The TuCSoN Coordination Infrastructure for Virtual Enterprises

Alessandro Ricci Andrea Omicini Enrico Denti
DEIS, Università di Bologna, Italy
E-mail: {aricci,aomicini,edenti}@deis.unibo.it

Abstract

Virtual Enterprises (VE) and Workflow Management Systems (WFMS) require deployable and flexible infrastructures, promoting the integration of heterogenous resources and services, as well as the development of new VE's business processes in terms of workflow (WF) rules coordinating the activities of VE's component enterprises. In this paper, we argue that a suitable general-purpose coordination infrastructure may well fit the needs of VE management in a highly dynamic and unpredictable environment like the Internet, by providing engineers with the abstractions and run-time support to address heterogeneity of different sorts, and to represent WF rules as coordination laws. We discuss the requirements for VE infrastructures, and suggest why VE management and WFMS may be seen as coordination problems. Then, we introduce the TuCSoN *coordination model and technology, and show, both in principle and in a simple case study, how such a coordination infrastructure can support the design and development of VE's WFMS.*

1. Virtual Enterprise management as a coordination problem

Two main issues arise when building a Virtual Enterprise (VE henceforth). The first is the integration of the existing services and infrastructures provided by the VE's component enterprises, which may range from passive information sources to activities such as services and business processes. The main problem, here, is heterogeneity, coming out at many different levels – from the technology level, to the information and the process levels. The second relevant issue is the development of the VE's distributed workflow management system (WFMS henceforth). On the one hand, the WFMS should integrate the different business processes featured by the VE's component enterprises. On the other hand, it should enable the specification, execution, and monitoring of the VE's specific business processes.

Both issues involve autonomous and heterogeneous activities, whose inter-dependencies should be properly managed and governed in order to reach global system goals. Such dependencies concern sharing and exchanging heterogeneous information resources, task assignments for VE business processes, and temporal and prerequisite constraints among the several, heterogeneous activities that characterise a VE workflow. Since managing and governing interaction involves the field of *coordination* [14], the above issues can be understood as *coordination problems*. So, a *coordination infrastructure* providing both suitably expressive coordination abstractions and an easily-deployable coordination technology fits well the needs of WFMS in VEs.

Such an infrastructure should *(i)* help to integrate VE's existing services and resources in a shared environment, minimising the impact and the requirement of resources on the local structures, and *(ii)* allow WFMS information and activities to be captured within a uniform conceptual framework, representing workflow rules as coordination laws, embodying them at run-time in coordination media.

In this paper we show how a coordination model [9] like TuCSoN [19] and the corresponding infrastructure [28] may be exploited to support VEs, and how coordination media like *tuple centres* [18] may be effectively used as workflow engines in the design and development of WFMS. The TuCSoN model promotes a clean separation among autonomous interacting activities, which can be effectively modelled as dynamic sets of (possibly mobile) agents, governed by social laws that rule interaction and manage inter-agent dependencies. There, WFMS rules can be expressed as coordination rules, and embodied within tuple centres as behaviour specifications. The combined effect of the agent paradigm [30] and the uncoupling properties of generative communication – lifted by tuple centres from the language level up to the design level [18] – provide the expressive power and flexibility to face the issues of WFMS in VE.

The remainder of this paper is organised as follows. Section 2 focuses on the main challenges and requirements raised by VE and distributed WFMS development. In Section 3, we introduce the notion of *objective coordination* [17], and show how WFMS problems in the VE field may be solved by adopting a coordination infrastructure like TuCSoN, outlining the advantages of this approach with respect to other agent-based approaches that offer no support for objective coordination. Finally, in Section 4 we sketch an example of a TuCSoN-based WFMS application representing a virtual bookshop built as a VE.

2. Infrastructure Requirements

Electronic Markets and Virtual Enterprises are the most relevant economical structures emerged from e-commerce. A VE is a temporary aggregation of autonomous and independent enterprises connected through a network, brought together to deliver a product or service in response to the customer needs [23]. So, VEs typically mean to combine the core competence of several autonomous and heterogeneous enterprises in a new agile and flexible enterprise, addressing original industrial / business targets [23].

An infrastructure supporting VE management has to address two main issues: *(i)* the integration of selected heterogeneous resources provided by single participants, and *(ii)* the execution of VE specific business processes that feature their own dynamics while exploiting as many resources, services, and processes from the component enterprises as possible. Since both aspects involve a multiplicity of autonomous interacting entities and the management and government of their interactions and inter-dependencies, they can be understood as coordination issues [14].

Heterogeneity comes out at the technology, information, and process levels. Technological heterogeneity is due to the different software environments (operating system, middleware) used as infrastructure for the distributed computing environment [15], while information can be heterogeneous because of differences in syntax, semantics, and ontology. At the process level, heterogeneous activities and business processes have to interact despite the differences in the nature of the involved entities – humans, software services, software agents of various computational paradigms, etc. – and in interaction protocols and languages.

