Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 17–23
ISBN 978-83-60810-14-9
ISSN 1896-7094

Connecting Methodologies and Infrastructures in the Development of Agent Systems

Giacomo Cabri, *Member*, *IEEE*, Mariachiara Puviani, *Member*, *IEEE* and Raffaele Quitadamo
Dipartimento di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
Via Vignolese, 905 41100 Modena – Italy
Email: {giacomo.cabri, mariachiara.puviani, raffaele.quitadamo}@unimore.it

***Abstract*—In the building of agent systems developers can be supported by both appropriate methodologies and infrastructures, which guide them in the different phases of the development and provide useful abstractions.**

Nevertheless, we assist to a situation in which methodologies and infrastructures are not connected each other: the products of the analysis and design phases could not always be exploited in the implementation phase in a direct way, even if sometimes CASE-tools are present to help in translating methodologies' diagrams in infrastructures' code. This leads to a "gap" between methodologies and infrastructures that is likely to produce fragmented solutions and to make the application maintenance difficult.

In this paper we face this issue, proposing three directions to solve the problem. We do not want to propose a "new brand" methodology and infrastructure tightly connected, rather, we aim at reusing as much as possible what already exists, not only in abstract terms, but also in concrete "fragments" of methodologies; an appropriate meta-language that describes how a methodologies works would be useful to more easily map them onto the infrastructures, or even to "compose" a new methodologies. A further approach is based on an "intermediate" layer between methodologies and infrastructures, which provides a mapping between the involved entities.

I. Introduction

THE demand of effective paradigms for developing complex systems is significantly increasing in the last years [16]. Developers are required to model and manage complex scenarios, often physically distributed; in addition, such a management should be more and more autonomous, in order to take the correct decisions with less human intervention as possible. The development of such intelligent complex systems must be addressed appropriately, with effective models and tools.

Unfortunately, the current situation presents two different approaches that are likely to be in contrast: a *top-down* approach, which follows the traditional software engineering directions by providing *methodologies* that however often discard the implementation phase; a *bottom-up* approach, which meets concrete requirements by providing *infrastructures* that are likely to lack model foundations. This situation leads to a "gap" between methodologies and infrastructures, with the consequence of fragmented solutions in the development of agent systems.

The *software agent* paradigm is one of the most exploited to build complex systems [16]. On the one hand, agents exhibit the features of *proactivity* and *reactivity*, which lead to a high degree of autonomy and a certain degree of intelligence. On the other hand, their *sociality* feature enables the building of systems where agents are distributed and interact to carry out common tasks, as happens in Multi-Agent Systems (MAS).

In our work, we have analyzed several agent methodologies and agent infrastructures, first to confirm the presence of such a gap, then to study some solutions to fill this gap. The aim of this paper is to point out the existing gap and to propose some approaches to fill it. Due to the page limitation and the fact that our work is at its beginning, in this paper we sketch three solutions without giving too many details, providing readers with the "flavor" of the possible directions.

In rest of the paper we start presenting the evaluated methodologies and infrastructures, and discussing about the gap between them (Section II). Then, the core of the paper is represented by the proposal of three possible approaches as solutions of such a gap (Section III). Finally, Section IV concludes the paper.

II. The Gap between Methodologies and Infrastructures

In this section we briefly introduce the evaluated methodologies and infrastructures, and then point out the existing gap we are going to fill, sketching also some previous work. We remark that the aim of this paper is not to report comparisons: interested readers can refer to our previous work [7], [10], [24].

A. Methodologies

Among several methodologies for developing agent systems, we have chosen the most spread ones:

- ADELFE (Toolkit for Designing Software with Emergent Functionalities) [4] defines a methodology to develop applications in which self-organization makes the solution emerge from the interaction of its parts, and it guarantees that the software is developed according to the AMAS (Adaptive Multi-Agent System) theory [11]. It is dedicated to the design of systems that are complex, open and not well-specified. It is based on well-known tools and notations coming from the object-oriented software engineering: UML and RUP (Rational Unified Process);

