**, **secure**, and they should act in **real-time**. **Negotiation** capabilities may also prove useful in order to solve conflicts among different entities interested in provided some network service, for example, setting up a call [59]. However, the key issue for the adoption of agent-based technology in the telecommunication field is **mobility**. As pointed out in [3], mobile agents prove extremely effective for network management. In particular they can be adopted for the following tasks.

- **Network Modeling**: mobile code is a convenient vehicle for performing discovery tasks [57], where "discovery" might target many goals, from finding the devices of the network to building more detailed models of the network.

- **Fault Management**: mobile agents can be exploited for network diagnosis and for remote maintenance of heterogeneous elements. Network diagnosis can be performed, for example, by societies of small, biologically inspired and relatively **simple agents** that need to **cooperate** for achieving the intelligence which is needed for diagnosing network faults [66]. Such agents can incorporate **learning techniques**, which may improve their future behavior.

- **Configuration Management**: provisioning services in telecommunication networks is a complex process, which usually involves several parties. Mobile agents can handle those tasks in an autonomous way [51], and they can also be adopted to implement plug-and-play network components as shown in [53].

- **Performance Management**: certain aspects of measuring the performance of networks are difficult if a centralized server is used. Network delays make measurement precision questionable. Instead of remotely polling network elements, a mobile agent can be dispatched to perform an analysis of the component locally, with more accurate results due to the absence of delays involved.

4.1.4 Transportation systems.

The **distributed** nature of MASs makes them suitable to model applications in the traffic and transportation management domain which requires interaction among geographically distributed entities. The real application is often **simulated** in order to understand how to properly dimension the transportation services, or to provide a decision support system to humans. A prototypical application in this field, where intelligent agents have been applied with valuable results, is supply chain management. To quote [58]:

> The supply chain of a manufacturing enterprise is a world-wide network of suppliers, factories, warehouses, distribution centers and retailers through which raw materials are acquired, transformed and delivered to customers [27]. This network also, in general, involves **heterogeneous**

> environments. .... Agent based approaches provide a natural way to design and implement manufacturing enterprise integration and supply chain management within such environments. .... However, the **security problem** resulting from the open architecture of agent based systems, particularly when using the Internet and the mobile agent technology, has been recognized by both manufacturing enterprises and the researchers in this area.

A related field in which intelligent agents should also be applied is software control for intelligent autonomous vehicles (as done, for example, in the ARGO project[4]). The basic agent attitudes **situatedness**, **autonomy**, **reactiveness** and **pro-activeness** are certainly highly relevant for this kind of applications. However, they center around intelligent physical agents rather than intelligent software agents, and thus other features, e.g. environmental conditions and ergonomic capabilities of an agent, may be in the focus of interest.

4.2 Commercial applications

4.2.1 Information management.

One of the most active and promising application fields for agent technology is information management. Two major problems that arise in this field concern information filtering and information gathering:

- *Information filtering* deals with the identification of incoming information which is relevant and important for a user. A filtering agent, providing this service, is usually **stationary** and **centralized**. It must be **autonomous**, since it must monitor the user's actions and provide useful suggestions without being asked to do so. As pointed out by P. Maes [41], agents of this kind must show **learning capabilities** in order to provide good suggestions about interesting information by looking at the user's behavior. The capability of interacting with the user by means of a **high-level interface** (speech understanding and natural language comprehension, for example) should allow non-expert users to benefit from the agent's services.

- *Information gathering* is related with finding the information which meets some requirements within a huge amount of data. In this context, **mobility** can prove really useful: the agent can ship itself or a clone towards the source of information and perform its research there without needing to fill up the user's machine with data to be examined and which eventually proves uninteresting. The amount of data exchanged over the network will be usually smaller if the agent is mobile and performs its task on a remote host. **Security** must be guaranteed. Also in this case **learning** capabilities are fundamental in order to maintain a faithful profile of the user's interests and needs.

In both cases, the application developed is not a simulation, but it is the "real thing". For this reason, **efficiency** and **robustness** of the chosen MASDK must be taken into account. Depending on the particular application domain, **integrating** an existing software for extracting semantic information from a text or indexing web pages, just to make an example, should be necessary. Since the information to be filtered or merged is heterogeneous in format and language, **ontological analysis** is usually necessary for achieving the task.

[4] `http://www.argo.ce.unipr.it/ARGO/english/index.html`

4.2.2 Electronic commerce.

In principle, the automation of some commerce can be faced by using **distributed**, **mobile**, **negotiating** agents. They should roam in the network to find the best offers for buying or selling goods and then engage in a negotiation in order to establish a final price they are willing to pay or to earn. **Security** issues and **robustness** and **efficiency** of code are also relevant, since the user wants to be sure that the management of her/his money is delegated to a reliable agent. The existence of an **ontology** of concepts is useful to allow heterogeneous agents to understand each other. Many commercial companies are already selling products that exploit agent technology for intelligent e-commerce.[5]

Another application for intelligent agents in the field of electronic commerce is *Internet Auctioning* [38]. At present, most business activity on the Internet is limited to publicizing the business opportunity and catalog based sales, but it will rapidly expand to include the negotiations conducted to settle the price of the goods or commodities being traded. These negotiations are currently conducted by human intermediaries through various forms of auctions, bidding systems for awarding contracts, and brokerages, but these activities could be also conducted by intelligent software agents, with a reduction of the cost and the time required for them. Research on this topic is quite promising, see [36, 16, 15] for some applications.

4.3 Entertainment applications

Entertainment has been a rapidly growing market in computer applications, and led to remarkable developments which include virtual reality and synthetic characters. One of the main issues in recent entertainment applications is that the synthetic characters be *believable*. They should show human-like **emotional attitudes** which allow a *willing suspension of disbelief* in the human player [25]. These synthetic characters should be able to interact in **real-time** with the player, they should **learn** from this interaction and act accordingly to the player's preferences and habits. Even if these applications just involve a simulation of reality, their increasingly wide and lucrative marketplace obliges them to be very **efficient** and **robust**.

An application domain which can draw from technology for entertainment is training and education. In fact, synthetic characters might be used in simulations of communication and other social processes, in a way such that the reactive behavior of a human agent in response to the behavior of a synthetic character representing another human counterpart such as a customer, suspect etc can be trained in a virtual environment. In particular, proper behavior in critical and emergency situations may be improved through such training.

4.4 Medical applications

A major obstacle to obtaining a comprehensive view of the health status of a patient is that the clinical data related to the patient are usually distributed across different medical centers. Moreover, their formats range from written text and standard data base records to image data like computer tomographies and radiographies. Agents with the ability to **integrate** ad-hoc software for extracting meaningful information from the heterogeneous sources of medical information could be adopted. They should be **distributed** across the medical centers, and they should be able to **cooperate** for combining the different information coming from different sources, and present a coherent view, possible in multi-media presentation. The **integration** of some well-established expert system in this MAS could lead to diagnostic capability of the system.

[5] See `http://agents.umbc.edu/Agents_for_.../Electronic_commerce/index.shtml` for links to some of these companies and to research projects in this field.

