Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 49–62. http://www.scpe.org
ISSN 1895-1767

A TOP DOWN APPROACH FOR DESCRIBING THE ACQUAINTACE ORGANISATION OF MULTIAGENT SYSTEMS*

JOAQUÍN PEÑA†, RAFAEL CORCHUELO† AND ANTONIO RUIZ-CORTÉS†

Abstract.
When the protocol of a complex Multi-Agent System (MAS) needs to be developed, the *top-down approach* emphasises to start with abstract descriptions that should be refined incrementally until we achieve the detail level necessary to implement it. Unfortunately, there exist a semantic gap in interaction protocol methodologies because most of them first, identify which tasks has to be performed, and then use low level description such as sequences of messages to detail them.

In this paper, we propose an approach to bridge this gap proposing a set of techniques that are integrated in a methodology called MaCMAS (Methodology for Analysing Complex Multiagent Systems). We model MAS protocols using several abstract views of the tasks to be performed, and provide a systematic method to reach message sequences descriptions from task descriptions. These tasks are represented by means of interactions that shall be refined systematically into lower-level interactions with the techniques proposed in this paper (simpler interactions are easier to describe and implement using message passing.) Unfortunately, deadlocks may appear due to protocol design mistakes or due to the refinement process that we present. Thus, we also propose an algorithm to ensure that protocols are deadlock free.

Key words. Top-down approach, agent protocol descriptions, interaction refinements, and deadlock detection.

1. Introduction.

Agent-Oriented Software Engineering (AOSE) is paving the way for a new paradigm in the Software Engineering field. This is the reason why a large amount of research papers on this topic are appearing in the literature. One of the main research lines in AOSE arena is devoted to developing methodologies for modelling interaction protocols (hereafter protocols) between agents.

1.1. Motivation.

When a large system is modeled, its complexity becomes a critical factor that has to be managed properly to achieve clear, readable, reusable, and correct specifications [8, 24, 30]. In the literature, there exist various techniques to palliate this problem. The most important are the *top down* and the *bottom up* approachs. The top down approach, which is the focus of this paper, first tries to describe software from a high level of abstraction, and then goes into further details until they are enough for implementing the system [32].

When the protocol of a large MAS has to be developed, it is desirable to start with an abstract description that can be refined incrementally according to the *top down approach*. In our opinion, there exist two drawbacks in most existing methodologies:

- On the one hand, most of them provide top-down approaches for modeling and developing these systems. These methodologies, general or protocol-centric, agree on using abstract messages and sequence diagrams to describe protocols [3, 19, 37, 15]. Although these messages represent a high level view of a protocol, which shall be refined later, the tasks that are performed are formulated as a set of messages. This representation implies that the abstraction level falls dramatically since a task that is done by more than two agents requires several messages to be represented. This occurs even if we consider a task between two agents. For instance, an information request between two agents must be represented with two messages at least (one to ask, and another to reply). This introduces a semantic gap between tasks to be performed identified at requirements and its internal design since it is difficult to identify the tasks represented in a sequence of messages. This representation becomes an important problem regarding readability and manageability of large MAS.
- On the other hand, abstractions of protocols (interactions) that allow designers to encapsulate pieces of a protocol that is executed by an arbitrary number of agents has been proved adequate in this context [3, 4, 19, 20, 38]. Unfortunately, interactions are generally used to hide unnecessary details about some views of the protocol. This improves readability and promotes reusability of protocol patterns, but they are not used for bridging the existing semantic gap between tasks and its representation.

1.2. Contributions.

In our proposal, we present a different approach to use interactions, which is based on the ideas presented in [4, 26, 38]. This approach is integrated on a methodology called MaCMAS that covers top-down and bottom-up. The top down software process is sketched in Figure 1.1. As shown, our goal is to

*This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472) and TIC200302737C0201
†University of Seville, {joaquinp, corchu}@lsi.us.es, aruiz@us.es

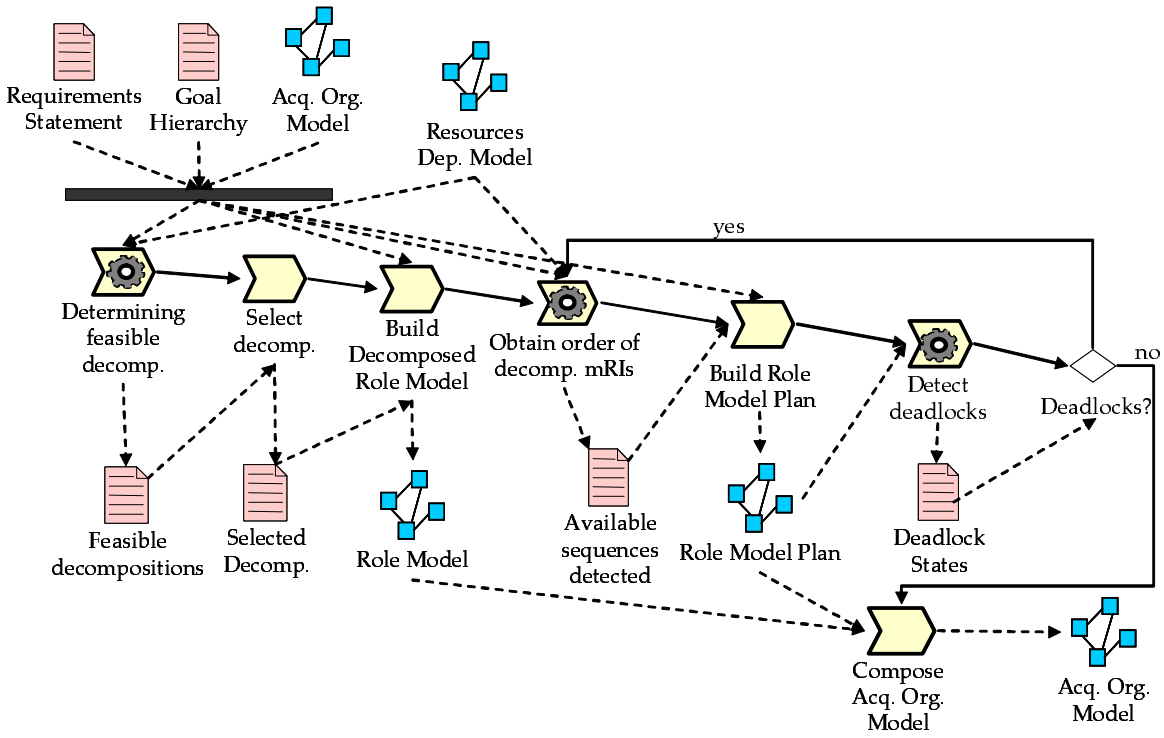

Fig. 1.1. *Software process of refinements.*

bridge this gap using interaction abstractions to model the tasks to be performed, and Finite State Automata (FSA), represented using UML 2.0, to model how to sequence them. Afterwards, we refine them systematically into simpler ones iteratively. This decreases the level of abstraction so that the interaction we obtain are simpler. Thus, they are described internally as message sequences easily, e.g. using AUML [3].