- Gaia [26] was the first methodology proposed to guide the process of developing a MAS from analysis to design, starting from Requirements Statements. It guides developers to a well-defined design for a MAS. A MAS is seen as an organisation of individuals, each of which playing specific roles in that organisation, and interacting according to its role. In its first version it suffers from several limitations: it is suitable only for designing closed MAS, and the notions it uses are not suitable for dealing with the complexity of real-word. Trying to overcome these problems its official extension is Gaia v.2;
- PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology [12], for designing and developing Multi-agent societies, using the UML standard notation. It aims at using standards whenever it is possible and considers two different aspects of agents: (i) autonomous entities capable of pursuing an objective through autonomous decisions, action and social relationship and (ii) parts of a system. The modelling of requirements is based on use-case, and ontology has a central role in the social model. It aims at reusing existing portions of design code, using a pattern-based approach. Its design process is incremental and iterative;
- Prometheus [22] is a methodology for developing agent-oriented systems, and it aims at covering all the stages of software development [13]. It includes provision of "start-to-end" support; a hierarchical structuring mechanism that allows design to be performed at multiple levels of abstraction; and support for detailed design of the internals of intelligent agents. Prometheus also uses an iterative process over all its phases;
- SODA (Societies in Open and Distributed Agent spaces) [1], [20] is an agent-oriented methodology for the analysis and design of agent-based systems, focusing on inter-agent issues, like the engineering of agent societies and the environment for MAS. Its new version takes into account both the Agents and Artifacts (A&A) meta-model, and a mechanism to manage the complexity of system description;
- Tropos methodology is intended to support all phases of software development [5]; in Tropos, organizational architectural styles for cooperative, dynamic and distributed applications are defined to guide the design of the system architecture. It adopts Eric Yu's i* model which offers the notions of actor, goal, role and actor dependency as primitive concepts for modelling the applications. Tropos has two key features: the notions of agent, goal, plan and various other knowledge level concepts are fundamental primitives used uniformly throughout the software development process; a crucial role is assigned to requirements analysis and specification when the system-to-be is analysed with respect to its intended environment.

B. Infrastructures

With regard to the agent infrastructures, we have considered the following ones:

- CArtAgO (Common Artifact for Agent Open environment) [25], exploits the concept of *workplace*, an organisational layer on top of *workspaces*. A *workspace* is a set of *artifacts* and *agents*, as a *workplace* is the set of *roles* and *organisational rules* being in force in a workspace. *Roles* can be played by *agents* inside the workplace, and they may or may not give permissions to *agents* to use some *artifacts* or to execute some specific operations on selected *artifacts*.
- JACK [2] is a multi-agent platform based on the BDI model; it is written in Java and is commercial-oriented. Following the BDI model, JACK enables the definition of agents' plans that start from the agents' knowledge and lead to the achievement of the agents' goals.
- JADE [3] (Java Agent DEvelopment framework) implements a FIPA-compliant general-purpose multi-agent platform fully implemented in Java language, that provides an agent-oriented infrastructure. The main concept of Jade is the *agent*, which is implemented by the *AgentJ* class and is associated to an Agent State, a Scheduler, and a Message Queue. Another important concept id the *behaviour*, which defines the main features of an agent. Jade offers a set of graphical tools that supports the debugging and deployment phases;
- MARS (Mobile Agent Reactive Spaces) [9] is a coordination medium based on programmable Linda-like tuple spaces for Java-based mobile agents. MARS allows agents to read and write information in the form of tuples adopting a pattern-matching mechanism. Moreover, its behaviour can be programmed to suit environment or application requirements. It it has been designed to complement the functionality of already available agent systems, and it is not bound to any specific implementation;
- RoleX (Role eXtension) [6] implements an infrastructure to manage roles for agents. It has been implemented in the context of BRAIN (Behavioural Roles for Agent INteractions) framework which proposes an approach where the interactions among agents are based on the concept of role. A *role* is defined as a set of *actions* that an *agent* playing that *role* can perform to achieve its task, and a set of *events* that an *agent* is expected to manage in order to act as requested by the *role* itself;
- TOTA (Tuples On The Air) [17] is a coordination middleware based on tuple spaces for multi-agent coordination, in distributed computing scenarios. TOTA assumes the presence of a network of possibly mobile *nodes*, each running a *tuple space*: each agent is supported by a local middleware and has only a local (one-hop) perception of its environment. Nodes are connected only by short-range network links, each holding references to a (limited) set of neighbour nodes: so, the topology of the network,

as determined by the neighbourhood relations, may be highly dynamic;