Other issues should be related with constantly monitoring the patient health status: a **reactive** agent capable of interpreting the sensory input from a patient and deciding whether the specialist should be informed provides a valuable help in constant monitoring of critical situations. Also in this case, the application is not a simulation, thus **robustness** and **reliability** are relevant issues to consider.

Projects adopting the agent technology for such health-care applications are not yet very widespread. There are a couple of prototypes, though, which have most of the features highlighted in the previous paragraph:

- The Patient Advocate [45] is designed to be an intelligent assistant for patient-centered health care. Residing on a home computer or special-purpose device and operating within an extended health-care information network, the Patient Advocate will extend medical expertise into the outpatient setting. It will have remote access to the patient's medical record, an understanding of the patient's health status and history, and a model of the patient's interest in health-related issues, preferences for modes and contents of interaction, etc.

- In [31] an agent-based system is described which was developed to assist in managing the care process in real-world settings. The agent's architecture combines a symbolic decision procedure for decision making with incomplete and conflicting information, the notions of commitments and convention for managing coherent cooperation, and a set of communication primitives for agent interaction.

5 Application-driven MASDK Selection

In this section, we discuss some interesting questions addressed in [61] about the choice of the right MASDK for developing an application.[6] The suggested answers are based on the previously identified features of MASDKs and on our analysis of existing scenarios.

What kinds of scenarios are to be developed?

In Section 4, we have described different scenarios, high-lightening the most relevant agents features for each of them. If the scenario to be developed falls inside an already analyzed class, it is likely that the individuated relevant features for that class will prove relevant also for the specific scenario. These features can help in choosing a set of suitable MASDKs for the application. Of course, this is just a rule of thumb and it should be used as such: the application to be developed may require different features than the described ones. Answering the following questions should help in refining the set of required features and thus the choice of the right MASDK. It can also help to find a set of suitable MASDKs when the scenario to be developed does not fall in any application classes of Section 4.

What are the scenarios to be used for? For instance, is it a research activity or is there a practical goal?

If the aim is to develop a commercial application, **economic** and **technical** issues play a very important role on the choice of the MASDK. Users of a commercial application want the application to be **robust**, **reliable**, **secure** and **efficient**. In order to assure that these requirements are met, the application developers need to choose a well-supported MASDK developed by a commercial or a research organization which provides

[6] The first seven questions are literally quoted from Sloman's paper; the eighth one is ours.

guarantees on the product and on the related support. On the other hand, for some kinds of research activity a **free of charge** prototype should be enough, even if some bugs or inefficiencies may arise.

Is the system being developed a simulation of something else, or is it "the real thing"?

When we analyzed some existing scenarios which use the MAS technology, we have already seen that there are applications which "simulate something else", for example manufacturing or transportation systems, and applications which are "the real thing", for example agents for information filtering or for medical applications. Depending on the kind and on the use of the application, the "simulations" may have **less efficiency** and **less real-time control** constraints than the "real things". This is true only if the simulation is carried out as an off-line process, useful to understand the real application features and to properly dimension the involved variables, or if it does not involve critical tasks. An agent installed inside a car, which by running a simulation tries to predict the behavior of other cars in the road and suggests which driving actions to take, must be both efficient and do this in real-time, and must moreover be fully reliable!

To what extent are the objectives and design strategies already well understood in advance of use of the toolkit?

If the objectives and design strategies are not already clear before choosing a MASDK, this may mean that the software engineering process for the application development is still in its early phases. Informal specifications are given and a deeper understanding of the design choices is required before implementing the application. In such a situation a MASDK which provides a clear **development methodology**, together with tools and languages for supporting its steps, may prove useful. It is possible that such MASDK helps in clarifying all the issues related with the scenario, but proves inadequate for the final implementation of the application. In this case, a second MASDK may be used for the implementation. The choice of this second MASDK should become easier thanks to the already performed identification of objectives and design strategies.

Which is more important: ease and speed of development and debugging, or speed of execution of the developed system?

The choice of a framework which allows for "fast development and debugging" strictly depends on the skills and preferences of the system developer(s). For this goal, the language which will be used to specify or implement the agents must be chosen in accordance with the developer(s) habits. The effort required for learning a new language, even if it allows for faster development, may be higher than using a language which the developer is familiar with. Determining the trade-off for this effort depends on the size of the application and the project, and should be carefully deliberated. Abstracting from the developer's personal preferences, there are frameworks which are more suitable for **fast prototyping** and frameworks which provide better facilities for the **MAS implementation**. The first ones could be chosen if ease and speed of development and debugging is more important than speed of execution of the development system. On the other hand, if the system needs to be implemented in C++, for example, to meet some efficiency requirements, the choice of the MASDK will be driven by this necessity.

Will thorough testing and collection of performance statistics be required?

If yes, then a MASDK supporting **simulation** capabilities will be needed.

Is it essential that the developed system be provably correct as in some safety critical applications?

For applications like these, formal **verification** techniques may be needed. A MASDK should obviously provide support in this respect, in order to prove properties of agents and MASs.

Does already developed software exist which can be reused in the new MAS? Is it already clear how to compose it?

The answer to the first question will be often "yes," while the answer to the second one will be often "no." In this situation, a MASDK which provides facilities for **integrating external software** and for developing hybrid **prototypes** (namely, prototypes built up by agents which are only specified and by already implemented software) would help. In fact, it would allow to clarify how to integrate existing software in a new application, by testing the behavior of a simplified prototype. If the (re)usability of external (legacy) software is not yet clear, it is very important to build prototypes in a flexible and fast way, in order to understand whether the existing component "fits" the new application as is, whether it needs to be modified, or whether it can not be reused at all.

6 Further Issues and Related Work

In this section, we first have a look at other agent scenarios, which differ from the "engineering" ones that we have considered in the previous sections either in nature or in spirit of the agents involved. We then briefly address related work on MASDK classification in the literature.

6.1 Other agent-related scenarios

6.1.1 Physical intelligent agents

In our paper, we have not dealt with scenarios where *physical intelligent agents*, situated within the physical world, should be developed. [7] Physical intelligent agents represent a very important category in the intelligent agents world. However, due to the peculiarities of problems that arise in their development, they should be treated different from *software agents*, and other features and criteria should be taken into account.

6.1.2 Digital butlers

Another category of agents whose development has not been discussed in detail involves so-called *digital butlers* [41]. Such semi-autonomous agents are very specialized on the high-level interaction with the user for providing a limited set of sophisticated services. This high-level interaction requires the ability of "understanding each other" by means of speech recognition and synthesis capabilities, natural language parsing, and artificial vision. Unfortunately, none of the MASDKs that we have analyzed explicitly supports these capabilities. Choosing a MASDKs which makes it easier to **integrate external packages** (e.g., IMPACT) would help: plugging in an agent the capability of understanding spoken natural language should be achieved by letting the agent access the proper external package. Of course, this cannot be as easy as if some dedicated component for this specific task would be available. On the other hand, one has greater flexibility in general as among various external packages an appropriate might be chosen. Also selecting a

[7] See `http://agents.umbc.edu/aw/Topics/Physical_Agents/index.shtml` for links to various intelligent robots sites, and [46] for an overview of the evolution trend in this field.

