A Traceability Reference Model for Agent Oriented Development

Rosa Pinto¹, Jaelson Castro¹, Patrícia Tedesco¹, Maria J. Silva¹, Fernanda Alencar²,

¹ Centro de Informática, Universidade Federal de Pernambuco, Caixa Postal 7851, Recife, PE, Brazil
{rccp, jbc, pcart, mjs2}@cin.ufpe.br

² Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco (UFPE), Caixa Postal 7146, Recife, PE, Brazil
fmra@ufpe.br

Abstract. Software developers have used agents as a way to understand, model, and develop complex systems more naturally. Traceability has been recognized by many as an important strategy for developing and maintaining high quality software. Reference models for supporting traceability information have been proposed in the literature. However, none was specifically tailored for agent-oriented systems. In this light, our research aims to support traceability through the agent-oriented software lifecycle. In particular, this paper proposes an agent-oriented traceability reference model and shows how it can be used in the context of Tropos. An Autonomous Database administration system is used to demonstrate the applicability of our approach.

1. Introduction

Software developers are using agents as a new metaphor for understanding, modelling, and implementing systems which operate on complex, dynamic, open and unpredictable environments. An important contribution for this field is Tropos [Giorgetti et al, 2005], an approach used for modeling of Multi-Agent Systems (MAS). It borrows the abstractions and concepts from organizational and social disciplines to understand, model, reason, analyze and design agent software systems. In doing so, Tropos provide a more flexible, higher-level set of constructs to deal with a world operating more on social principles than on mechanistic rules.

Requirements engineering (RE) has argued that in order to successfully develop complex software systems, it is necessary to support the modelling process with traceability mechanisms and tools [Spanoudakis and Zisman, 2005]. In particular, the use of agent technology with its greater reliance on codified knowledge, its supposed flexibility, adaptively and autonomy [Yu, 2002], introduces new challenges for the support of requirements traceability. The capabilities of agent and the social aspect of agency should be considered. Our research aims to support traceability through the agent-oriented software development.

Requirement Traceability refers to the ability to ensure continued alignment between stakeholders’ requirements and various outputs of the system development
process. A requirements traceability process describes and follows the life of a requirement, in both a forward and backward direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases) [Gotel, 1996]. Software traceability is performed by generating, representing, recording and maintaining traceability relations (links) between software artefacts either manually or automatically.

This paper proposes an agent-oriented traceability reference model. It is constructed from a meta model based on the MOF (Meta Object Facility) [OMG, 2007]. It helps generating, representing, recording traceability relations. In order to demonstrate the applicability of our approach, we describe how the reference model has been applied to a real system.

Next, Section 2 presents the meta-models, reference model that are required to support traceability of requirements in AOSE. In Section 3, we apply Tropos to a case study and show how our traceability reference model can be used to support the requirement traceability process. Section 4 describes related works and finally Section 5 concludes the paper, pointing out research questions that are still open.

2. A Traceability Model for Agent-Oriented Development

Different uses and perspectives on traceability exit. So, there are wide variations on the format and content of traceability information across different system development efforts. Various classifications incorporating different types of traceability relation and reference models have been proposed to achieve a better understanding of the traceability process. In fact, a reference model is needed to facilitate the construction of a requirement traceability scheme [Toranzo, 2002]. To the best of our knowledge none address the specific issues raised by the agent oriented paradigm. Therefore, for instance, it is possible to trace how agents can reach agreement through negotiation on matters of not common interest.

Although the developers agree on the importance of trace between the artefacts, until now the meta-models for traceability do not take into account the agent view. In agent-oriented software development is important to know the capabilities of agents and trace how the agent interact with other agents in order to successful carry out the tasks that we delegate to them [Wooldrige, 2002]. We claim that a specific reference model aids is required. Next, our traceability meta-model and reference model for requirements traceability in AOSE is presented.

2.1 Traceability Meta Model

The aim of our meta-model is to construct a class diagram (like UML) to express a traceability reference model for agent oriented development. To do this, we define a set of traceability relations between software artefacts. The meta-model helps to clarify the identification of the new relations required for the agent paradigm.