We have used a protocol abstraction called multi-role interaction (mRI), which was first proposed in [25]. An mRI is an abstraction that encapsulates a set of messages between an arbitrary number of agent roles. Furthermore, the refinement process we use is based on the ideas presented in [10] since the interaction we use is similar to such used in this work. The refinement process relies on analysing the knowledge used by each role in an mRI and using this information to transform an mRI into several simpler mRIs automatically. An mRI is simpler when both the number of participant roles and the computation made by it decreases. The main advantages of refining mRIs are the followings:

- First, its internal description is easier since the computation to perform in the obtained tasks are simpler.
- Second, it is easier to implement interactions with a low number of participant roles [12, page 206] [2, 33, 21, 35].
- Finally, mRIs are critical deadlock free regions and they are mutually exclusive. Thus, if the number of participant roles increases, the concurrency grain decreases, what is clearly not desirable [34].

The main drawback of such refinements is that they may lead to deadlocks. In this paper, we also propose a technique to detect if a refinement may introduce deadlocks (see Figure 1.1); it also characterises them by means of regular expressions that help finding the refinements that are not adequate in a given context. It is based on analysing the FSA that represents the protocol of a role model and some previous work on deadlock detection in the context of client/server interactions [5, 14, 36]. It improves on other results in that it can be automated because it does not require any knowledge about the implied, intuitive semantics of the interactions as other approaches.

This paper is organised as follows: in Section 2 we present the related work about protocol modeling in MAS and about interaction refinements; in Section 3, we summarise the methodology where this work is integrated; in Section 4 we present the example that we use to illustrate our approach; in Section 5 we present our ideas on

protocol modeling and we show the refinement techniques applicable; in Section 6 we present our approach to the automatic deadlock detection process; Section 7, we show our main conclusions. Finally, an appendix that shows an implementation of the case study using IP.

2. Related work. In this section we cover the related work on protocol modelling and on refinements.

2.1. Protocol Modeling. As we showed in the previous section, we think that most approaches model protocols at low level of abstraction since they require the designer to model complex cooperations as message-based protocols. This issue has been identified in the Gaia Methodology [38], and also in the work of Caire *et. al.* [4], where the protocol description process starts with a high level view based on describing tasks as complex communication primitives (hereafter interactions). We think that the ideas presented in both papers are adequate for this kind of systems where interactions are more important than in object-oriented programming.

On the one hand, in the Gaia methodology, protocols are modeled using abstract textual templates. Each template represents an interaction or task to be performed between an arbitrary number of participants. Furthermore, interactions are decorated with the knowledge they process and the permissions each role has, their purpose, their inputs and outputs, and so on.

On the other hand, in [4], the authors propose a methodology in which the first protocol view is a static view of the interactions in a system. Each interaction is used by a set of agent roles and they are decorated with the knowledge each role uses/supplies. Later, the internals of these interactions are described using AUML [3].

As the methodologies cited above, we also use interactions to deal with the first stage of protocol modeling. Furthermore, we also represent a static view of interactions and the knowledge that each role consumes and produces in each of them. Unfortunately, both methodologies do not provide an automatic method for refining complex interactions into smaller interactions that are closer to the implementation level. In this paper, we elaborate on such a method.

Furthermore, in methodologies that use sequence diagrams to model protocols, it has been also identified the need for advanced multi-role interactions that encapsulate a piece of protocol. Unfortunately, in most of them these interactions are used to define reusable patterns of interaction or for hiding details in some complex views. Several examples of such use of interactions can be found in the literature: For instance, AUML nested protocols [3] or micro-protocols [19]. These approaches provide the user with a set of tools to model complex co-operations; however, most designers use message–based descriptions.

2.2. Refinements. The need for such protocol primitives has also been identified in other areas such as distributed systems [11, 7, 23]. In this context such interactions have been studied for long, and there exist advanced techniques to refine them (synchrony loosening refinements [10]). Unfortunately, these refinements can lead to deadlock. Although the theory of refinements has reached a rather elaborate state in other contexts, cf. [1], there are not many results on interaction refinements or the characterisation of their anomalies. The main reason is that classical refinements are context-free, whereas interaction refinements are context–sensitive. Thus, the main problem is the establishment of their monotonicity properties [10], whereby their application to subparts of a protocol preserves the correctness of the whole protocol with respect the set of valid synchronisation patterns it describes.

The state–of–the–art technique that focus on design time properties was presented in [12]. It is based on designing a formal proof system (*cooperating proof*) that allows to prove a sufficient condition for monotonicity that ensures that a system composed of interactions is deadlock free. It is based on analysing linked interactions, i.e., interactions that need to be executed in sequence, to avoid deadlocks, which was previously suggested in [9, 18]. Unfortunately, this technique is quite difficult to apply in practice because it requires in-depth knowledge of the implied, intuitive meaning of the interactions, and no automatic proof rules were designed for showing the satisfaction of the sufficient condition.

Our proposal can detect if a refinement may lead to a deadlock situation automatically, and also characterises the set of traces that lead to it by means of regular expressions. It is based on FSA analysis used by many researchers in the context of client/server deadlock detection of interaction models [5, 14, 36].

3. Engineering MultiAgent Systems with MaCMAS. MaCMAs[1] is a methodology for engineering complex multiagent systems that is integrated with several research fields, i.e. autonomic computing [31], software product lines [27, 28] and evolving systems [29].

[1] see `james.eii.us.es/MaCMAS/` for further details on MaCMAS

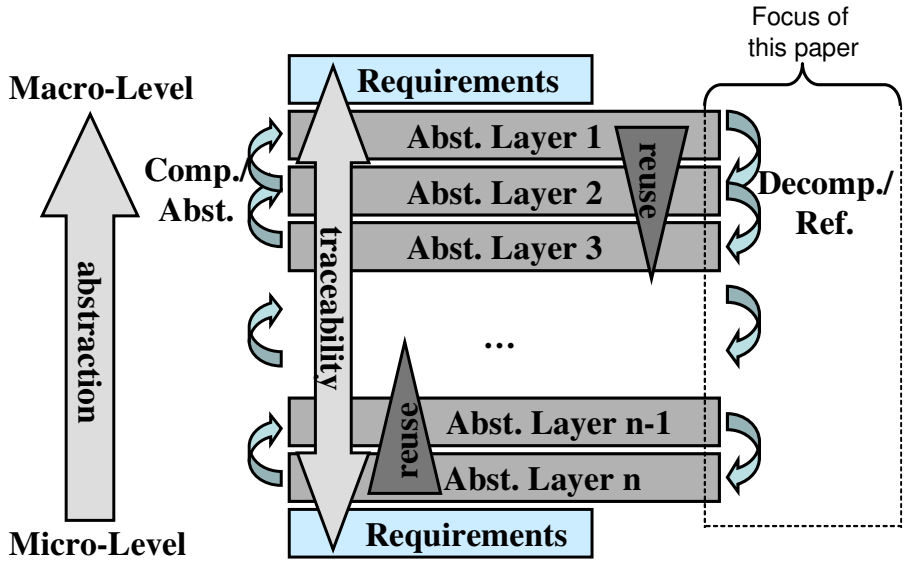

FIG. 3.1. *Process Overview*

MaCMAS covers carefully the five principles to deal with complexity in software engineering where top-down and bottom-up are of high importance [16, 17, 30]: abstraction, decomposition/refinements, composition/abstraction, automation and reuse.

In Figure 3.1, we show an overview of the main concepts applied in MaCMAS from the software process point of view. As shown, models of the system are structured into a set of abstraction layers. Top models are the most abstract while bottom models are the most refined models. MaCMAS provides also a set of vertical and horizontal transformations. Vertical transformations are applied to split models or to compose models, and horizontal transformations are used to refine and abstract models in order to cover bottom-up and top-down software processes.