MASDK which supports the conceptualization of agents in terms of **mental** and **emotional** attitudes, with **deliberative** and **meta-management capabilities**, would help: the closer the agent model to a human being model, the easier the mutual understanding. Despite these very rough guidelines, we think that it would not be very easy to develop *digital butlers* using one of the MASDKs analyzed in Section 3. To our knowledge, no ad-hoc toolkits for developing such agents are currently available.

6.1.3 Artificial life

Finally, a third agent-related field which should be considered is artificial life simulation [30]. Artificial life is the discipline studying the behavioral and evolutionary foundations of cognition and intelligence. The research in this field ranges over various subjects, including adaptive behavior, coordinated perception and action, and evolutionary learning techniques (such as genetic algorithms). Artificial life agents are generally not regarded as intelligent software agents. Nonetheless, some of the MASDKs that we have analyzed should suit this kind of simulations, and we think it is worthwhile to spend some words on such a scenario.

Artificial life agents are always **situated** within a virtual environment which is explicitly modeled. For this reason, a MASDK providing a **physical environment model** can suite this application. The goals of agents are often implicit and they are achieved by **reacting** to changes happening in the environment. There is usually no need to conceptualize the agents in terms of **mental** and **emotional** attitudes, or to add **meta-management capabilities** to them. However, the agents should be able to **learn** and **adapt** to changes in the surrounding virtual environment. Agents communicate either explicitly or by obtaining information about other agents through sensing the environment. If communication takes place via message passing, the **communication language** is usually at a low level: neither speech acts and complex message structures, nor sophisticated coordination protocols are usually required. Since Artificial Life mainly involves **simulation** issues, the ability of quickly setting up a simulation, collecting different statistics, and presenting them in a clear and intelligible way is definitely more relevant than **efficiency**. The simulation may be **centralized**, and issues such as **code mobility** or **integration** of external software are not crucial.

6.2 Related work on MASDK classification

Important seminal contributions to the development of comparison and selection criteria for agent tools have been made at the *AAAI-98 Workshop on Software Tools for Developing Agents*, and in particular in the papers [61, 40, 56].

Sloman [61] poses a set of high-level question about agents tools, which we have addressed and answered already in Section 5. It should be noted that the answers provided by Sloman are different from ours, in particular the first one where we refer to a set of prototypical scenarios (telecommunications, medical care, etc.) for which we highlight some useful features in Section 4. Sloman does not refer to any particular application domain. He deeply analyzes what the internal agent architecture should look like, according to the application that should be developed; this aspect has been dealt with in our paper less in detail. Sloman also provides some toolkit design options which are relevant according to the questions above, but does not provide any systematic comparison among existing MASDKs. Sloman's paper and ours can be considered complementary, since they provide answers to the same questions from different perspectives.

Logan's paper [40] shares with our paper the goal of helping the researcher and developer to choose the most appropriate MASDK for his/her current problem. It outlines a heuristic mapping from features of an agent system to toolkits. The classification criteria adopted in [40] are divided into four groups:

1. the properties of the environment in which the agent system is embedded (observable vs partially

observable; deterministic vs non-deterministic; discrete vs continuous; single-agent system vs multi-agent system);

2. the types of actions it can perform (fallible vs infallible; action utility and cost; action types);

3. the kind of goals that can be attributed to the agent (autonomous generation of goals; achievement vs maintenance goals; single vs multiple goals; commitment to goals; utility of goals; meta-goals); and

4. the kind of beliefs that can be attributed to the agent (consistent, certain, nested propositional beliefs).

Logan demonstrates the classification criteria on mail filtering and transportation applications. However, no comparison of existing MASDKs according to the given criteria is described.

Clearly, while Logan's aim is similar, his criteria and ours are more or less orthogonal. The criteria in [40] are motivated by an agent architecture model, though, rather than from a whole agent system model as stand behind our criteria. In this sense, the criteria of Logan and ours are complementary, and may be combined in the evaluation of MASKDs in order to get a fine-grained view of the picture.

The short paper of Schoepke [56] individuates a set of business, technical, and practical issues which represent a *commercial* view of the MASDK selection problem. We followed these guidelines when we identified the the technical and business features described in Section 2. The main difference between Schoepke's work and ours is that he only deals with business, technical, and practical issues and thus only covers a subset of the criteria we have analyzed in our paper. Furthermore, no concrete MASDKs are analyzed.

An approach very similar to ours, but for a different purpose, is pursued by Müller in [47]. While we want to find "the *right MASDK* to do the right thing," Müller's aim is to find "the *right agent architecture* to do the right thing." Apart from their different goals, Müller's approach and ours are very close: after having identified some application areas for agent technology, he proposes a classification of agents based on the characteristics of the applications classes identified. Based on the classification, Müller provides rules of thumb to help a software engineer or system designer in his/her choice of the proper agent architecture.

7 Conclusions

In this paper, we made an effort towards providing a methodology and general guidelines for selecting a multi-agent system development kit (MASDK) in order to realize a multi-agent system, which takes on an application-driven perspective. We have presented and discussed a number of criteria and features of agent systems which are relevant to this choice, and have related them to various common scenarios and application domains. Furthermore, different from related earlier work in the literature, we have evaluated some available MASDKs on these criteria. In particular, we examined AgentBuilder, CaseLP, DESIRE, IMPACT, and Zeus as representatives of different types of MASDKs, with respect to the criteria that we have compiled.

One goal of our paper is to help an application developer to understand if any of the MASDKs considered is suitable for his/her application. If this turns out not to be the case, then the developer should apply the evaluation criteria to any other MASDKs, such that a set of potentially usable MASKDs emerges. Our experience is that this kind of selection support is barely needed, and we have received encouraging feedback in this matter. Furthermore, our paper is also of interest to the agent researcher, since it provide a handle to classify and compare any generic agent system tool which he/she develops to other such tools on a set of established features. We are aware that our work is not the final word on MASDK selection. However, it makes a contribution to this important issue on which further research can build upon.

Our work can be extended and augmented in various directions. An obvious direction is to extend our analysis to a larger set of MASDKs which are available. The number of MASKDs available to date is extremely difficult to assess; about 50 may be a reasonable order of magnitude, but the number is steadily growing. In a preliminary version of this paper, we have attempted to scholarly classify a large number of the existing MASDKs, but eventually had to quit since this would have become too complex. We believe that a focused selection of a few system representing different characteristics, as done in this paper, is informative enough to show range of characteristics in the classification framework.

Another direction, which is orthogonal to the previous, is to extend and refine the set of criteria which we fostered for MASDK characterization. Further criteria, including those considered in [40] and [61] could be taken into account. In this paper, however, we have confined to identify a core set of, as we believe, significant features for MASDK characterization. The set of features may also increase if the range of application areas is enlarged, or if applications are considered in more detail. We expect, however, that further extensions and refinements will not lead to a drastic change of the set of core features that we have identified, and thus will not render the contribution of this paper obsolete. Pursuing these and other issues is part of our future work.