- TuCSoN (Tuple Centers Spread Over Networks) [21] is a coordination medium based on Linda-like tuple spaces; it is focused on the the communication and coordination of distributed/concurrent independent agents. In TuCSoN,the *Agent Coordination Context* (ACC) works as a model for the *agent* environment, by describing the environment where an *agent* can interact. It also enables and rules the *interactions* between *agents* and the environment, by defining the space of admissible agent *interactions*. ACC has first to be negotiated by each *agent* with the MAS infrastructure, and then the *agent* specifies which *roles* to activate: if the *agent* request is compatible with the current organisation rules, a new ACC is created, configured according to the characteristics of the specifies *roles*, and is released to the *agent* for active playing inside the organisation.

C. The Gap

In our work we have studied the chance of integration between methodologies and infrastructures, evaluating whether and how the concepts considered by the methodologies can match with the concepts dealt with in the infrastructures. To this purpose, we exploited meta-models of both methodologies and infrastructures; in fact, the meta-model described the entities that represent the considered concepts of methodologies/infrastructures without looking at the different phases where they are involved, and allows to evaluate whether a matching exists and of which extent.

What emerges from our study is that there is not continuity between the methodologies and the infrastructures, presenting a gap between analysis and design on the one hand and development and implementation on the other hand. This gap concerns in particular the entities that represent the involved concepts. In fact, only few entities can be found in both methodologies and infrastructures with the same meaning, while others are present only in methodologies or infrastructures.

As an meaningful example better detailed in [8], consider the development of an application that simulates an auction. From a first analysis, the developer can design three roles, *bidder*, *auctioneer* and *seller* that can be implemented using an agent infrastructure. From a subsequent analysis, the design of a *broker* role emerges as needed in order to manage the real transactions from bidders and sellers. If there is a separation between the methodology and the infrastructure, i.e., there is not a connection between the “role” entity of the methodology and one or more entities of the infrastructure, the addition of a role could require the re-implementation of the application. Instead, if a connection is present, the modifications flow from the analysis/design to the implementation in a smooth way.

The lack of connections could be natural, since methodologies and infrastructures are likely to start from different needs; but it could lead to some problems; first, resulting in a fragmented development, but, more important, making maintenance very difficult: if a methodology entity has not a corresponding infrastructure entity, its change requires to find out how it was implemented.

As mentioned in the introduction, the reasons of such a gap are likely to derive from the different origins of methodologies and infrastructures: from the one hand, the traditional software engineering approaches follow a *top-down* direction; on the other hand, concrete requirements have called for implementation-oriented solutions providing platforms and frameworks to build applications. This gap could also reflect the twofold origin of the agent paradigm: artificial intelligence from the one hand, and distributed systems from the other hand.

Of course, there could be no need for exploiting infrastructure(s) not connected to a chosen methodology: there are different methodologies that have a natively compliant infrastructure. But we remark that the development of complex systems is likely to require the exploitation of different infrastructures, not always connected with the chosen methodology.

So, our aim is bridging agent methodologies and agent infrastructures in order to achieve a continuous support in the development of systems.

Previously, we have performed some work on specific issues.

As a first attempt, we try to map the methodologies’ entities with the infrastructures’ ones. To this purpose, first of all we have evaluated the entities common to the different infrastructures. This is useful to understand which the “core” entities are, which will deserve a support by the methodologies. This work is similar to build a common ontology that will semantically maps terms across methodologies and infrastructure, but the result will not cover all the necessary entities. So, we did not rely only on the *names* of the entities, but we spent an effort to map also entities with different names but the same meaning.

For instance, we have considered the PASSI methodology and the RoleX infrastructures, and we have produced a mapping based on their meta-models [8]. Molesini et al. have performed a similar attempt [18], considering one methodology (SODA) and three infrastructures (CArtAgO, TuCSoN and TOTA). As another example from our work, we have focused on the *role* entity, and we have evaluated how it is dealt with in methodologies and in infrastructures, in order to map the different approaches [23].