As shown, for covering the rest of principles, traceability between models at different abstraction layers and reuse of models and their abstractions/refinements is also provided.

In MaCMAS, two kind of refinements are proposed. One that is base on analyzing information on requirement documents, concretely system goals hierarchies, to recommend the user of the CASE tool which models can be refined and which is the best decomposition recommended. The other refinement, which is the focus of this paper, is based on analyzing the dependencies between the elements in a model to recommend a refinement.

3.1. Models. In other to engineer MASs, MaCMAS provides a rich set of UML2.0-based models that can be summarized in:

a) Static Acquaintance Organization View: This shows the static interaction relationships between roles in the system and the knowledge processed by them. It comprises the following UML models:

Role Models: shows an acquaintance sub-organization as a set of roles collaborating by means of several mRIs. As mRIs allow abstract representation of interactions, we can use these models at whatever level of abstraction we desire. We use role models to represent autonomous and autonomic properties of the system at the level of abstraction we need.

Parameterized Role Models : A parameterised role model permits us to represent reusable collaboration patterns parameterising some of their elements.

Resources dependency model: A resources dependency model provides means for documenting the dependencies between knowledge entities and services provided by roles in the context of an mRI and for documenting the dependencies between the knowledge of mRIs.

Relating role models model: As a result of using decomposition and composition and of instantiating parameterised role models, we usually manage role models that are obtained from others. This model show the relationships between several role models.

Ontology: shows the ontology shared by roles in a role model. It is used to add semantics to the knowledge owned and exchanged by roles.

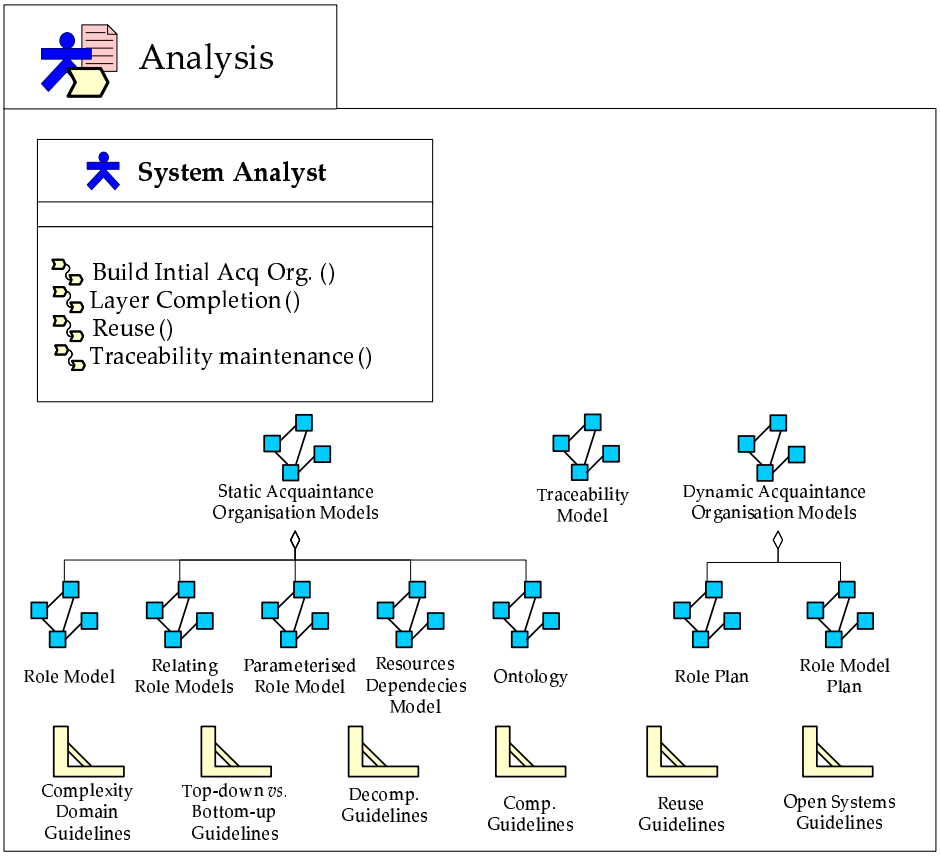

FIG. 3.2. *Acquaintance analysis discipline*

b) Behavior of Acquaintance Organization View: The behavioral aspect of an organization shows the sequencing of mRIs in a particular role model. It is represented by two equivalent models:

Plan of a role: separately represents the plan of each role in a role model showing how the mRIs of the role sequence. It is represented using UML 2.0 ProtocolStateMachines [22, p. 422]. It is used to focus on a certain role, while ignoring others.

Plan of a role model: represents the order of mRIs in a role model with a centralized description. It is represented using UML 2.0 StateMachines [22, p. 446]. It is used to facilitate easy understanding of the whole behavior of a sub-organization.

c) Traceability view: This model shows how models in different abstraction layers relate. It shows how mRIs are abstracted, composed or decomposed by means of *classification*, *aggregation*, *generalization* or *redefinition*. Notice that we usually show only the relations between interactions because they are the focus of modeling, but all the elements that compose an mRI can also be related. Finally, since an mRI presents a direct correlation with system goals, traceability models clearly show how a certain requirement system goal is refined and materialized. This is main what helps us to bridge the gap between requirements and design.

For the purpose of this paper, we only need to detail role models, role model plans, which are shown in the following sections.

4. The Example. The example we use hereafter is a debit–card system. This problem can be viewed as one of the basic coordination patterns in the agent e-commerce world, and it involves three different agent roles (hereafter roles): a point of sales role (PS) which interacts with the user, a customer account manager role(CA), and a merchant account manager role (MA). When a customer uses his or her debit card, the agent playing role PS agrees with a CA agent and merchant account agent on performing a sequence of tasks to transfer the money from the customer account to the merchant account, which shall also be charged the costs of the transaction. If