Acknowledgements

The authors want to thank all the people who helped them in carefully describing the MASDKs, by answering a lot of tedious questions: the president and CEO of Reticular Systems, Inc., D. Ballard; all the participants to the Zeus mailing list, in particular J. Collins, D. De Beer, S. Hati, E. Mangina, and S. Thompson; C.M. Jonker and J. Treur from the DESIRE team; and J. Dix, T.J. Rogers, and V.S. Subrahmanian from the IMPACTgroup.

References

[1] E. Appiani, M. Martelli, and V. Mascardi. A Multi-Agent Approach to Vehicle Monitoring in Motorway. Technical Report DISI TR-00-13, 2000.

[2] D. Ballard (Principal Investigator). Intelligent Agents and Agent Communication Languages for Mission Operations Phase II - Final Report. NASA Goodard Space Flight Center, Greenbelt, MD. NASA Reference NAS-97018, Reticular Reference 35-9, June 1998.

[3] A. Bieszczad, B. Pagurek, and T. White. Mobile Agents for Network Management. *IEEE Communication Surveys*, 1(1):2–9, 1998.

[4] P. Bonatti, S. Kraus, and V. S. Subrahmanian. Secure Agents. Computer Science Department Technical Report, CS-TR-4068, University of Maryland, October 1999.

[5] M. Bozzano, G. Delzanno, M. Martelli, V. Mascardi, and F. Zini. Logic Programming & Multi-Agent Systems: a Synergic Combination for Applications and Semantics. In K. Apt, V. Marek, M. Truszczynski, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 5–32. Springer Verlag, 1999.

[6] M. Bozzano, G. Delzanno, M. Martelli, V. Mascardi, and F. Zini. Multi-Agent Systems Development as a Software Engineering Enterprise. In G. Gupta, editor, *Proc. of First International Workshop on Practical Aspects of Declarative Languages (PADL'99)*, pages 46–60, San Antonio, Texas, January 1999. Springer-Verlag. LNCS 1551.

[7] J. Bradshaw. Introduction to Software Agents. In J. Bradshaw, editor, *Software Agents*. AAAI Press/The MIT Press, 1997.

[8] F. Brazier, B. Dunin-Keplicz, J. Treur, and L. Verbrugge. Modelling Internal Dynamic Behaviour of BDI agents. In A. Cesto and P. Schobbes, editors, *Proceedings of the Third International Workshop on Formal Models of Agents, MODELAGE'97*. Springer Verlag, 1999. Lecture Notes in Artificial Intelligence.

[9] F. M. T. Brazier, F. Cornelissen, R. Gustavsson, C. M. Jonker, O. L. B. Polak, and J. Treur. A Multi-Agent System Performing One-to-Many Negotiation for Load Balancing of Electricity Use. *International Journal of Electronic Commerce*, 2000. Revised version submitted.

[10] F. M. T. Brazier, F. Cornelissen, C. M. Jonker, and J. Treur. Compositional Specification of a Reusable Co-operative Agent Model. International Journal of Cooperative Information Systems. In press., 1999.

[11] F. M. T. Brazier, C. M. Jonker, and J. Treur. Modelling Project Coordination in a Multi-Agent Framework. In *Proceedings of the Fifth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE'96*. IEEE Computer Society Press, Los Alamitos, 1996.

[12] F. M. T. Brazier, C. M. Jonker, and J. Treur. Compositional Design and Reuse of a Generic Agent Model. *Applied Artificial Intelligence Journal*, pages 491–538, 2000.

[13] F. M. T. Brazier and J. Treur. Compositional Modelling of Reflective Agents. *International Journal of Human-Computer Studies*, 1999.

[14] British Telecommunications. The Zeus Agent Building Toolkit. `http://www.labs.bt.com/projects/agents/zeus/`.

[15] M. Calisti and B. Faltings. Agent-Based Negotiations for Multi-Provider Interactions. In *Proceedings of ASA 2000, 2nd International Symposium on Agent Systems and Applications*, Zurich, Switzerland, 2000.

[16] M. Calisti, B. Faltings, and S. Mazziotta. Market-Skilled Agents for Automating the Bandwidth Commerce. In *Proceedings USM 2000, 3rd IFIP/GI International Conference on Trends towards a Universal Service Market*, Munich, Germany, 2000.

[17] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins. What Are Ontologies, and Why Do We Need Them. *IEEE Intelligent Systems*, 14(1):20–26, 1999.

[18] P. Ciancarini and M. Wooldridge, editors. *Agent-Oriented Software Engineering: The State of the Art*. Springer-Verlag, 2001. LNAI Volume 1957.

[19] R. Degl'Innocenti. UML come linguaggio per specificare agenti: analisi, estensione ed integrazione in CaseLP. Master's thesis, DISI – Università di Genova, Genova, Italy, 2000. In Italian.

[20] D. C. Dennett. *The Intentional Stance*. MIT Press, Cambridge, MA, USA, 1987.

[21] J. Dix, H. Munoz-Avila, and D. Nau. IMPACTing SHOP: Planning in a Multi-Agent Environment. *Annals of Math and AI*, 2001. currently under submission.

[22] J. Dix, V. Subrahmanian, and G. Pick. Meta Agent Programs. *Journal of Logic Programming*, 46(1–2):1–60, 2000.

[23] C. Elliot. The Affective Reasoner: A Process Model of Emotions in a Multi-Agent System, 1992. PhD thesis, Northwestern University. The Institute for Learning Sciences, Technical Report No. 32.

[24] J. Engelfriet, C. M. Jonker, and J. Treur. Compositional Verification of Multi-Agent Systems in Temporal Multi-Epistemic Logic. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V*, pages 177–194. Springer-Verlag, 1999. LNCS 1555.

[25] L. N. Foner. Entertaining Agents: A Sociological Case Study. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 122–129, Marina del Rey, CA, USA, 1997. ACM Press.

[26] Foundation for Intelligent Physical Agents. FIPA Home Page. `http://www.fipa.org/`.

[27] M. S. Fox, J. F. Chionglo, and M. Barbuceanu. The Integrated Supply Chain Management System, 1993. Internal Report, Dept. of Industrial Engineering, Univ. of Toronto.

[28] S. Franklin and A. Graesser. Is it an Agent, or Just a Program? In J. P. Müller, M. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III*, Berlin, Germany, 1997. Springer-Verlag. LNAI Volume 1193.

[29] M. Genesereth and S. Ketchpel. Software Agents. *Communications of the ACM*, 37(7):48–53, 1994.

[30] N. Gilbert and K. G. Troitzsch. *Simulation for the Social Scientist*. Open University Press, 1999.

[31] J. Huang, N. R. Jennings, and J. Fox. An agent-based approach to health care management. *Applied Artificial Intelligence: An International Journal*, 9(4):401–420, 1995.

[32] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[33] N. R. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38, 1998.