In general, traceability relations are not properly defined, leading to inconsistent interpretations by different stakeholders during the software development. Hence, it is paramount to create standard, precise and rich semantics for traceability. We propose a
traceability meta-model (see Fig. 1) which aims to clarify some common relationships found in agent oriented development. The convention adopted is the following: the filled rectangles come from Toranzos’ meta-model [Toranzo, 2002], while the others are extensions. For the sake of space, only the relevant attributes are shown in Fig. 1.

The traceability metamodel describes the elements that can exist in our reference model. They can be of two types: (1) GeneralizableElement classifies and categorizes the instances of Class. It represents different parts, entities, and objects in software artefacts that are traceable which instance is represented by e. Class is a set of elements that share a common structure and a common behavior; (2) Relationship is a connection among elements. It can be of two types: Inheritance, and Association. Inheritance is a hierarchy where the superclass represents generalization and subclasses represent specializations, so using this type of relation (link) we can define how an element can be refined by other elements. Association denotes a semantic dependency among two elements, one been origin and the other destination. So, the direction of our traceability relation is explicitly specified. Multiplicity is an association property. Multiplicity denotes how many things may be connected across an instance of an association.

Fig. 1: Traceability Relations (Meta-Model).

Aggregation and Allocation are types of Association. Aggregation denotes a hierarchy. In traceability, it is used to identify how complex elements of a system can be broken down into components or how elements can be combined to form other elements. Likewise, Allocation is a simple association used to designate a portion of system resources for a particular task or operation. The reasoning behind the association is captured by a special kind of association called AssociationClass. It has class properties represented by AssociationClassProperties, i.e., it connects a set of elements and defines a set of features that belong to the relationship itself. It has two properties: Degree and Tree. Degree denotes the level of satisfaction of the association (e.g. High, Low), and Tree denotes how the association is satisfied (i.e. a tree with the same semantic of logical connectives AND and OR).

AssociationClass is used in traceability to increase the semantic of relations. The types of AssociationClass are Satisfiability, NeedsResource, Contribution,
Representation, Rationale, Ownership, Realization, and Subject-to. Hence, our traceability meta-model covers the following types of relations:

*Satisfiability*: the element \( e_1 \) satisfies an element \( e_2 \), if \( e_1 \) meets the expectations, needs, and desires of \( e_2 \); or \( e_1 \) complies with a condition represented by \( e_2 \).

*NeedsResource*: this specifies a dependence of resource (informational or physical) between the elements.

*Contribution*: represents associations between requirements artefacts and stakeholders that have contributed to the generation of the requirements.

*Representation*: \( e_1 \) represents \( e_2 \) if \( e_1 \) contains an alternative way of specifying \( e_2 \).

*Rationale*: this represents and maintains the rationale behind the creation and evolution of elements, and decisions about the system at different levels of detail.

*Ownership*: \( e_1 \) owns \( e_2 \) if \( e_2 \) is part of \( e_1 \)'s mental state.

*Realization*: \( e_1 \) realizes \( e_2 \) if \( e_1 \) executes \( e_2 \) or if \( e_2 \) constrains the execution of \( e_1 \)'s actions.

*Subject-to*: the element \( e_1 \) depends on element \( e_2 \), if the existence of \( e_1 \) relies on the existence of \( e_2 \), or if changes in \( e_2 \) have to be reflected in \( e_1 \).

*Ownership* and *Realization* relations were proposed to capture agent characteristics such as autonomy, intention, belief and goals.

Traceability information can be represented using traceability matrices, which relate requirements to stakeholders, between each other or design modules. The matrices’ representation of an association is a tuple whose structure varies according to the association type [Toranzo 2002]. For instance, the association resource has two components \( (<\text{DepDegree}; \text{Tree}>). \) The first component, \( \text{DepDegree} \), expressed the degree of dependency in a qualitative way (e.g., \( <\text{H: High, M: Medium or L: Low}> \) or \( <\text{S: Sufficient or P: Partial}> \) ) or quantitative way (values between 1 and 10). The second component, \( \text{Tree} \), represents the type of the logic tree which will relate the elements into decomposition. This component can assume the values \( <\text{A}> \) (read \( <\text{A}> \) as AND) or \( <\text{O}> \) (read \( <\text{O}> \) as OR).