From these experiences, we have learned two main lessons. The former is that the exploitation of meta-models is very useful to understand the involved entities and their relationships; the latter is that a more global approach is needed.

III. Proposed Approaches

Starting from the previous considerations, in this section we sketch three possible approaches that can fill the gap. Our work has just begun, so we will provide readers with its first results, which require further study in the future.

There are a lot of different methodologies and infrastructures, so we consider impossible to map each methodology and each infrastructure one in the other; instead we aim at more

general solutions to help developer, especially those who have to integrate software agents in an existing application, but also those addressing new systems.

We also discarded the idea of creating a “new brand methodology” and a “new brand infrastructure” that can be used together, because they will be “yet another methodology” and “yet another infrastructure” to add in the context of the existing ones.

A. An Intermediate Layer

The first solution is proposed taking into account existing or legacy components or applications that must be integrated.

This approach consists of defining an intermediate layer, which provides entities that are connected to the methodologies on the one hand and to the infrastructures on the other hand (see Fig. 1).

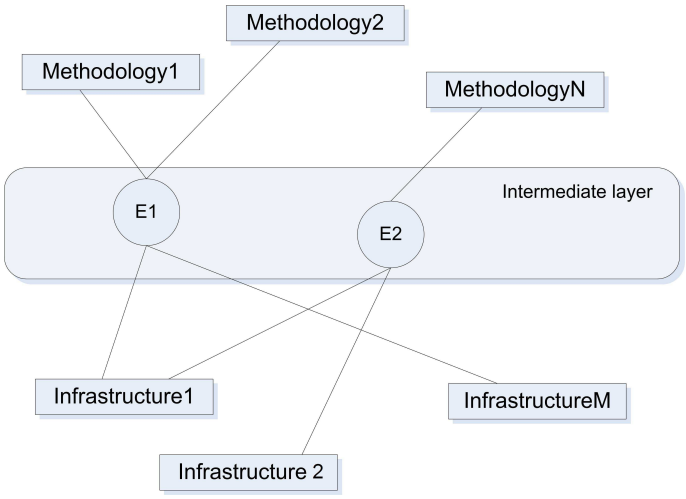

Fig. 1. An intermediate layer

From a particular point of view, this layer can be seen as an ontology, but we prefer to consider it more practically as a “common entities” layer, since it provides all the fundamental entities of the infrastructures chosen for the mapping; we have started from the most important ones, but then we are going to integrate all the others as well.

The entities that emerge from our study not only as common, but also as part of the foundation of the infrastructures, are:

- Agent
- Role
- Action
- Event

This layer can be exploited to map a methodology on every specific infrastructure. First of all, it can map entities starting from a direct mapping (one-to-one) and successively mapping the “composed” entities (one-to-n or n-to-one); during this work the studies of the mapping between methodologies’ entities and infrastructures’ ones previously presented can be very useful.

This new layer will permit to choose the preferred methodology and start to build the applications; then, when the infrastructure has to be chosen, the layer can be used to map the entities of the chosen methodology and the chosen infrastructures.

During this phase of translation, an interaction with the developer can be useful because not all the entities can be easily mapped, and sometimes, due to the chosen methodologies or infrastructures, the specifications of an entity by the developer can help the mapping. For example, MARS does not have the “event” concept, but has an “access event”, so the developer can be asked if the event is an access event or another one.

Of course this approach has some limitations, mainly because it is difficult to consider all the existing methodologies and infrastructures in a complete way; but it can be exploited for not forcing developers to choose a particular tool: it permits developers to be free in their choice, and to integrate existing applications.

B. SPEM

An interesting idea to fill the gap is to use the most important features of methodologies and infrastructures to map them together; but it is very difficult because of their different languages and their different phases. If we have the same language for all the methodologies (and infrastructures) it can be possible to make a mapping and even to “select” useful parts of the considered methodologies.

This study can be possible using SPEM (Software Process Engineering Metamodel. In Fig. 2 we briefly present the 1.1 SPEM notation, which has been used for our studies; for further information see also [15] and [19].