[34] N. R. Jennings and M. Wooldridge. Agent-Oriented Software Engineering. In J. Bradshaw, editor, *Handbook of Agent Technology*. AAAI/MIT Press, 2001. To appear.

[35] C. M. Jonker, J. Treur, and W. de Vries. Compositional Verification of Agents in Dynamic Environments: a Case Study. In F. van Harmelen, editor, *Proc. of the KR98 Workshop on Verification and Validation of KBS*. Morgan Kaufmann, 1998.

[36] A. Kashyap. Design And Development of a Framework for Testing and Developing Internet Auction Systems, 2000.

[37] S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94(1–2):79–97, 1997.

[38] M. Kumar and S. I. Feldman. Internet Auctions. `http://ftp.sage.usenix.org/publications/library/proceedings/ec98/full_p%apers/kumar_auctions/kumar_html/kumar.html`.

[39] H. Lieberman. Autonomous Interface Agents. In *Proceedings of the ACM Conference on Computers and Human Interface, CHI-97*, Atlanta, Georgia, 1997.

[40] B. Logan. Classifying Agent Systems. In *Proc. of the AAAI-98 Workshop on Software Tools for Developing Agents*, Wisconsin, USA, 1998.

[41] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7), 1994.

[42] E. Mangina et al. MUlti Agent SYstem Knowledge Representation For Power Plant. In *Proc. of the International Conference on Intelligent System Application to Power Systems*, Budapest, Hungary, June 2001.

[43] M. Martelli, V. Mascardi, and F. Zini. A Logic Programming Framework for Component-Based Software Prototyping. In A. Brogi and P. Hill, editors, *Proc. of 2nd International Workshop on Component-based Software Development in Computational Logic (COCL'99)*, Paris, France, September 1999.

[44] J. Mayfield, Y. Labrou, and T. Finin. Evaluation of KQML as an Agent Communication Language. In *Intelligent Agents II*. Springer Verlag, 1995. Lecture Notes in Artificial Intelligence 1037.

[45] S. Miksch, K. Cheng, and B. Hayes-Roth. An intelligent assistant for patient health care. In *Proceedings of the First International Conference on Autonomous Agent*, Marina del Rey, CA, USA, 1997.

[46] H. Moravec. Rise of the robots, 1999.

[47] J. P. Müller. The Right Agent (Architecture) to Do the Right Thing. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V*, pages 211–225. Springer-Verlag, 1999. LNCS 1555.

[48] H. Munoz-Avila, J. Dix, D. S. Nau, and Y. Cao. IMPACTing shop: Planning in a Multi-Agent Environment. Currently under submission, 2001.

[49] D. S. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*. Morgan Kaufmann, 1999.

[50] H. Nwana, D. Ndumu, L. Lee, and J. Collis. ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. *Applied Artifical Intelligence Journal*, 13(1):129–186, 1999.

[51] B. Pagurek et al. Network Configuration Management In Heterogeneous ATM Environments, July 1998.

[52] C. J. Petrie. Agent-Based Software Engineering. In J. Bradshaw and G. Arnold, editors, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, Manchester, UK, 2000. The Practical Application Company Ltd.

[53] K. S. Raza. Implementation of Plug-and-Play Printer with Mobile Agents. Technical Report.

[54] W. S. Reilly and J. Bates. Building Emotional Agents. Technical Report CMU-CS-92-143, School of Computer science, Canergie Mellon University, Pittsburgh, PA, 1992.

[55] Reticular Systems. AgentBuilder. Inc. `http://www.agentbuilder.com/`.

[56] S. H. Schoepke. A Business View Regarding the Selection Of Agent Development Toolkits. In *Proc. of the AAAI-98 Workshop on Software Tools for Developing Agents*, Wisconsin, USA, 1998.

[57] C. Schramm, A. Bieszczad, and B. Pagurek. Application-Oriented Network Modeling with Mobile Agents, 1998.

[58] W. Shen and D. H. Norrie. Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. *Knowledge and Information Systems, an International Journal*, 1(2):129–156, 1999.

[59] N. Skarmeas and K. L. Clark. Process Oriented Programming for Agent Based Network Management. In *Proceedings of ECAI Workshop on Intelligent Agents for Telecomminications Applications (IATA96)*. IOS Press, 1996.

[60] A. Sloman. Motives, mechanisms, and emotions. *Cognition and Emotion*, 1(3):217–233, 1987.

[61] A. Sloman. What's an AI Toolkit For? In *Proc. of the AAAI-98 Workshop on Software Tools for Developing Agents*, Wisconsin, USA, 1998.

[62] The IMPACT Team. IMPACT Applications. `http://www.cs.umd.edu/projects/impact/apps.html`.

[63] University of Genova. CaseLP. `http://www.disi.unige.it/research/Logic_programming/agent_pubs.html`.

[64] University of Maryland. IMPACT. `http://www.cs.umd.edu/projects/impact/`.

[65] Vrije Universiteit Amsterdam. The DESIRE Research Programme. `http://www.cs.vu.nl/vakgroepen/ai/projects/desire/desire.html`.

[66] T. White, B. Pagurek, and F. Oppacher. Connection Management using Adaptive Agents, 1998.

[67] I. Wright. Emotional Agents, 1997. PhD thesis, University of Birmingham.

[68] F. Zini and L. Sterling. Designing Ontologies for Agents. In M. C. Meo and M. Vilares-Ferro, editors, *Proc. of Appia–Gulp–Prode 1999: Joint Conference on Declarative Programming*, pages 29–42, L'Aquila, Italy, September 1999.

A MASDK Comparison: Details

This appendix contains a detailed analysis and comparison of the MASDKs introduced in Section 3.

A.1 Agent basic attitudes

In spite of the different agent definitions and conceptualizations, all the MASDKs we have analyzed support the definition and implementation of agents which comply to the basic model. Almost agents possess a mailbox, and this ensures that they can be social and situated (the surrounding environment may be sensed through message exchange). The existence of an engine stating the macro-actions an agent performs during its life-cycle ensures the autonomy of the agent. The reactiveness and pro-activeness of agents is also supported by all the frameworks.

A.2 Agent advanced attitudes

- **Mental attitudes.** AgentBuilder agents are conceptualized in terms of behavioral rules, beliefs, commitments, intentions and capabilities.

  The other MASDKs do not conceptualize agents in terms of mental attitudes, but they allow the definition of agents characterized by mental features:

  - [6] describes the specification of the BDI architecture in CaseLP.
  - in [8] a generic model for the internal dynamic behavior of BDI agents modeled in DESIRE is proposed.
  - [22] extends IMPACT agent programs by beliefs. This is not a real BDI architecture, but desires and intentions can be implemented through the agent action language and the IMPACT software code call mechanism.
  - [50] and [42] show extensions of the Zeus environment for supporting mental attitudes.

- **Deliberative capabilities.** AgentBuilder allows the developer to link a separate planning module into the basic agent inferencing cycle. The planning modules are not provided as part of the AgentBuilder toolkit: they must be implemented by the agent's developer according to the application features. Examples involving street route path planning have been developed.