The traceability meta-model defines the language in which traceability models can be created. In the sequel, our traceability reference model is defined.

### 2.2 Traceability Reference Model

The traceability reference model (Fig. 2) helps developers to find information considered to be traced as well as the nature of the relation between the recorded information. It helps requirements understanding, capture, track validation and verification. It can be used to refer to any activity that creates artefacts, including implementation.\(^1\)

Agents are entities with two important capabilities: autonomous actions and interaction with other agents. The interaction is not simply by exchanging data, but by engaging in analogues of kind of social activity (cooperation, negotiation and the like). So, when a developer works with the agent paradigm, besides the traditional INFORMATION concept – i.e data with means, it is necessary to deal with autonomous elements (AGENT), their capabilities (represented by PLAN) and their interaction (capturing by DEPENDENCE). INFORMATION, AGENT, PLAN and

---

\(^1\) The names of generalizable elements are written in CAPITAL letters, and instance of relationships are underlined.
DEPENDENCE are generalizable elements that represent the root of our traceability reference model.

AGENT is an entity that can be viewed as perceiving its environment through sensors or acting upon that environment through actuators. This element is further categorized into: WORLD AGENT, which is an agent with concrete, physical manifestations, such as a human individual or hardware/software. SOFTWARE AGENT is a computer system that is situated in some environment and capable of autonomous action in this environment in order to meet its design objectives. AGENTS possess BELIEFS and INTENTIONs.

BELIEF corresponds to the information an agent has about the environment it is located in. PLAN is a sequence of actions ordered-by the set of BELIEFS to accomplish a particular GOAL. INTENTION represents the commitment an AGENT has to perform certain actions achieved-by GOALs. GOAL describes situations that are desirable. It can be SOFTGOAL or HARDGOAL [Yu 2002]. It is refined-into sub-goals – represented by part-of relation. It is specialized into ORGANIZATIONAL GOAL and SYSTEM GOAL. Using these specializations it is possible to define how the software system fulfils the organizational goals, why it is necessary, what are the possible alternatives, what are the implications to the involved parts.

Fig. 2: Traceability Reference -Model.
AGENT plays ROLEs. ROLE is an abstract characterization of agent behaviour within a particular organizational context. It has permissions, resources, and one or more goals. ROLE realizes PLANs. An execution of the plan is constrained by RIGHTs and NORMs. RIGHT corresponds the things an AGENT is allowed to do when it plays a ROLE. NORMs are organization rules, which defines the general and global (supra-role) constraints requirements for the proper instantiation and execution of MAS that the actual organization will have to respect. It also expresses the information about how the organization is expected to work [Zambonelli et all, 2001]. DEPENDENCY indicates that one role depends, for some reason, on the other in order to do something, and there is a commitment between them. The depending role is called dependeer and the role who is depended upon the dependee. The object around which the dependency relationship centres is called the dependum. Dependence exists to attain some goal, execute some plan or deliver a resource. These four types of dependencies are required to indicate the nature of the freedom and control in the relationship between two agents regarding a dependum. They are described in [Yu, 2002].

AGENT controls or uses RESOURCEs. RESOURCE represents a physical or an informational entity that a particular agent wants and another agent can deliver. It can be EXTERNAL or ORGANIZATIONAL. ORGANIZATION is a set of ROLES – represented by part-of relation. It can be specialized in ORGANIZATION OF SOFTWARE AGENT. ORGANIZATION is regulated-by NORMs.

INFORMATION represents traceable elements – REQUIREMENT, CONSTRAINT, ACTIVITY, RESOURCE, ROLE, RIGHT, and NORM. The recursive depend-on relation of INFORMATION is inherited by its sub elements and the satisfy relation between it and CONSTRAINT element represents all sub elements related with INFORMATION that should obey the imposed constraints so that the system could be implemented. The responsibility relation between the INFORMATION and WORLD AGENT elements express the stakeholders that contribute or are responsible for the development of diagrams, programs, requirements, etc.