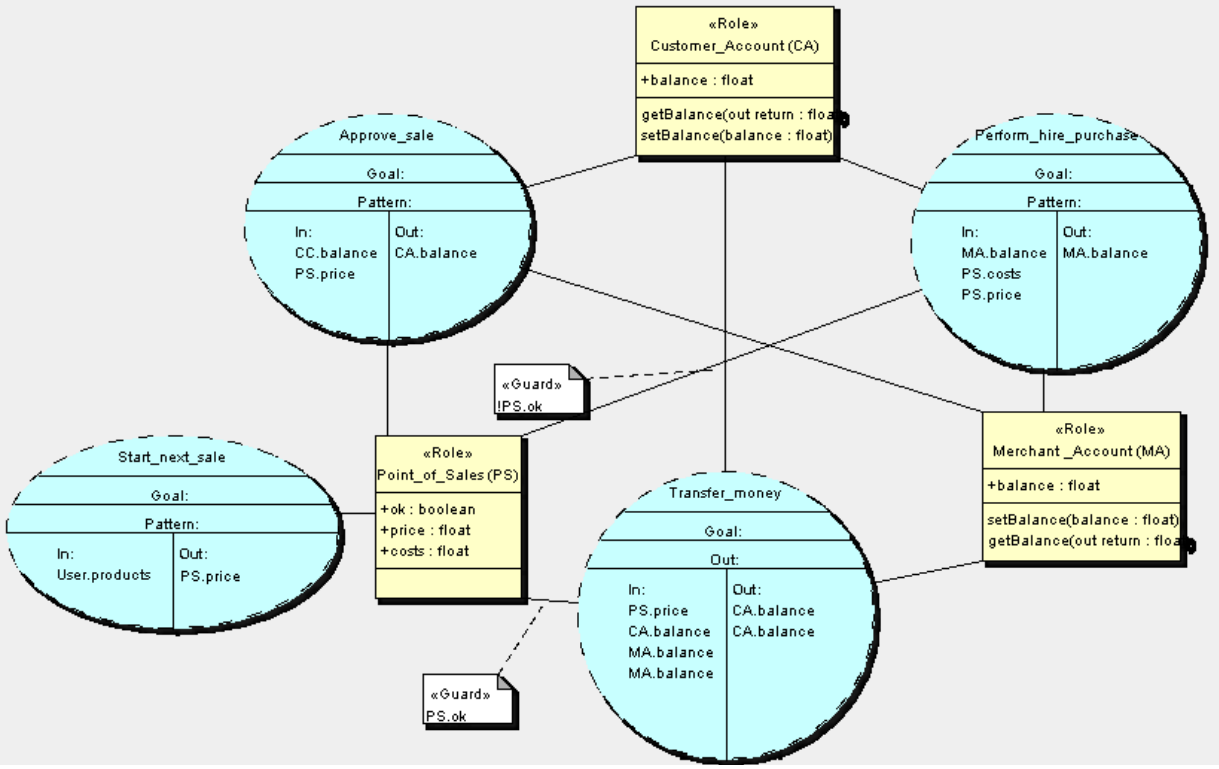

Fig. 5.1. *Static interaction view of the debit–card system.*

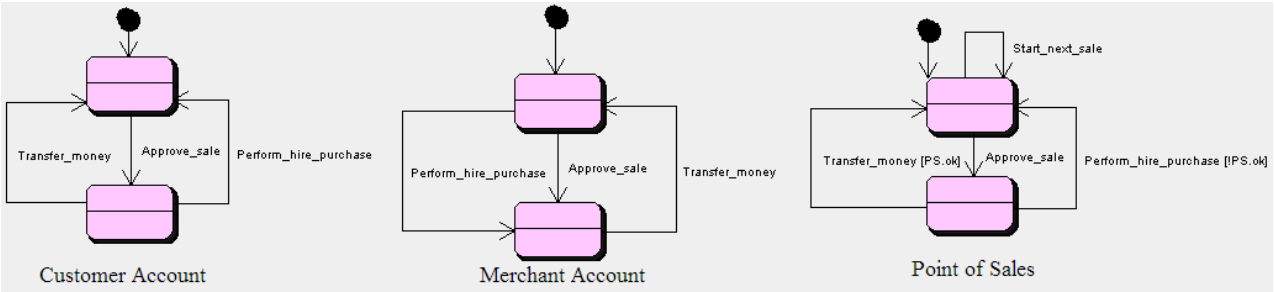

Fig. 5.2. *Plans of the roles in the debit–card system.*

the customer account cannot afford the purchase because it has not enough money, the customer account agent then pays on hire–purchase.

5. Modeling the Protocol with MaCMAS. As we showed above, our approach starts when the requirements system goals to be performed have been already obtained. Then, we model each task as an mRI as we show in the role model in Figure 5.1.

These system goals in our example are modeled as the following mRIs: *approv* is used by the CA role to inform the other parties if it can afford a purchase; *transfer* is used to transfer money from the CA to the MA by means of the PS; mRI *hire_p* is used to buy on hire-purchase; finally, there is a two-party mRI called *next_sale*, which is not further detailed, whose goal is to encapsulate the operations needed to read the sum to be transferred and the customer data from his or her debit card. For further details on the knowledge processed by each participants and in the mRI see the Appendix.

Once the mRIs are identified and linked with their participant roles, we represent their possible sequences by means of FSAs (see Figure 5.2). When an mRI is executed by more than one role it must appear a transition in all the roles that perform it. Each of these transitions represents the part of the mRIs that a role perform. Whereby,

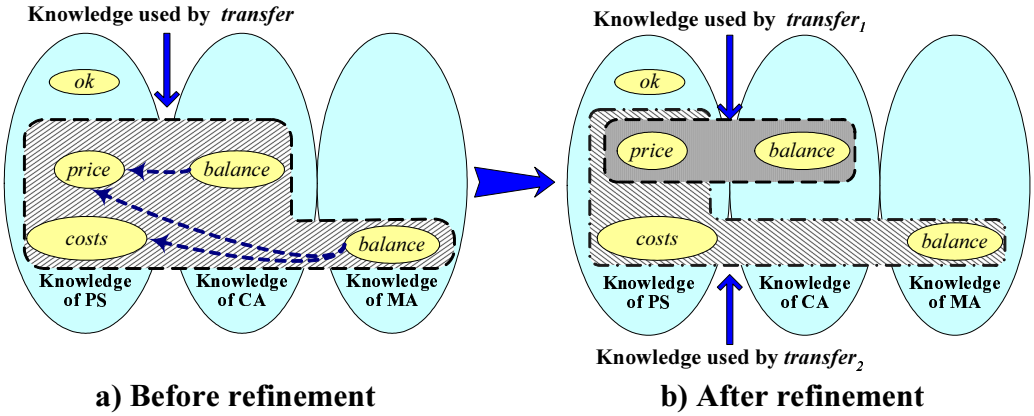

FIG. 5.3. *Decoupling mRI transfer.*

to execute an mRI we must transit from one state to another in all the roles that participate on it. Furthermore, with the algorithms presented in [25], which we outline in section 6, we can automatically infer a single FSA that represents the role model protocol as a whole. This alternative representation can be used for better readability.

Finally, each mRI have to be decorated with some additional information: such as the dependencies between they knowledge it process, a guard for each role, and so on. The knowledge dependency, as we show in the next section, can be analysed in order to refine mRIs. Furthermore, the guard of mRIs allows each role to decide if it want to execute the mRI or not, which has been proved adequate to deal with proactivity of agents [7, 19, 25].

5.1. Refinements. The model we presented in previous section takes advantage of complex three–party mRIs, which provides a high level design of the protocol. However, it should be refined in an attempt to transform its mRIs into a set of simpler ones that are closer to message sequences description. That is to say, describing them internally shall be easier. This is the next step in our approach.

The refinements are based on analysing the dependencies between the knowledge that roles use from others in a particular mRI. In order to automate the refinement process the designer has to build a dependency graph (see Figures 5.3, 5.4 and 5.5) which shall be analysed with the algorithms proposed in [18, 10]. To illustrate how our technique works we applied it to our example.

The first refinement we can apply is *decoupling* [12]. It can transform certain n–party mRIs into an m–party mRI ($m < n$) followed by an mRI with $n - m + 1$ participants. We can illustrate it by means of mRI $transfer$ in our example. Figure 5.3 shows a diagram in which we have depicted the knowledge of its roles and their dependencies. As shown, both the MA and CA need to update their balances according to some information in the knowledge of the PS. The idea is thus to decouple mRI $transfer$ into two binary mRIs so that the CA updates its balance before the MA. Thus, as we can see in Figure 5.3 mRI $transfer_1$ will executed by PS and CA, and $transfer_2$ by PS and MA (see Figure 5.7 for the new sequences of execution). We have applied this refinement to the mRI $hire\_p$, as well.