  CaseLP and DESIRE do not provide planning facilities. Planning problems have been addressed for some specific cases but no general planners are provided.

  IMPACT is currently being extended with an AI planning system: [48] describes how to couple an HTN (Hierarchical Task Network) planner into the IMPACT framework. The idea is to create several specific agents in IMPACT that simulate together an efficient HTN planner called SHOP [49] (developed independently from Dana Nau in the planning community). Once these agents are fully implemented they can be seen as one particular reusable planning strategy.

  Zeus agents posses an internal Planner; the Planner reasons about how to obtain goals using the agent's database of tasks/capabilities and the publicly known capabilities for other agents. It is possible to configure the planner length, namely the number of time-grains that the agent will normally plan its activities. Depending on this length, the agents will be more reactive or more deliberative.

- **Meta-management.** From a syntactic point of view, it is possible to define agents whose beliefs may involve other agent's beliefs in all the MASDKs we are analyzing.

  DESIRE provides a generic model for reflective agents with meta-management capabilities [13].

  IMPACT faces this issue with a great rigor and provides a precise semantics to nested beliefs [22].

- **Emotionality.** Emotions are not considered by any MASDK.

- **Adaptivity.** Pre-defined learning strategies are provided by no toolkit.

A.3 Social ability

- **Agent Communication Language.** AgentBuilder agents communicate in KQML[44].

  CaseLP agents may communicate using KQML, or whatever language over which they agree. All the applications developed by now used a simplified version of KQML.

  Communication between agents is modeled in DESIRE by the activation of information links. The ACL shared by the agents is a first-order language described in [12].

  IMPACT has a simple, XML-based ACL built-in which serves for exchange of service offers and requests. However, through custom software packages and proper action any common ACL can be made available, independent of a particular application.

  Zeus agents share a common communication language which is FIPA-compliant [26], modulo some small differences[8].

- **Communication protocols.** All the MASDKs under consideration communicate via message-passing. As far as lower level communication protocols are concerned, the basic object to object communications mechanism supported by AgentBuilder is RMI (Remote Method Invocation). However, it is also possible to use TCP/IP sockets. The coming version of AgentBuilder will provide support for CORBA/IIOP.

  CaseLP agents are logically distributed but they are physically centralized, and they run as a unique process. At the lower level the exchange of information occurs via internal data structures. The coming version of CaseLP, D-CaseLP, will support CORBA.

  Agents executing within one process of a multi-agent DESIRE system communicate through internal data structures. Agents executing in different processes of a distributed multi-agent DESIRE system communicate through TCP/IP.

  IMPACT agents exchange messages via Java's RMI.

  Zeus installation requires that each host machine should be capable of TCP/IP communication, but there is no need for any middleware services to be installed.

- **Coordination protocols.** AgentBuilder includes a *Protocol Manager* which provides tools to specify the messages and conversational protocols between agents. However, no libraries of protocols are provided.

[8] These differences concern the request-forward performative, not implemented in Zeus, the format of addresses, the absence of resolvers and the low-level communication protocol.

CaseLP and IMPACT do not provide already implemented coordination protocols or facilities for implementing them.

DESIRE provides generic agent models for coordination and cooperative projects based on joint intentions and strategic user interaction [11]. In [9] the compositional development method DESIRE has been used to analyze, design, implement and verify a multi-agent system capable of negotiation for load management.

The same applies for Zeus, which provides a library of coordination and negotiation protocols.

- **Human-agent interaction.** No MASDK gives support in this direction.

A.4 Software engineering support

- **Methodology.** AgentBuilder method for constructing intelligent agents is composed by different steps ranging from the initial organization and control of the development project to the debugging of executing agents. All the steps are supported by tools and facilities provided by the toolkit.

  The CaseLP environment pays high attention to the MAS specification stage: the steps for developing a working prototype, starting from a vague natural language description, are clearly defined. For each step, tools and languages are provided to support the MAS developer in his/her task.

  The DESIRE modeling framework provides a design method for compositional multi-agent systems involving the analysis of the tasks performed by and between agents. Since the agent is a compositional entity, the same approach can be followed for specifying the MAS both at the macro-level (architecture of the overall MAS, interactions among agents) and at the micro-level (architecture of the single agent, control flow among its internal components).

  The IMPACT framework provides a step-by-step procedure for building an agent by "agentizing" legacy software. This procedure is supported by the IADE (IMPACT Agent Development Environment).

  The Zeus methodology consists of four stages: domain analysis, agent design, agent realization and runtime support. No software support is given for domain analysis and agent design, even if interesting suggestions are provided to face these tasks. The last two steps are supported by the Zeus Agent Generator and the Zeus Visualizer.

- **Ontological analysis.** In both AgentBuilder and Zeus an Ontology Manager (Ontology Editor, respectively) exists which provides tools for visualizing domain concepts and relationships by means of a graphical representation of them, and their codification for storage and subsequent retrieval.

  The possibility to define ontologies and to exploit the ontological knowledge for a better structuring of knowledge, enhancing knowledge sharing and reuse, has been studied also in the CaseLP environment [68]. It is not yet implemented.

  As far as DESIRE is concerned, an Ontology Server Agent has been developed in a specific application, but this was not yet added to the generic environment.

  The IMPACT Server provides a variety of services that are required by a society of agents as a whole, rather than by a single agent. Among them are a Type Server, a Thesaurus Server, and an Ontology/Translation Server which allow to organize and relate concepts in order to reformulate queries with different syntax but with identical semantics.

- **Specification.** By *specification language* we mean a language which allows to describe agents at a certain level of abstraction, or from a particular point of view. Specifications written in this language do not need to provide all the details for making the agent work, since they may abstract from some lower-level details, concentrating on higher-level features. The language used to fully describe the behavior of the agents is called, in this paper, *agent implementation language*. Clearly, an executable specification language can be seen as the agent implementation language. This is what happens with AgentBuilder, DESIRE and IMPACT: they have a unique declarative specification-implementation language, thus the two stages of high-level specification and detailed implementation are collapsed into a unique step.

  The CaseLP approach is different: the developer can choose the most suitable specification language from a set of supported ones, to describe both the macroscopic interconnections among agents and their microscopic internal functioning (see [43] for a description of the languages and [1] for a practical example of use of some of them). Hints for a semi-automatic translation from some of these languages into the prototype implementation language are also provided. The supported specification languages are not standard, but the integration of UML into CaseLP is currently under study, and some preliminary results in this direction have been obtained [19].

  Finally, Zeus suggests to adopt UML for the role modeling stage and provides a lot of examples of role model diagrams for agent roles which can be reused in different applications. No support is provided for the semi-automatic translation of UML specifications into working Zeus agents.

- **Verification.** Support for formal verification of properties is given by DESIRE and, partially, by CaseLP.

  DESIRE provides tools to verify and validate static properties of components such as consistency, correctness and completeness. Some first explorations have been done to extend verification to dynamic properties [24, 35].