WorkProduct		Anything produced, consumed, or modified by a process.
WorkDefinition		Operation that describes the work performed in the process.
Activity		The main subclass of WorkDefinition, it describes a piece of work performed by one ProcessRole.
ProcessRole		Defines a performer for a set of WorkDefinitions in a process. It represents abstractly the "whoole process" or one of its components.
ProcessPackage		
Phase		A specialization of WorkDefinition. Its precondition defines the phase entry criteria and its goal defines the phase exit criteria.
Document		A WorkProduct
UMLModel		A WorkProduct

Fig. 2. The SPEM notation (v. 1.1)

With this “language” it is possible to describe processes and their components in term of pieces of process called “fragments”, using meta-models. This approach has been

used by the OMG specifications, in the context of "FIPA Methodology Technical Committee" project [19], to translate some well known methodologies, like Adelfe, Gaia, Tropos, Passi, etc. in a common language; and the work is still going on with other methodologies. This approach can be used also for infrastructures if there is a clear meta-model to be used for the translation, even if today it is used only for methodologies.

SPEM specification is structured as a UML profile, and provides a complete MOF (Meta Object Facility)-based meta-model. Each fragment that is created is composed of (not necessary all of them) [15]:

1) A portion of process defined with a SPEM diagram;
2) One or more deliverables (artifacts like AUML/UML diagrams, text documents, and so on);
3) Some preconditions which said that it is not possible to start the process specified in the fragment without the required input data or without verifying the required guard condition;
4) A list of concepts (related to the MAS meta-model) to be defined (designed) or refined during the specified process fragment;
5) Guideline that illustrates how to apply the fragment and best practices related to that;
6) A glossary of terms used in the fragment;
7) Composition guidelines—A description of the context/problem that is behind the methodology from which the specific fragment is extracted;
8) Aspects of fragment: textual description of specific issues;
9) Dependency relationships useful to assemble fragments.

The main idea of SPEM is that a software development process is a collaboration between abstract active entities (*process roles*) that perform operations (*activities*) on concrete entities (*work products*).

SPEM can be used to translate all the different methodologies and infrastructures, to help developers to choose which to use for their applications.

Today, the most used version of SPEM is 1.1, but the 2.0 version is just be implemented, and there is also a tool that can validate the created fragments and their integration.

C. A Composed Methodology

The last solution starts from the SPEM one and is the more radical, but it could be the most effective in the long term.

Given that the difficulties of the translation of all the methodologies and infrastructures is a very long work, a possible approach is to create a "composed methodology" that can be useful for developers, starting from SPEM fragments. This direction has been proposed also by the FIPA Methodology Technical Committee [14]. They have provided a repository of fragments to choose to compose an ad hoc methodology, adding the possibility to create new fragments.

To this purpose, we have selected some important fragments from the existing ones. In our study we have focused on (i) the common processes and entities of the methodologies and (ii) the entities that enable a connection with the main entities of the infrastructures. The composed methodology relies on different important fragments of ADELFE, PASSI, Gaia, Premetheus.

We have defined three phases: *Requirement* (Fig. 3), *Analysis* (Fig. 4) and *Design* (Fig. 5), while the *Implementation* phase is on going work.

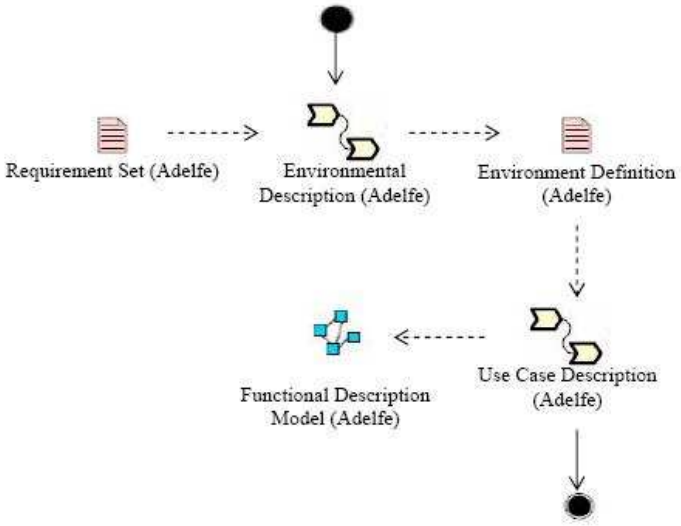

Fig. 3. Requirement phase

Some of these different fragments are well integrated, even if they come from different methodologies, while other fragments have been changed inside to match input and output of the different phases. This work has some difficulties because, as just said, entities with the same name but coming from different methodologies can not be used in the same way (e.g. the concept of "agent" in ADELFE is adaptive, in PASSI is FIPA-like and in Prometheus is a BDI one). The implementation and validation of this composed methodology is still going on, trying to adjust the different fragments.