In the sequel a real case study is presented to demonstrate our approach.

3. Case study

The xaADB (eXternal Architecture for Autonomous Database administration) is a multi-agent system that aims at providing autonomy to Database Management Systems (DBMS). Acting as an electronic Database Administrator, the xaADB involves catalogued fault resolution, fail alerts, bad settings alerts and performance trouble resolution [Silva, at all, 2007].

In the sequence we describe how our traceability approach can be used in conjunction with the late requirement phases of Tropos [Giorgini, et all 2005]. We suppose the early requirements have been already defined. After constructing the reference model for this example, we will have recorded requirements, enabling the change impact assessment, as well as, the analysis of the degree of satisfaction of certain softgoals associated to the xaADB system. We could also trace the means-ends analysis used in the Tropos approach, as well as which and how softgoals contribute to satisfy a plan.
The process to followed includes three activities. First we collect all relevant information (section 3.1). Then we structure the information (section 3.2). Last but not least, we construct the traceability matrices.

3.1 Information Gathering

In late requirements phase we extend the conceptual model developed during early requirements to include the system-to-be, i.e., the xaADB. As late requirements analysis proceeds, xaADB is given additional responsibilities. DBA (Data Base Administrator) wishes from xaADB system that it fulfill several quality attributes including **High performance**, **Usable system**, **Trustful system**, **Supportable system**, **Secure system** softgoals that are instances of SOFTGOAL class (Fig. 3). Also, he wants **Autonomous fault resolution performed** goal (that is an instance of the GOAL class) to be satisfied. DBA is instances of the WORLD AGENTS class. So, when a change is required, the correspondent stakeholders can be questioned about possible doubts as well as conflicts to be resolved.

![Fig. 3: Actor Diagram for XAADB system](image)

The xaADB system depends on two other system actors: O.S. (Operational System) and DBMS (Database management system). These actors are world agents whose type is software system. Thus, O.S. and DBMS are instances of the WORLD AGENTS class. **O.S. resources and services provided** and **DBMS resources and services provided** are instances of EXTERNAL RESOURCE class.

After an analysis of the xaADB actor, we define the Strategic Rationale (SR) model (Fig. 4).

All the goals resulting from analysis of the **Autonomous fault resolution performed** hard goal (Fig. 4) correspond to system sub-goals, i.e. they are the state of affairs the software system aims to achieve through its functionalities. Thus, **Database control structure faults monitored and solve**, **Database physical objects faults monitored and solved**, **Connectivity faults monitored and solved**, **O.S. problems monitored and controlled**, **Database logical objects faults monitored and solved**, **Events notified**, **Information and knowledge managed** and **Users authenticate** goals (Fig. 4) are instances of the HARD GOAL class and are generated by AND-decomposition. AND-decomposition is represented by part-of relation. All the plans resulting from means-ends analysis of these goals correspond to operations that the system should be able to perform. Hence, **Monitor and suggest solutions to database logical objects faults** and **Monitor and solve database logical objects faults** plans are instances of the PLAN class which perform requirements. These plans can be decomposed into sub-plans by
AND/OR decompositions. These refined plans correspond to requirements. Thus, Fix tables and indexes fragmentation, Fix tables and indexes corruptions, Resize tables and indexes partitions, Alert security violations, Resize tables and indexes logical areas, Solve performance problems are instances of REQUIREMENTS class. All softgoals correspond non-functional requirements that the system must satisfy. Thus, High performance, Usable system, Trustful system, Supportable system and Secure system and Self learning soft goals (Fig. 4) are instances of the SOFT GOAL class.

The goals identified for the xaADB can be delegated to others agents that play roles as shown in Fig. 5. For example Information knowledge managed goal is delegated to Knowledge base manager role that is instance of ROLE class. xaADB is instance of ORGANIZATION class.