The second refinement we can apply is *participant elimination* [12]. It consists of eliminating those roles from the set of participant roles of an mRI whose knowledge is not referred to by other roles and do not refer to the knowledge of any other role. Figure 5.4 shows a diagram in which we have depicted the knowledge of the roles participating in mRI *approv* and their relationships. Obviously, role MA can be eliminated from this mRI.

Another refinement called *splitting*, which cannot be apply to our example, consist in breaking an mRI into two mRIs if the knowledge accessed by several groups of roles are disjoint as is depicted in Figure 5.5 with a fictitious mRI.

The resulting role plans after applying all refinements are presented in Figure 5.7. Apparently, they works well but we can discover that the refinements have introduced a deadlock situation if we take a closer look. Consider a trace in which the following mRIs are executed: $next\_sale, approv, transfer_1$, and $hire\_p_1$. This execution deadlocks because of an unfortunate interleaving in which, after approving a sale and charging the CA, this role is ready to interact with the PS by means of $transfer_2$; however, the MA is readied then to

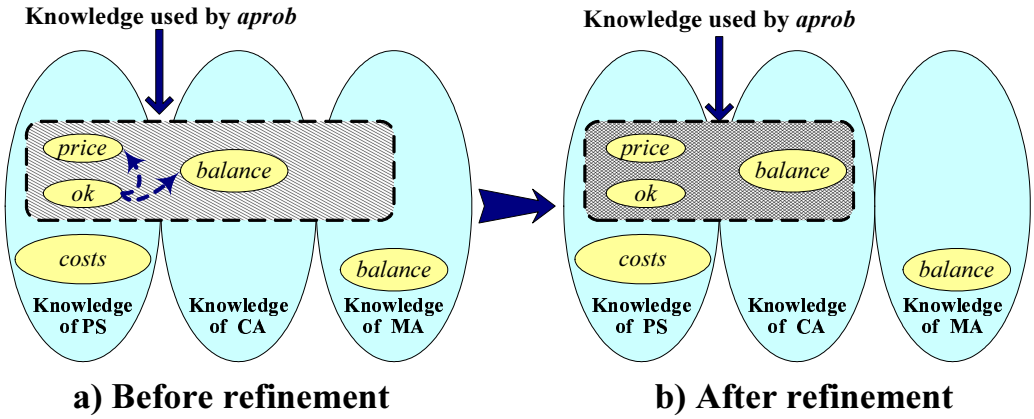

FIG. 5.4. *Eliminating role from approv.*

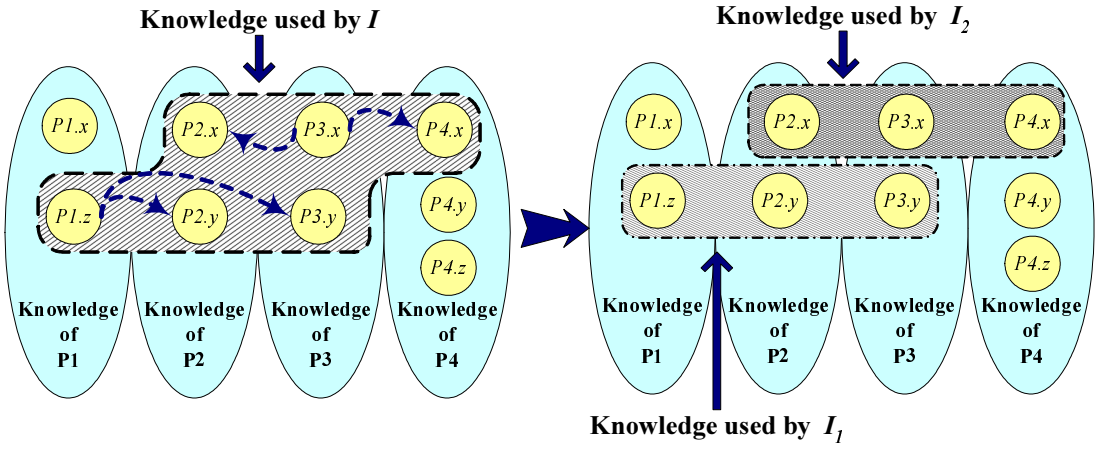

FIG. 5.5. *Splitting fictitious mRI I.*

execute both $transfer_1$ and $hire\_p_1$. If $hire\_p_1$ is executed now, it leads to a situation in which no role can continue because PS is readying $transfer_2$ and waits for the CA to ready it, the CA is readying *approv* and waits for the PS to ready it, and the MA is waiting for any of them to ready $transfer_1$ or $hire\_p_1$. This situation can be avoided if we use a guard for $transfer_i$ and $hire\_p_i$ that ensures that when one of these mRI is executed the guard of the others shall be evaluated as false, but unfortunately this is not possible in general.

These refinements allow us to execute several mRIs at the same time since the the knowledge they computed before refinements is now computed separately in different mRIs. In addition, they simplify the number of participant roles that each mRI uses, which lead us to easier implementations (the protocol to coordinate n parties is more difficult that such for two parties) [12, page. 206][2, 33, 21, 35]. Finally, another advantage is that the amount of knowledge to be processed in each mRI decreases thus easing their internal design.

For instance, the mRI *transfer* has been broken into two simpler mRIs: $transfer_1$ and $transfer_2$. $transfer_1$ computes the balance of the CA and $transfer_2$ computes the balance of the MA. Thus, simpler computations are performed. Furthermore, the original mRI had three participant roles, and the new mRIs have only two, whose coordination/negotiation protocol is simpler to implement. The refined role model is presented in Figure 5.6.

6. Ensuring Deadlock Free Refinements. Our approach to detect deadlocks is based on building an FSA and analysing its paths. Next, we present some results we need, and then we show how to construct the FSA and how to analyse it.