So, the first requirement for a VE infrastructure is to promote a conceptual homogeneity of resources, information, and activities. Working as the organisational glue, a VE infrastructure should support consistent interactions both with shared services and among processes, while minimising the impact of the VE on participants' local systems [31]. In this context, security and trust issues are particularly relevant, in that a VE infrastructure should provide abstractions and mechanisms to ensure security, safety and privacy, protecting the VE from the outside world, while taking into account the mutual requirements of the individual VE components. An infrastructure for VE management should help to govern the dependencies that characterise the execution of VE business processes, such as task assignments, prerequisite and temporal constraints [14], which require the development of a WFMS [11].

Currently, workflow systems are the most important model used by organisations to automate their business processes, supporting their specification, execution and monitoring. A business process is precisely a workflow – that is, a set of activities to be executed in some order, involving multiple collaborating entities in a distributed environment to accomplish a given task [11]. A workflow system is meant to improve the process throughput, to promote a better use of resources, and to enable efficient process tracking [24]. Moreover, since workflow applications are subject to frequent changes caused by the business environments [6], flexibility and adaptability are required in different directions, from dynamic evolution of existing coordination policies to proper reactions to unpredictable situations [13].

In the VE case, WFMS have typically to face the issue of distribution – that is, to coordinate a multiplicity of heterogeneous activities spread over the network. As explicitly stated in the definition of requirements for a WFMS standard [24, 29], a distributed workflow infrastructure should be based on the principle of loose-coupling between heterogeneous workflow *backends* – i.e., WF engines or coordinators, where coordination takes place – and heterogeneous workflow participants/service providers.

Accordingly, an infrastructure for WFMS should provide support for *(i)* communication among participants and backends with no need to agree on point-to-point protocols, as well as *(ii)* for flexible and scalable workflow participation, allowing participants to adopt both traditional devices (such as desktop PC) and non-traditional devices (such as thin clients, or PDA), requiring no a-priori information on backends, nor dedicated connections to the backends [24]. Moreover, such an infrastructure should support *(iii)* disconnected operation, allowing WF participants to be mobile, location-independent, and non-frequently connected to the network [24]; finally, it should enable *(iv)* traceability of the business process state [15].

In addition, inter-organisational workflow systems [5] for use in the VE context should be able to cope both with the openness and distribution of the Web environment, and with heterogeneity at several levels (see Section 1).

In order to face such requirements, the VE infrastructure should promote a clean separation between the conceptual places where the workflow rules are specified and enforced, and the workflow participants, by uncoupling the coordination activity and the coordinated activities. Also, workflow rules should be explicitly represented, inspectable and modifiable, possibly in a dynamic way.

3. VE, WFMS, and Coordination in TuCSoN

TuCSoN is an infrastructure for the coordination of Internet agents, particularly suitable to mobile information agents [19]. The TuCSoN coordination model is based on the notion of (logic) *tuple centre* [18], which is a Linda tuple space [8] empowered with the ability to define its behaviour in response to communication events according to the specific coordination needs.

It has been argued [1, 21, 22] that the openness and the wideness of the Internet scenario make it suitable to conceive the Internet as a multiplicity of independent environments (e.g. Internet nodes or administrative domain of nodes), and to design applications in terms of agents that explicitly locate and access resources in this environment. The TuCSoN support for such an approach to application design is based on a multiplicity of independent interaction spaces, namely *tuple centres*, that abstract the role of the environment. (Mobile) agents access tuple centres by name, either locally in a transparent way, or globally on the Internet in a network-aware fashion [19].

A local interaction space can be used by agents to access the local resources of an environment and as an *agora* where to *meet* other agents and coordinate activities with them. This is why tuple centres, as fully distributed interaction media, can be understood as *social abstractions* [10], which allow to constrain agent interactions explicitly and to enforce the coordination and cooperation activities that define the agent aggregation as a society. Interaction protocols can then be designed and distributed among agents and media, adopting the balance that is most adequate to the specific application goals.

One first advantage is the enhancement of agent autonomy: agents can be designed focusing only on their own goals and tasks, disregarding dependencies from other agents and without having to track (open) environment evolution. Moreover, prescriptive coordination can be achieved, getting the capability to constrain agent interactions so that they reflect sound behaviours, according to the social goals defined for the agent organisation. Decentralisation is enhanced, too, since tuple centres are spread over the network, living in infrastructure nodes visited by rambling agents migrating from node to node. Finally, the topological nature of the TuCSoN global interaction spaces [19] makes it possible to deal with the typical issues of distributed systems [3] – in particular, to enforce flexible security policies, workload allocation policies, fault tolerance policies.

Heterogeneity is addressed in TuCSoN both at the model and infrastructure level. From the technology viewpoint, TuCSoN has been developed in Java, thus guaranteeing portability among platforms. Its architecture is based on a light-weight core, designed to support different sorts of middleware – from sockets to Java RMI, CORBA, and proprietary solutions. This minimises the impact on hosting environments and enables a wide range of hardware devices (desktops, PDAs , etc) to exploit the infrastructure.