There are some advantages in exploiting fragments of existing methodologies. First, it is not "yet another methodology", because it takes concrete parts of the other methodologies, not only ideas and concepts; then, the exploited fragments have been tested by developer for different years and in different scenarios; further, there is already related documentation, case studies, tools and practice; finally, some developers can find some parts of the methodology they are used to.

Even if this third solution is more radical, it does not completely discard the existing background, on the contrary it tries to reuse (parts of) existing solutions and experiences.

IV. Conclusions

In this paper we have addressed the gap existing between methodologies and infrastructures when developing agent systems.

To fill this gap, we have proposed three possible directions. The first relies on an *intermediate layer*, which provides some common entities that are mapped to both methodologies' and infrastructures' ones. The second exploits SPEM, to

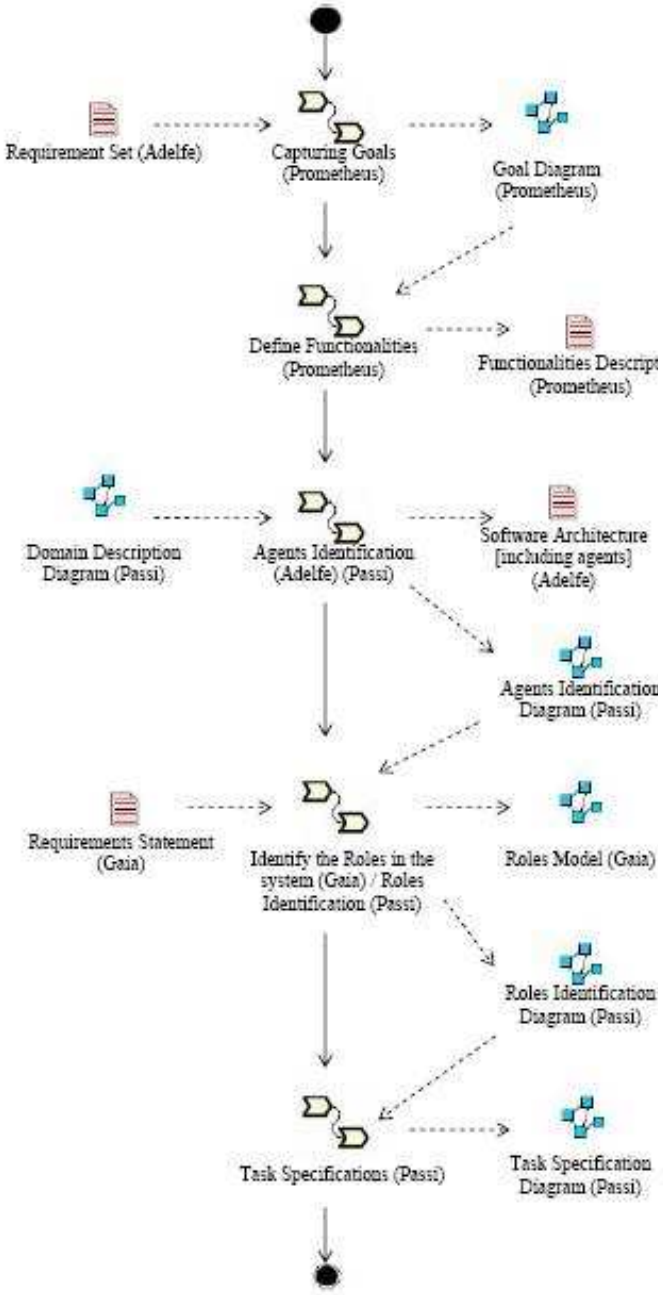

Fig. 4. Analysis phase

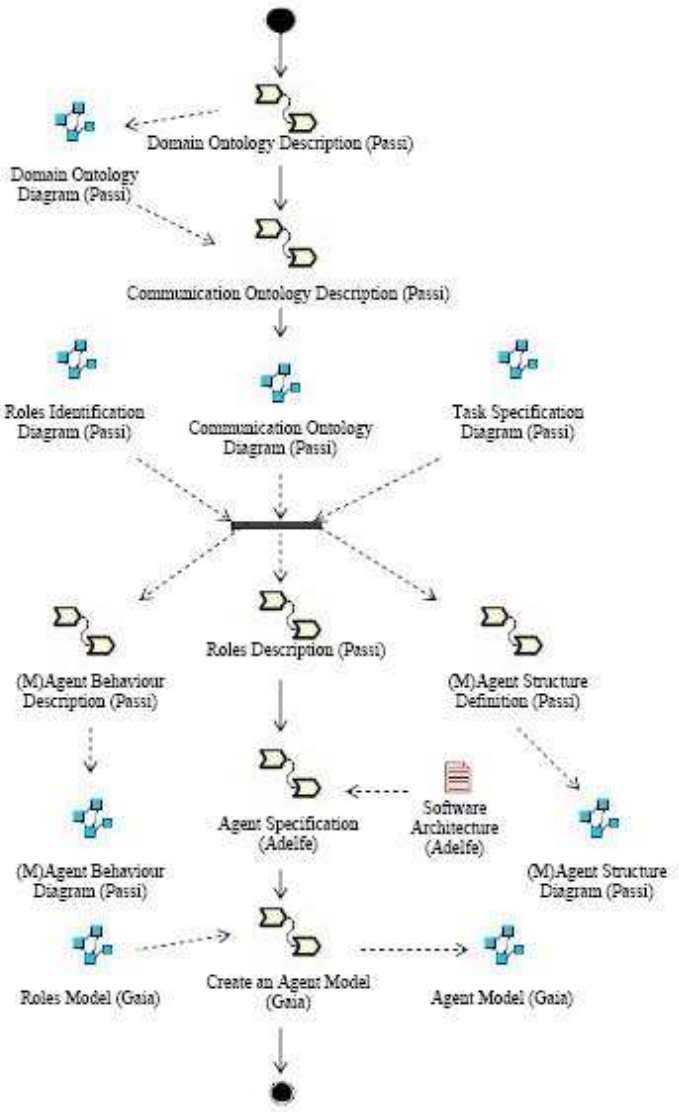

Fig. 5. Design phase

describe both methodologies and infrastructures by a common language, in order to make the mapping easier. Finally, the third approach proposes a new methodology, which is not "yet another methodology", instead it is a "composed methodology" of fragments extracted from existing methodologies. The proposed solutions range from the most conservative one (useful when existing components are to be integrated with a reduced effort) to the most innovative one (more useful in the development of new systems), even if we have spent an effort to not discard the existing knowledge in the field.

The approaches sketched in this paper deserve still a lot of future work.

With regard to the *intermediate layer*, we must integrate also other entities beside the ones mentioned in Section III-A, and we must propose and test the possible mappings.

SPEM has recently been updated to version 2.0, promising different improvements that must be evaluated, considering also all the legacy work on this meta-language.

Also the definition of a *composed methodology* can benefit from the new version of SPEM, in particular form the capability of checking the correctness and the completeness of fragment composition.

Finally, whichever the chosen direction, it must be appropriately tested, possibly with formal tools, but certainly with concrete case studies.

ACKNOWLEDGMENT

Work supported by the Italian MiUR in the frame of the PRIN project "MEnSA—Agent oriented methodologies: engineering of interactions and relationship with the infrastructures".