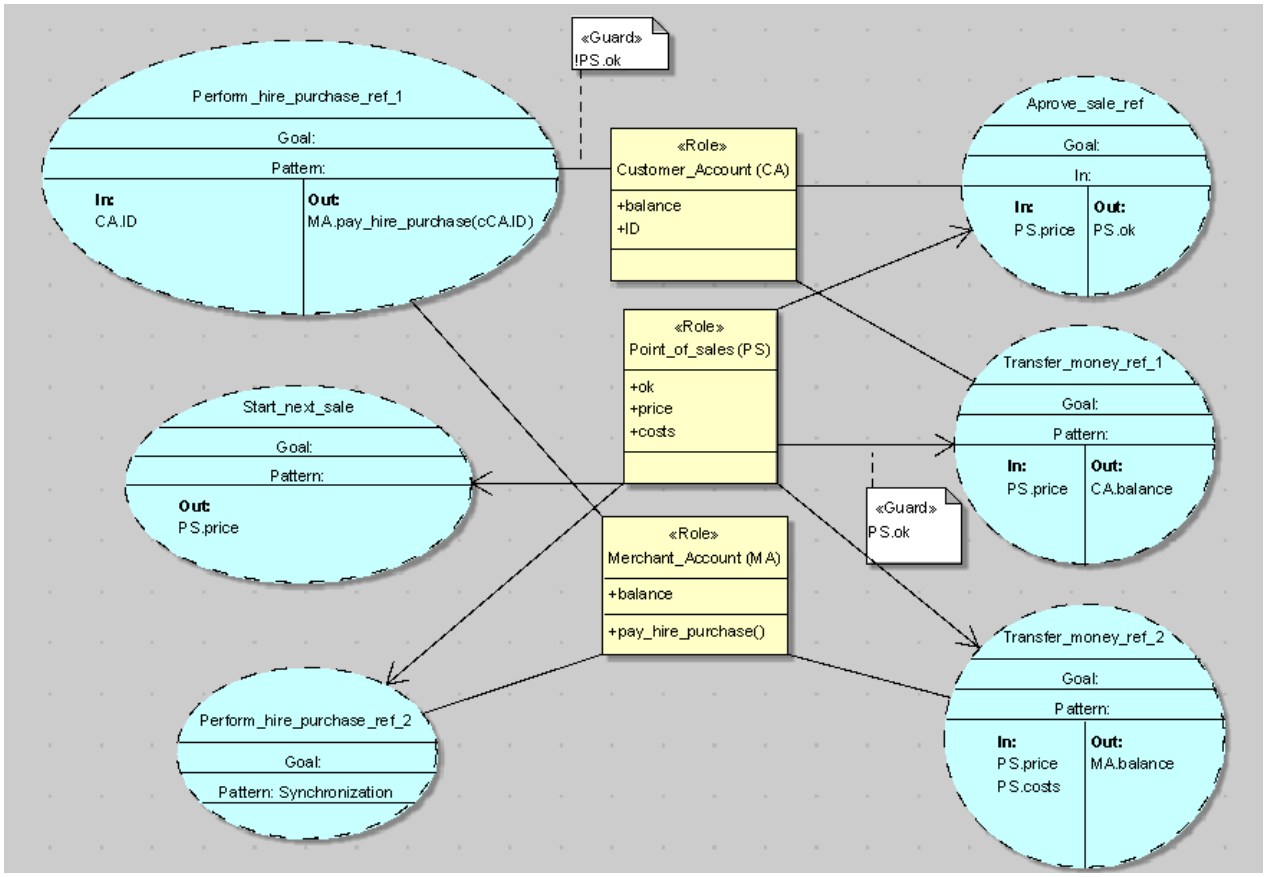

FIG. 5.6. *Role model of the debit–card system after refinements.*

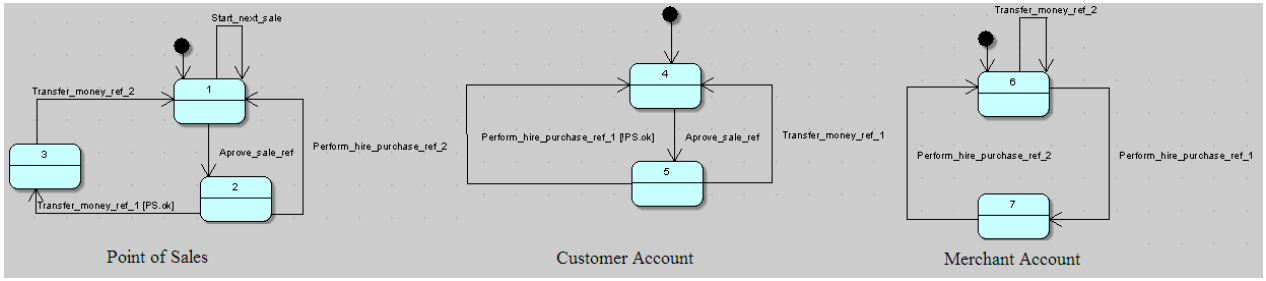

FIG. 5.7. *Role plans after refinement.*

As we can see in Figure 5.7, the definition of the protocol of each role is done by means of FSAs. They can be characterised as follows:

DEFINITION 6.1 (Finite State Automaton). *A finite state automaton (FSA) is a tuple of the form* $(S, \Sigma, \delta, s^0, F)$*, where* S *is a set of states,* Σ *is a set of mRIs (the vocabulary in FSA theory),* $\delta : S \times \Sigma \rightarrow S$ *is a transition function that represents an mRI execution,* $s^0 \in S$ *is an initial state, and* $F \subseteq S$ *is a set of final states.*

Thus, let $A_i = (S_i, \Sigma_i, \delta_i, s_i^0, F_i)$ $(i = 1, 2, \cdots, n)$ be the set of FSAs that represents each role in a role model. Starting from this information we can build a new FSA $C = (S, \Sigma, \delta, s_0, F)$ that represents the protocol as a whole, where

- $S = S_1 \times \cdots \times S_n$
- $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$
- $\delta(a, \{s_1, \ldots, s_n\}) = \{s'_1, \ldots, s'_n\}$ iff $\forall\ i \in [1..n] \cdot (a \notin \Sigma_i \wedge s_i = s'_i) \vee (a \in \Sigma_i \wedge \delta(a, s_i) = s'_i)$

- $e_0 = \{e_1^0, \ldots, e_n^0\}$
- $F = \{F_1, \ldots, F_n\}$

This algorithm has been presented in [25] and builds the new FSA exploring all the feasible executions of mRI. Their states are computed as the cartessian product of all state in FSA of roles. Then, for each new state (composed of one state of each role) we check if an mRI may be executed (all their roles can do it from that state), and if so, we add it to the result. The FSA we obtain in our example is shown in Figure 6.1.

6.1. Analysing the Resulting FSA. The final step consists in analysing the resulting FSA by searching for deadlock states, i.e., states from which a final state cannot be reached.

We use a transition relation called $\longrightarrow_B$ to calculate these states. It is applied on tuples of the form (C, N, X), where C denotes an FSA, N denotes the set of states to be analysed, and X denotes the set of deadlock states found so far. We formalise $\longrightarrow_B$ by means of the following inference rule:

$$\frac{s \in N \wedge s \notin X \wedge P = \mathsf{pred}(s, C)}{(C, N, X) \longrightarrow_B (C, N \setminus P, X \cup P)}$$

Where the predicate pred is defined as follows:

Definition 6.2 (Predecessors). *Let A be an FSA and $s \in S$ a state. We denote its set of predecessors by* $\mathsf{pred}(s, A)$ *and define it as follows:*

$$\mathsf{pred}(s, A) =$$

$$\{s' \in S \mid \exists \sigma \in \Sigma \cdot \delta(s', \sigma) = s\}$$

This transition relation allows us to explore the set of states of an FSA starting at its final states and going back to its predecessors until no new unexplored state is found. The set of unexplored states at that step is the set of deadlock states because there is no path in the FSA that links them to a final state. Therefore, we can define a function deadlock that maps an FSA into its set of deadlock states as follows:

$$\mathsf{deadlock}(C) = C_S \setminus N \text{ if } N \subseteq C_S \wedge$$

$$X \subseteq C_S \wedge (C, C_F, \emptyset) \longrightarrow_B^! (C, N, X)$$

Here, $\longrightarrow_B^!$ denotes the normalisation of $\longrightarrow_B$, i.e., its repeated application to a given tuple until it can not be further applied to the result. Formally,

$$T \rightarrow^! T' \Leftrightarrow T \longrightarrow_E^* T' \wedge \nexists T'' \cdot T' \longrightarrow_E T''$$

If deadlock returns an empty set, then the refinements we have applied do not introduce any deadlocks. Otherwise, we need to characterise the execution paths that may lead to them.