3.2 Information Structuring

Having gathered all the information needed to be traced, we can now structure it. We show the creation of the traceability matrix capturing the analysis of xaADB system depicted in Fig. 3. In this paper, we highlight the traceability of analysis of the xaADB system actor (Fig. 3).

We can define instances of the satisfy association between instances of the PLAN class and HARDGOAL class, calling it of Accomplish. Between PLAN and REQUIREMENTS class we can instantiate relations of the type satisfy, naming it...
Realized-by. An instance of the support associations between PLAN and SOFTGOAL class will be referred to Support or Oppose. The decomposition GOALS in SUB- GOALS can be mapped using part-of association and called Decomposed-in. xaADB is an organization of roles instanced by the aggregation Internal-role. The roles accomplish the goals of the xaADB. This is represented by the association named Perform.

We can also define the correspondent set of valid values like a tuple whose first component is degree, while second one represents the logical tree value as defined in section 2.1. The influence between PLANS and REQUIREMENTS can be evaluated as <H> (High), <M> (Medium) or <L> (Low), which corresponds to degree of the association Realized-by. The relation between HARDGOAL and PLAN is a means-end analysis and could be mapped using Accomplish association where the second component is <A> (read A as AND) or <O> (read O as OR) which compose the logical tree. Association between PLANS and SOFTGOALS are similar to relationship used in the NFR framework (+++, +, --, -) [Yu 2002]. It can be mapped in the following away: the positive and negative influence can be mapped to the Support and Oppose association respectively. The softgoal is supported or opposed in sufficient (S) or partial (P) ways which are degree values of the association. The second component is <A> (read A as AND) for all the instances. These set of values increases the understanding of stakeholders about the semantic of trace.

3.3 Defining the Traceability Matrixes

Having structured all the gathered information, we can now create traceability matrixes. For example, the following traceability matrix may be proposed: Accomplish, Decomposed-in, Support, and Realized-by (Table 1, 2, 3 and 4). The symbol “←” indicates the direction of the relation.

The DBA hope that the main goal - “Autonomous fault resolution performed”- is accomplished (Fig. 3). The SR model (Fig. 4) provides a representational structure for expressing the rationales behind xaADB system to achieve the Autonomous fault resolution performed goal. This goal can be refined in sub-goals using AND-decomposition. This can be represented by the Decomposed-in matrix (Table 1). According to this table the “Autonomous fault resolution performed” goal involves (i.e. it is AND decomposed into) all the sub goals presented.

Table 1. Decomposed-in Matrix

<table>
<thead>
<tr>
<th>Decomposed-in &lt;&lt;satisfy&gt;&gt;</th>
<th>HG1 – Autonomous fault resolution performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>HG2 – Database control structure faults monitored and solved</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>HG3 – Database physical object faults monitored and solved</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>HG4 – Database logical object faults monitored and solved</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>HG5 – Connectivity faults monitored and solved</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>HG6 – OS problems monitored and solved</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>HG7 – Event notified</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>HG8 – Information and knowledge managed</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>HG9 – Users authenticated</td>
<td>&lt;H, A&gt;</td>
</tr>
</tbody>
</table>

The plans that accomplish sub-goals represented in Table 1 can be analyzed by means-end link. The components that participated in the means-ends relationship are captured by the Accomplish matrix. For example Table 2 indicates the “Database logical object faults monitored and solved” (HG4) goal can be accomplished by two alternatives plans PL5 or PL6.
We can discover how a plan is decomposed using the Realized-by matrix. For example, Table 3 shows that plan 5 (PL5 - Monitor and solve database logical objects faults) is decomposed (Realized-by) into four functional requirements (FR1, FR2, FR3 and FR6).