REFERENCES

[1] aliCE Research Group. SODA home page. http://soda.alice.unibo.it
[2] AOS Autonomous Decition-Making Software. Jack agent platform. http://www.agent-software.com/ 2008.
[3] F. Bellifemine. Developing multi-agent systems with jade. In *Proceedings of PAAM 99, London (UK)*, pages 97–108, 1999.
[4] C. Bernon, M.P. Gleizes, G. Picard, and P. Glize. The Adelfe Methodology For an Intranet System Design. In *Proc. of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS), Toronto, Canada*, 2002.
[5] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. A knowledge level software engineering methodology for agent oriented programming. In *Proceedings of the fifth international conference on Autonomous agents*, pages 648–655. ACM Press New York, NY, USA, 2001.
[6] G. Cabri, L. Ferrari, and L. Leonardi. Enabling mobile agents to dynamically assume roles. In *Proceedings of the ACM Symposium on Applied Computing, Melbourne (USA), March*, pages 56–60, 2003.
[7] G. Cabri, L. Leonardi, and M. Puviani. Service-Oriented Agent Methodologies. In *Proceedings of the Sixteenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2007*, 2007.
[8] G. Cabri, L. Leonardi, and M. Puviani. Methodologies and Infrastructures for Agent Society Simulation: Mapping PASSI and RoleX. In *Proceedings of the 2^{nd} International Symposium "Agent Based Modeling and Simulation", at the 19^{th} European Meeting on Cybernetics and Systems Research (EMCSR 2008), Wien, March 2008*, 2008.
[9] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: a programmable coordination architecture for mobile agents. *Internet Computing, IEEE*, 4(4):26–35, 2000.
[10] Giacomo Cabri, Letizia Leonardi, and Mariachiara Puviani. Methodologies for Designing Agent Societies. In *The Second Workshop on Engineering Complex Distributed Systems (ECDS 2008), Barcelona Spain, March 2008*, 03 2008.
[11] D. Capera, J.P. George, M.P. Gleizes, and P. Glize. The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *Proceedings of the Twelfth IEEE International Workshops onE nabling Technologies: Infrastructure for Collaborative Enterprises, 2003*, pages 383–388, 2003.
[12] M. Cossentino, L. Sabatucci, S. Sorace, and A. Chella. Patterns reuse in the PASSI methodology. In *ESAW-03*, pages 29–31. Springer, 2003.
[13] K. H. Dam and M. Winikoff. Comparing Agent-Oriented Methodologies. In *Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003)*, volume 14. Springer, 2003.
[14] FIPA. Methodology technical committee. http://www.fipa.org/activities/methodology.html,2003
[15] FIPA Methodology Technical Committee. FIPA-SPEM. http://www.pa.icar.cnr.it/%7ecossentino/FIPAmeth/metamodel.htm
[16] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
[17] F. Mamei and F. Zambonelli. Programming stigmergic coordination with the TOTA middleware. In *Proceedings of the 4^{th} international conference on Autonomous Agents and Multi-Agent Systems, New York (USA)*, pages 415–422, 2005.
[18] A. Molesini, A. Omicini, E. Denti, and A. Ricci. SODA: A roadmap to artefacts. *Engineering Societies in the Agents World VI*, 3963:49–62, 2005.
[19] Object Management Group. SPEM. http://www.omg.org/technology/documents/formal/spem.htm
[20] A. Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini and M. J. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 185–193. Springer, 2001.
[21] A. Omicini and F. Zambonelli. Coordination for internet application development. *Autonomous Anents and Multi-Agent Systems*, 2(3):251–269, 1999.
[22] L. Padgham, M. Winikoff, and A. Melbourne. *The Prometheus Methodology*, pages 217–234. Springer, 2004.
[23] M. Puviani, G. Cabri, and L. Leonardi. Agent Roles: from Methodologies to Infrastructures. In *Proceedings of the 2008 workshop on Role-Based Collaboration, at the 2008 International Symposium on Collaborative Technologies and Systems (CTS'08), Irvine, California, USA, May 2008*, 2008.
[24] Mariachiara Puviani, Giacomo Cabri, and Letizia Leonardi. Agent Roles: from Methodologies to Infrastructures. In *The 2008 workshop on Role-Based Collaboration, at the 2008 International Symposium on Collaborative Technologies and Systems (CTS'08), Irvine California, USA, May 2008*, 05 2008.
[25] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO: A framework for prototyping artifact-based environments in MAS. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for MultiAgent Systems*, volume 4389 of *LNAI*, pages 67–86. Springer, February 2007.
[26] F. Zambonelli, N.R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.