  CaseLP agents may be specified in different languages, among which $\mathcal{E}_{hhf}$; this is a linear logic language amenable for formal verification, and a preliminary study on how to use this language for verifying properties of the system has been described in [5]. Up today verification has been performed on toy examples, thus it is not clear if the approach can be extended to more complex scenarios.

  In Zeus there are some deadlock prevention algorithms in the communication and scheduling mechanisms, but support for verification of different properties is not provided.

- **Prototyping.** The Agency Viewer of AgentBuilder is used to run a set of agents belonging to the same agency. It allows to follow the communication flow among agents, change the settings for the overall MAS and for the single agents, examine the current status of an agent.

  The CaseLP Visualizer, running under the ECLiPSE version of the toolkit, allows to follow the execution of the simulation both on-line and off-line. The execution stops when events to be monitored (which can be message exchange and/or state update) take place; all the details of the agent state evolution and of message exchange are described to the user.

  DESIRE has an automated prototype generator and provides an execution environment useful for process visualization.

  The IMPACT IADE allows the developer to view how the semantics of an agent program evolves with respect to the current state of the agent. A test dialog screen allows to test agent programs prior to execution, while the test execution screen allows to follow the execution of the program. It is also possible to monitor the agent's message box and to follow how concurrent actions are executed.

The Zeus toolkit provides facilities very similar to the AgentBuilder ones; the Visualizer provides tools useful for inspecting the social relations among agents, the status of the tasks a set of agents is performing, the internal status of each agents. It is also possible to dynamically change the current goals, resources and other features of an agent by means of a Control Tool.

- **Simulation.** The AgentBuilder Agency Viewer allows to save a Message Log File for later examination. No tools are provided for extracting information from this Log File which can be examined only manually.

  Also CaseLP allows to save messages for later inspection, but it provides both a graphical and a textual tool for extracting statistical information on them. For example, it is possible to associate a weight to the messages exchanged by the agents and to determine which is the minimum, maximum and average weight of messages per time slot, their standard deviation, and so on. This has proven useful for properly dimensioning communication channels in case of a limited bandwidth [1].

  DESIRE does not provide simulation facilities.

  The IMPACT Agent Log behaves like a bulletin board where agents can post messages resp. information. This log is used for record keeping, usage statistics (which are used in performance analysis/optimization) and process visualization.

  The Zeus Statistic Tool allows a user to collate various statistics about a society of agents on a per agent basis as well as on a community basis. The statistics collected include: the number of messages and their types sent by the agents over a time period; the number of messages sent out by the agents in coordinating different goal; the average loading of the agents; and, the coordination versus execution time ratio.

A.5 Implementation of agents and MAS

- **Agents implementation language.** The AgentBuilder agent programming language is called RADL. The RADL file is automatically produced by the AgentBuilder Toolkit when it is fed with the necessary information for building an agent. To execute an agent it is necessary to specify which external Java classes it uses. The Agent Program, composed by the RADL file and some external Java classes, is executed by the Reticular's run-time agent engine; the combination of the Agent Program and the agent engine produces an executable agent.

  The CaseLP agent programming language is called ProlAg: it is a standard Prolog enriched with primitives for safe state update and for communication. Two compilers exist for translating the ProlAg agent code into SICStus Prolog and into ECLiPSE.

  DESIRE agents are specified/implemented in a formal specification language for system design supporting conceptual design through detailed but implementation independent design. An implementation generator automatically translates specifications into code in a specific implementation environment.

  IMPACT agent programs are collections of action rules describing which actions an agent is permitted (respectively, forbidden, obliged, no more obliged) to take, or which actions it actually takes, according to the result of an access to the underlying software and/or to other permissions (obligations, ...) it has. An agent behavior is also determined by the integrity constrains it must satisfy.

Zeus agents are programmed by entering their characterizing features through the Agent Editor panel. The agent is ascribed the application-specific tasks it is capable of performing, the rules it can execute, its initial resources, the knowledge about other agents in the system, and so on. The agent code can be automatically generated when all these features have been defined.

- **Agents skeletons.** AgentBuilder and CaseLP do not provide agent skeletons for covering standard roles.

  A great work in this direction has been done by the DESIRE team: they have developed a number of reusable generic models for agents and tasks [10]. For example, generic agent models for autonomous agents with reactive, pro-active and social behavior have been developed, as well as for reflective agents and agents based on mentalistic notions. Generic task models for design, diagnosis based on causal and anti-causal knowledge, and process control are also provided.

  IMPACT does not provide standard skeletons, but it supports libraries: for actions, user-defined and other functions, action execution models, etc.

  In Zeus it is possible to equip an agent with a pre-built coordination protocol and interaction strategies; this can be seen as reusing a pre-built skeleton for defining the interaction behavior the agent will follow.

- **Physical environment models.** No model of the physical environment is explicitly supported by the MASDKs we are analyzing. Each agent has its own model of the physical environment, and it uses this model to decide how to act.

- **Agents and MAS definition GUI.** All the MASDKs provide a GUI for the definition of the MAS.

- **Utility agents.** No already-defined utility agents are provided by AgentBuilder, by CaseLP and by DESIRE.

  The IMPACT Server can be seen as an agent providing utility services such as Registration Service, for registering the services an agent provides, and Yellow Pages Service, which accesses the data structures created by the registration server for matching service request and service availability.

  The Zeus utility agents are the Agent Name Servers (ANS), the Facilitators, the Visualizers and the Database Proxies. The ANSs maintain a registry of known agents, enabling them to map agent identities to a logical network location. The Facilitators help in retrieving agents which provide the required service. The Visualizers are useful for execution monitoring and debugging. The Database Proxies provide the means of storing persistently the agent session information.

- **Debugging facilities.** All the toolkits which allow to follow the execution of the MAS session and to interactively change its parameters, provide in a certain sense the means for debugging the application. Also the ability to formally verify MAS properties can help in debugging the application under development.

A.6 Technical issues

- **Mobility.** No toolkit fully supports mobility. From a conceptual point of view, it should not be so difficult to build mobile agents with the toolkits implemented in Java, since it should be possible to integrate the code for mobile agents already developed in this language. But, from a practical point of view, some problems should arise, and maybe this is the reason why AgentBuilder, IMPACT and Zeus, all implemented in Java, do not fully support mobile agents. Some experiments have been done within the Zeus framework, but the answer to the question "Does Zeus support mobility" is still **no**. The IMPACT team is maybe at a better point on dealing with mobile objects, but they are still considering security and re-initialization issues regarding a mobile Java object thread, thus the support is not completely reliable yet.

- **Distribution.** AgentBuilder, DESIRE, IMPACT and Zeus allow to transparently distribute the agents in the system over a network.

  CaseLP allows the realization of a centralized prototype, where distribution is only simulated. D-CaseLP will support real agent distribution.

- **Concurrency.** Apart from CaseLP agents, which are activated sequentially by a round-robin scheduler, all the other MASDKs allow concurrent execution of agents.

- **Security.** The only toolkit which faces security issues, at least from a theoretical point of view, is IMPACT [4]. IMPACT focuses on the ability of agents to follow the

  – data security principle: for each data-object in an agent's state, there may be restrictions on which other agents may read, write, or manipulate that data; and the
  – action security principle: for each action in an agent's repertoire, there may be restrictions on which other agents may utilize those actions.