<table>
<thead>
<tr>
<th>Accomplish &lt;satisfy&gt;&gt;</th>
<th>PL5 - Monitor and solve database logical objects faults</th>
<th>PL6 - Monitor and suggest database logical objects faults</th>
<th>PL13 – Get configuration</th>
<th>PL14 – Authenticate users</th>
</tr>
</thead>
<tbody>
<tr>
<td>HG4 – Database logical object faults monitored and solved</td>
<td>&lt;H, O&gt;</td>
<td>&lt;H, O&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HG8 – Information and knowledge managed</td>
<td></td>
<td></td>
<td>&lt;H, O&gt;</td>
<td></td>
</tr>
<tr>
<td>HG9 – Users authenticated</td>
<td></td>
<td>&lt;H, O&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The sub-goals can be attributed to roles that compose the xaADB organization. For example Table 4 shows that DBMS manager, OS manager, Connectivity manager, Report manager and Knowledge base manager are roles required by the organization.

<table>
<thead>
<tr>
<th>Internal-role &lt;&lt;part-of&gt;&gt;</th>
<th>xaADB</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 – DBMS manager</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>R2 – OS manager</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>R3 – Connectivity manager</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>R4 – Report manager</td>
<td>&lt;H, A&gt;</td>
</tr>
<tr>
<td>R5 – Knowledge base manager</td>
<td>&lt;H, A&gt;</td>
</tr>
</tbody>
</table>

The goals accomplished by roles are shown in Table 5. Following our example, the R5- Knowledge base manager role performs the HG8 – Information and knowledge managed goal.

<table>
<thead>
<tr>
<th>Perform &lt;&lt;satisfy&gt;&gt;</th>
<th>HG7 – Event notified</th>
<th>HG8 – Information and knowledge managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 – DBMS manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2 – OS manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3 – Connectivity manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4 – Report manager</td>
<td>&lt;H, A&gt;</td>
<td></td>
</tr>
<tr>
<td>R5 – Knowledge base manager</td>
<td></td>
<td>&lt;H, A&gt;</td>
</tr>
</tbody>
</table>

Hence, by using our reference model, it is possible to trace which requirements perform a plan that accomplishes a goal, and which role (played any agent) is responsible for a requirement.

4. Related Work

Ramesh [Ramesh and Jarke, 2001] introduces the idea of reference model to trace requirements. This model enables the user to extract and adapt their elements to construct his/her own requirement model for a specific project. Gotel in [Gotel, 1996] presents one result of the empiric work related with the identification and understanding.
of the problems and practices associated with the requirements traceability. Odell proposed meta-modeling constructs to model agents, their roles and their groups. This meta-modeling provides the basic foundational elements required in multi-agent systems to foster dynamic group formation and operation [Odell, Nodine and Levy, 2005]. However, these works do not consider agent traceability explicitly.

Toranzo in [Toranzo, 2002] introduces a set of types of relationships and structures the traceable information in levels (external, organizational and management) to improve the semantic of requirement traceability. Our work extends Toranzo’s work to take in consideration the agent oriented paradigm.

Our proposal outlines a reference model to help the software engineer to find and structure the necessary information to perform traceability when developing agent-oriented systems. By using this reference model, we can capture the whole “history” of a requirement from the goals that originate requirements up to their implementation.

5. Conclusions and further work

Requirement traceability has been recognized by many as an important pre-requisite for developing and maintaining high quality software. In this work, we outline a reference model that can be used in conjunction with the any agent-oriented development approach to address requirements traceability.

As an example, we showed how the requirements phase of a real system developed using the TROPOS approach could be traced. The benefits of requirements traceability are manifold: software quality can be improved since we can check if all stakeholders’ requirements are addressed by the system. Similarly, an impact analysis can also be performed before the implementation of a requested change. This is possible because the requirements impacted by the change can be detected since the links between these requirements and other system’s artifacts, such as design and implementation ones, can be traced. Hence, estimating change and effort become more accurate and consequently we can minimize the time and cost of software maintenance.

Our reference model is still evolving in the sense that we are working on adding more refined MAS aspects to it (e.g. negotiation, coordination). We plan to use it in the context of the TROPOS approach. In particular, we need to support architectural, detailed design and implementation phases. A requirement traceability process is also under development. A proper tool for supporting it, is another topic that needs to be addressed. Further case studies are also paramount to check the viability of the approach in complex situations.

References


on Agent Oriented Software Engineering (AOSE’04). New York City, New York – July