Consider that $\mathsf{deadlock}(C) = \{b_1, b_2, \ldots, b_k\}$, thus, we can build a new set of FSAs

$$B_i = (C_S, C_\Sigma, C_\delta, C_{s^0}, \{b_i\})(i = 1, 2, \ldots, k).$$

Notice that these FSAs have only a final state that is a deadlock state in the original FSA. Thus, if we use the algorithms presented in [14] for transforming an FSA into its corresponding regular expression, we can obtain the set of regular expressions that characterise the execution paths that lead to deadlocks.

If we analyse the FSA in Figure 6.1, we can easily check that its set of deadlock states is a singleton of the form $\{(3, 4, 7)\}$. Thus, if we make this the only final state, we can obtain the following regular expression that characterises the execution paths that lead to deadlocks:

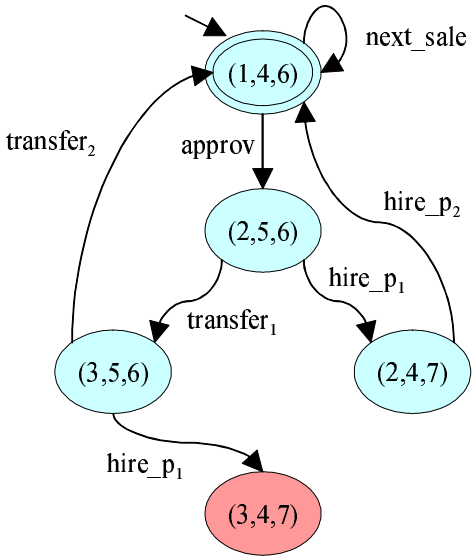

FIG. 6.1. *Resulting FSA.*

$$(next\_sale \mid approv \cdot transf_1 \cdot \\ \cdot transf_2 \mid approv \cdot hire\_p_1 \cdot hire\_p_2)^* \cdot \\ \cdot approv \cdot transf_1 \cdot hire\_p_1$$

Thus, when a set of refinements are applied we can use the technique presented above to search for deadlocks, and if they appear, we characterise it by the deadlock regular expression. Then, we can use this characterization to apply a different set of refinements and repeat this process until getting a deadlock free protocol. Finally, we obtain a set of new simpler mRIs that can be described internally and implemented easier. In our example the deadlock appears between mRI $transfer$ and $hire_p$ and the problem can be easily solved not refining one of them or applying another set of refinements.

7. Conclusions. The description of interaction protocols in complex MASs may be a difficult, tedious process due to the large number of complex tasks that agents must perform coordinately. Thus, in order to palliate this problem, we have proposed a refinement technique integrated in a methodology that is based on an interdisciplinary technique that builds on MAS and distributed systems research results.

Our technique improves previous research in that we add some protocol views between requirements analysis and the description of a protocol by means of message sequences; we use interactions as first class modeling elements. Furthermore, these descriptions are easily refined to reach the needed abstraction level to be described internally. Thus, we provide a progressive method to proceed from requirements analysis to message sequences descriptions. Furthermore, we have provided an automatic method to detect deadlocks.

REFERENCES

[1] R. BACK, A calculus of refinements for program derivations. *Acta Informatica*, 25(6):593–624, 1988.
[2] R. BAGRODIA, Synchronization of asynchronous processes in CSP. *Transactions on Programming Languages and Systems*, 11(4):585–597, Oct. 1989.
[3] B. BAUER, J. MULLER, AND J. ODELL, Agent uml: A formalism for specifying multiagent interaction. In M. Wooldridge and P. Ciancarini, editors, *Proceedings of 22nd International Conference on Software Engineering (ISCE)*, LNCS, pages 91–103, Berlin, 2001. Springer-Verlag.
[4] G. CAIRE, F. LEAL, P. CHAINHO, R. EVANS, F. GARIJO, J. GOMEZ, J. PAVON, P. KEARNEY, J. STARK, AND P. MASSONET, Agent oriented analysis using MESSAGE/UML. In *Proceedings of Agent-Oriented Software Engineering (AOSE'01)*, pages 101–108, Montreal, Canada, May 2001.
[5] J. C. CORBETT, Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering*, 22(3):161–180, March 1996.

[6] R. Corchuelo, D. Ruiz, M. Toro, and A. Durán, Avances en la coordinación de objetos activos. *Novática*, 143:34–37, Jan.–Feb. 2000.
[7] J. C. Cruz, OpenCoLaS a coordination framework for CoLaS dialects. In *Proceedings of COORDINATION 2002*, York, United Kingdom, 2002.
[8] E. Dijkstra, *A Discipline of Programming*. Prentice-Hall, 1976.
[9] T. Elrad and N. Francez, Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2:55–173, 1982.
[10] N. Francez and I. Forman, Synchrony loosening transformations for interacting processes. In J. Baeten and J. Klop, editors, *Proceedings of Concurr'91: Theories of concurrency—Unification and extension*, number 527 in LNCS, pages 27–30, Amsterdam, The Netherlands, Aug. 1991. Springer-Verlag.
[11] N. Francez and I. Forman, *Interacting processes: A multiparty approach to coordinated distributed programming*. Addison–Wesley, 1996.
[12] N. Francez and I. R. Forman, *Interacting Processes*. Addison–Wesley, 1996.
[13] C. A. R. Hoare, Communicating sequential processes. In R. M. McKeag and A. M. Macnaghten, editors, *On the construction of programs—an advanced course*, pages 229–254. Cambridge University Press, 1980.
[14] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.
[15] C. Iglesias, M. Garrijo, and J. Gonzalez, A survey of agent-oriented methodologies. In J. Müller, M. P. Singh, and A. S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 317–330. Springer-Verlag: Heidelberg, Germany, 1999.
[16] N. Jennings, An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
[17] A. Karageorgos and N. Mehandjiev, A design complexity evaluation framework for agent-based system engineering methodologies. In A. Omicini, P. Petta, and J. Pitt, editors, *Fourth International Workshop Engineering Societies in the Agents World*, volume 3071 of *Lecture Notes in Computer Science*, pages 258–274. Springer, 2004.
[18] S. Katz, I. Forman, and W. Evangelist, Language constructs for distributed systems. In *IFIP TC2 Working Conference on Programming Concepts and Methods*, Galilea, Israel, Apr. 1990.
[19] J. Koning, M.Huget, J. Wei, and X. Wang, Extended modeling languages for interaction protocol design. In M. Wooldridge, P. Ciancarini, and G. Weiss, editors, *Proceedings of Second Internationa Workshop on Agent-Oriented Software Engineering (AOSE'02)*, LNCS, Montreal, Canada, May, 2001. Springer–Verlag.
[20] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes, On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, Boston, MA, 1990.
[21] N. Natarajan, A distributed synchronisation scheme for communicating processes. *The Computer Journal*, 29(2):109–117, Apr. 1986.
[22] O. M. G. (OMG), Unified modeling language: Superstructure. version 2.0. Final adopted specification ptc/03–08–02, OMG, August 2003. `www.omg.org`
[23] G. Papadopoulos and F. Arbab, Coordination models and languages. In *Advances in Computers*, volume 46. Academic Press, 1998.
[24] D. L. Parnas, On the criteria to be used in decomposing system into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.
[25] J. Peña, R. Corchuelo, and J. L. Arjona, Towards Interaction Protocol Operations for Large Multi-agent Systems. In *Proceedings of the 2nd Int. Workshop on Formal Approaches to Agent-Based Systems (FAABS 2002)*, volume 2699 of *LNAI*, pages 79–91, NASA-GSFC, Greenbelt, MD, USA, 2002. Springer–Verlag.
[26] J. Peña, R. Corchuelo, and J. L. Arjona, A top down approach for mas protocol descriptions. In *ACM Symposium on Applied Computing SAC'03*, pages 45–49, Melbourne, Florida, USA, 2003. ACM Press.
[27] J. Peña, M. G. Hinchey and A. Ruiz-Cortés, Multiagent system product lines: Challenges and benefits. *Communications of the ACM*, 49(12), December 2006.
[28] J. Peña, M. G. Hinchey, A. Ruiz-Cortés and P. Trinidad, Building the core architecture of a nasa multiagent system product line. In *7th International Workshop on Agent Oriented Software Engineering 2006*, page to be published, Hakodate, Japan, May, 2006. LNCS.
[29] J. Peña, M. G. Hinchey, M. Resinas, R. Sterritt, and J. L. Rash, Designing and managing evolving systems using a mas-product-line approach. *Journal of Science of Computer Programming*, 2006.
[30] J. Peña, R. Levy, and R. Corchuelo, Towards clarifying the importance of interactions in agent-oriented software engineering. *International Iberoamerican Journal of AI*, 9(25):19–28, 2005.
[31] J. Peña, M. G. Hinchey, and R. Sterritt, Towards modeling, specifying and deploying policies in autonomous and autonomic systems using an aose methodology. In *EASE '06: Proceedings of the Third IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASE'06)*, pages 37–46, Washington, DC, USA, 2006. IEEE Computer Society.
[32] R. Pressman, *Software Engineering: a Practitioner's Approach*. MacGraw Hill, New York, N.Y., 2nd edition, 1986.
[33] F. Schneider, Synchronization in distributed programs. *ACM Transactions on Programming Languages and Systems*, 4(2):125–148, Apr. 1982.
[34] M. Singhal, Deadlock detection in distributed systems. *Computer Magazine of the Computer Group News of the IEEE*, 22(11):37–48, 1989.
[35] J. Van De Snepscheut, Synchronous communication between asynchronous components. *Information Processing Letters*, 13(3):127–130, Dec. 1981.
[36] M. Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings 1st Annual IEEE Symp. on Logic in Computer Science, LICS'86, Cambridge, MA, USA, 16–18 June 1986*, pages 332–344. IEEE Computer Society Press, Washington, DC, 1986.