  The mechanisms to support security have not been implemented yet.

- **Real-time control.** It is not so easy to tell in advance if a toolkit will be suitable for real-time applications. We know that AgentBuilder and IMPACT have been used for developing such kind of applications (see [2] and [62]), but it is not enough to say that every time-critical application can be modeled with them. As far as CaseLP and Zeus are concerned, they are definitely not suitable for implementing real-time applications (at least, with the currently available versions), while the suitability of DESIRE for these applications is not clear.

- **Software integration.** The ability of integrating heterogeneous software is one of the major aims of IMPACT, whose formal theory is based on the concepts of code calls (calls to external software packages with a well defined API) and code call conditions (conditions on the objects returned by a code call). The actions an agent can perform are characterized by preconditions, add and delete lists which involve code call conditions. The semantics of agent programs provides, in a certain sense, a semantics to the way external software is accessed. IMPACT has been used to integrate

  As far as the other toolkits are concerned, they allow the integration of external software, but in a less formal way. AgentBuilder agents support JNI. Thus any external software module meeting JNI requirements can be used.

  CaseLP allows the integration of those packages versus which the implementation language (SICStus Prolog or ECLiPSE) provides an interface.

  DESIRE can integrate Java, Prolog, or any external program communicating through sockets, files or pipes.

  Zeus can integrate relational databases through JDBC, and other external packages have been successfully integrated by the users of the toolkit.

- **Other issues.**
  - AgentBuilder:
    * Platform required: AgentBuilder requires a Java Virtual Machine capable of executing Java 1.1.x. It has been tested on Solaris, Windows 95, Windows 98, Windows NT, Linux, and IRIX platforms.
    * Implementation language: Java.
    * System resources required: Hardware: 133 MHz Pentium machine recommended for Windows 95/98, Windows NT and LINUX platforms.
      RAM requirements: Windows 95/98, Windows NT, and LINUX: 16 MB (32 MB recommended); Solaris: 64 MB.
      Disk space requirements: Windows 95/98/NT: 11 MB; Linux: 15 MB; Solaris: 15 MB.
    * Price: AgentBuilder Pro: $ 4995.00; AgentBuilder Lite: starting from $ 495.00.
    * Availability of evaluation releases: an evaluation copy of AgentBuilder Pro is available.
  - CaseLP:
    * Platform required: CaseLP requires a SICStus Prolog or an ECLiPSE interpreter. The SICStus Prolog implementation has been tested on Linux, Windows 95/98 and Windows NT.
    * Implementation language: SICStus Prolog and ECLiPSE.
    * System resources required: CaseLP is extremely lightweight, it is sufficient that the system supports the Prolog interpreter.
    * Price: CaseLP is not a commercial toolkit.
    * Availability of evaluation releases: since a new distributed version is under development, no copies of the old centralized version are made available; the distributed version of CaseLP will be made available for free.
  - DESIRE:
    * Platform required: DESIRE is actively used on Windows 95/98, NT, 2000. It will work on any not too old Windows platform. On Linux platforms, it is actively used on Redhat, Suse. It will work on any not too exotic Linux variant. Finally, it is actively used on Solaris.
    * Implementation language: Prolog, xpce, Java, C.
    * System resources required: The GUI requires a screen resolution of at least 1024 by 768 pixels. Depending on the complexity of the application and on performance requirements a reasonably modern hardware configuration will be sufficient (i.e. say 500 MHz Pentium III with 64 MB internal memory).
    * Price: Not commercially available.
    * Availability of evaluation releases: For those who attend the annual course.
  - IMPACT:
    * Platform required: IMPACT runs on Sparc Ultra 1's (Solaris), Linux, Windows 95, 98, 2000 and NTWS 4.0.
    * Implementation language: The Agent Development (AgentDE), Roost, and Message Logger environments are currently implemented via Java 1.2.2. Moreover, many students in a recent (Fall '2000) course ran it from Java 1.3 without difficulty. It should be possible

to run it on any system where the Java 1.2.2 (or 1.3) virtual machine run-time code is installed.

  * System resources required: The Java archive (JAR) file (the IMPACT class library) is currently about 2 Megabytes. Compiled agent size depends on the program, but typically are about 50K.

    A quick test using the Windows NTWS 4.0 platform task monitor shows that running the following Java applications boosts the system resource load as follows:

    · 19.5 Megs AgentDE
    · 18.5 Megs Local Roost (with 12 agents)
    · 19.5 Megs Networked Roost (with 12 agents)

    These numbers will vary according to the VM implementation used (Palm tops have a much smaller VM) and should drop further as the IMPACT code will be optimized.
  * Price: To be negotiated, depends on the user (company, academia). Licensed by the Office of Technology Commercialization, University of Maryland, College Park.
  * Availability of evaluation releases: To be negotiated.

- Zeus:
  * Platform required: as Zeus uses the latest Swing GUI components it will run on any platform that has a JDK1.2 virtual machine installed. Each host machine should also be capable of TCP/IP communication, but there is no need for any middleware services to be installed. Zeus has been tested on Windows 95/98/NT4 and Solaris platforms.
  * Implementation language: Java.
  * System resources required: There is no need of high-end machine to run Zeus agents. Tests have been done with a Pentium 133 MHz and SPARC-5s.
  * Price: Zeus is Open Source software, and can be freely downloaded and distributed under the licensing conditions derived from the Mozilla Public License.
  * Availability of evaluation releases: all the Zeus releases are free.

A.7 Business issues

- **Organization.** We have already cited the organizations developing the various MASDKs in Section 3.
- **Training.**

  AgentBuilder provides training at additional cost for the toolkit.

  The DESIRE group organizes courses on the design of intelligent multi-agent systems, based on DESIRE. The last one was held from April 26 to May 2, 2000.

  IMPACT organized 3 Workshops for training.

  CaseLP and Zeus do not provide any training.
- **Documentation.** The documentation provided by AgentBuilder aims at helping a developer to perform her/his task; for this reason it is easily readable and allows to get an overview of the system, without facing too many technical details.

As far as CaseLP and DESIRE are concerned, the documentation is mainly constituted by scientific publications; it is not always easy to find some introductory material on the subject, an no manuals of use are available.

IMPACT and Zeus are described either by simple, introductory material, and by high-level scientific publications. Probably, the documentation for these toolkits is the more complete one.

- **Support.**

  AgentBuilder email support is free with purchase of the toolkit and telephone support is also available. We posted some general questions to the information provider and we got immediate answer.

  CaseLP is not yet available, thus no support is provided.

  DESIRE support depends on the availability of time of the staff. We posted some questions to the staff and we waited for three weeks to obtain a complete answer.

  There is no online/hotline support on a 24h basis for IMPACT.

  Finally, we personally experimented the efficiency and usefulness of the Zeus mailing list; an archive of mails is maintained where one can find an answer to many doubts; we also posted questions to the mailing list and got many interesting answers almost immediately.