[37] M. Wood and S. A. DeLoach, An overview of the multiagent systems engineering methodology. In *Proceedings of the First International Workshop on Agent-Oriented Software Engineering*, number 1957 in LCNS, Limerick, Ireland, 2001. Springer-Verlag.
[38] M. Wooldridge, N. R. Jennings, and D. Kinny, The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

Appendix A. IP Code of the example. It exists several languages based on the Multi-party Interactions (MPI) to describe systems where several processes have to coordinate [6, 10, 13]. IP [12] is worthy of special attention since, although its implementation is relatively simple, moreover it allows to check properties thanks its formal character. Following we will do a brief review of its statements and its more relevant characteristics for our work, and finally we will write the source code of the debit–card system example.

An IP specification is built with a set of sequential processes that cooperates between them using multiparty interactions. Its abstract syntax is the following:

$$\begin{array}{rcl} S & ::= & I_1[\overline{x:=e}] \\ & | & [[]_{i=1}^{n} B_i \,\&\, I_i[\overline{x_i:=e_i}] \rightarrow S_i] \\ & | & \star[[]_{i=1}^{n} B_i \,\&\, I_i[\overline{x_i:=e_i}] \rightarrow S_i] \\ & | & S_1; S_2 \\ & | & skip \end{array}$$

Each processes will be able to participate in several interactions, but only one at the same time. The statement of interaction has the form $I[\overline{x:=e}]$ where I is the name of the interaction and $\overline{x:=e}$ is a sequence of parallel assignments in where we can consult the state of the rest of participants in the interaction, usually referred as communication code. Each Interaction has a set of fixed participants in the set of processes of the system, so that it can be executed only when not any is executing other interaction and all of them are in a point of the specification where the questioned interaction can be executed.

```
TRANSFERS :: [PST() || CustomerAccount() || MerchantAccount()],
where
 PST() :: s: sale := null, ok : boolean;
   *[ v ≠ null & approv[ ok := (cc.balance ≥ s.price)] →
      [ok & transfer[v := null] → skip
      []
      ¬ok & hire_p[] → skip]
    []
      v = null & next_sale[...] → skip],
 CustomerAccount() :: cc: account;
   *[ approv[] →
      [transfer[cc.balance := cc.balance - s.price] → skip
      []
      hire_p[cc.hire_purchase(ma.ID)] → skip] ],
 MerchantAccount() :: ma: account;
   *[ approv[] →
      [transfer[ ma.balance := ma.balance + s.price - v.m_costs ] → skip
      []
      hire_p[ma.balance := ma.balance - s.m_costs] → skip ]
    ].
```

Fig. 7.1. *IP specification of the debit–card system.*

For example, if we analyze the interaction *transfer* in the IP code of the example in the figure 7.1, we can notice it has in its participants[2] with the *PST*, with the *CustomerAccount* and with the *MerchantAccount*. This interaction will not be executed until all its participants will be in an adequate point of the specification and

[2]To determine the participants of an interaction we only have to see in which processes appears in the specification

```
TRANSFERS :: [PST() || CustomerAccount() || MerchantAccount()], where
 PST() :: v: sale := null; ok : boolean;
   *[ v ≠ null & approv[ ok := (cc.balance ≥ s.price)] →
      [ok & transfer_1[] → transfer_2[v := null]
      []
      ¬ok & hire_p_2[] → skip
      ]
    []
      v = null & next_sale[...] → skip ],
 CustomerAccount() :: cc: account;
   *[approv[] →
      [transfer_1[cc.balance := cc.balance - s.price] → skip
      []
      hire_p_1[cc.hire_purchase(ma.ID] → skip ] ],
 MerchantAccount() :: ma: account;
   *[ ι[] →
      [transfer_2[ ma.balance := ma.balance + s.price - s.m_costs] → skip
      []
      hire_p_1[ma.balance := ma.balance - s.m_costs] → hire_p_2[] ]
    ].
```

FIG. 7.2. *IP specification of the example after applying the refinements.*

when this will happen, its participant will execute its communication code. For example, the *PST* will calculate the value of variable *ok* using the balance of the *CustomerAccount* and the amount to transfer.

IP also has statements to write non-deterministic choice with guards $[[]_{i=1}^{n} G_i \rightarrow S_i]$ and loops with nondeterministic choice with guards $*[[]_{i=1}^{n} G_i \rightarrow S_i]$. The guards are of the form $B\&a[\overline{x:=e}]$, where B is a boolean condition involving the local state of a process, and the rest is an usual interaction statement. The behaviour of these statements is very simple: The non-deterministic choice checks all the boolean conditions and wait then for the interactions whose boolean condition is true to have all its participants; if no one could do so the statement will not have any effect. In loops the behaviour is similar, only that it will repeat the non-deterministic choice until all the boolean conditions are false.

Furthermore, in IP we can make the statements above to execute sequence $(S_1; S_2)$, and we can use the null statement that is represented as *skip*.

Finally, the code resultant after applying all the refinements described above is shown in Figure 7.2.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006