The TuCSoN model directly supports the integration between heterogeneous information sources. Its tuple-based coordination model naturally supports uncoupled interactions, thanks to *generative communication* (communication data that survive communication acts) [8], which provides *agent uncoupling*, from both the *space* and *time* viewpoints – that is, agents can interact needing neither to know each other nor where they are (space uncoupling), and independently of their contemporaneous existence (time uncoupling). These are key aspects for a peer-to-peer interaction model between VE backends and participants, and make it possible to go beyond point-to-point communication protocols, supporting location-independent, disconnected workflow participation.

TuCSoN tuple centres are more powerful coordination abstractions than tuple spaces, since they can be programmed so as to react to communication events, thus defining the coordination laws to rule agent interactions. As discussed in [17], this allows logic tuple centres to act as intelligent information mediators, mapping knowledge to knowledge. So, even in a VE open environment, participants are not required to change their model for knowledge representation, and dynamic integration of new participants is easily supported with no responsibilities or accommodation (changes) for present and future interacting agents.

Tuple centres' dynamic behaviour specification is also the key property that allows TuCSoN to fully support heterogeneity at the process level and, more generally, to face the coordination of dynamic aspects inside the VE. In fact, TuCSoN supports a form of *uncoupled coordination* [27], or *objective coordination* [25], where coordination rules are embodied outside interacting entities, and can effectively govern their interaction in a predictable way. This is a key aspect both to provide the clean separation between workflow participants and backends and to enforce prescriptive coordination, two important requirements illustrated in the previous Section, not easily fulfilled by an infrastructure that supports subjective coordination only.

Coordination rules in TuCSoN are expressed as programs in the ReSpecT specification language [4], which is embodied within tuple centres: as a Turing equivalent language, ReSpecT ensures that any computable coordination rule can be enforced. As a consequence, workflow rules can be defined in TuCSoN as coordination laws decoupled from interacting agents, and embedded into properly defined coordination abstractions, so as to keep WF coordinators and participants distinct at design, development, and run-time. ReSpecT can be considered as a sort of an assembly language for interaction, with strong formal basis, yet general and powerful enough to support the development of higher-level specification language, tuned for specific applications. For these reasons, future investigations are planned to design and implement classic workflow languages, including the visual ones, on top of ReSpecT.

Finally, coordination laws specified in tuple centre in TuCSoN are dynamically inspectable. Since a ReSpecT behaviour specification is structured as a multi-set of logic tuples, an intelligent agent can in principle retrieve it and reason about current rules governing the interactions – in particular, about workflow rules. Moreover, since a tuple centre's behaviour specification can be dynamically changed, an intelligent agent could monitor the evolution of a VE, and pro-actively adapt WF rules in order to improve VE behaviour and performances. This would indeed provide a TuCSoN-based WFMS with the flexibility and adaptability required to face rapid changes in business environments, as well as to react to unpredicted situations.

4. A Case Study

In order to experiment with the effectiveness of the TuCSoN technology for VE and WFMS, we set up a virtual bookshop case study. The virtual bookshop is a VE that aggregates several companies of different sorts working together to sell books on the Internet: there are the *book seller*, the *carrier* (dispatching books from sellers to clients), the *interbank service*, the *book publisher*, and the *Internet service provider*. Any individual enterprise wishing to play a role in this VE can dynamically join it at any time by enabling the TuCSoN infrastructure and suitably configuring it so as to accomplish the simple VE's requirements.

In the following, we sketch the execution of a simple business process: the on-line purchase of a *single book* (first case) and of a *book set* (second case). In the first case, a single book is bought from a selected bookseller, and must be delivered to the customer by a specific carrier: this example shows a workflow rule enforcing the sequence of autonomous and heterogeneous activities (book order, dispatching, payment). The second case involves the purchase of a set of books, each from (possibly) a different bookseller, again having all them delivered by a single, specific carrier. This example illustrates the AND-branching and AND-join workflow rules that enforce the concurrent execution of the ordering activities, whose results are then collected and provided to the dispatching activity.

4.1. Purchase of a single book

This first example shows a possible way to manage a set of activities to be executed in sequence, facing coordination issues about synchronisation and information flow among the involved activities. The business process starts when a buyer interface agent places a tuple describing a new order in the proper tuple centre at the Internet portal node, which is programmed with the ReSpecT specification illustrated in Table 1: let us suppose it is called `vbs`.

This event triggers the first reaction of Table 1: a new transaction ID is generated, the information about the order is saved (tuple `order_info`) and the state of the order is registered (tuple `order_state`). The order state may be `ordering` (the book is not yet available at the bookseller's), `ready` (the book is ready at the bookseller's but not yet collected by the carrier), `dispatching` (the book has been collected by the carrier, but not yet delivered to the customer), `delivered` (the book has been delivered to the customer, and payment is going on) or `closed` (payment has been successfully brought to end). A new agent, `monitorAgent`, is then spawned: its tasks is to monitor the business process and provide the customer with information about the order state upon request. The actual book order at the bookseller's is performed by a mobile agent (`buyerAgent`) moving to the seller's node.

When the book is ready at the bookseller's, the buyer mobile agent comes back home and outputs a new `book_ready` tuple. This event triggers the second reaction of Table 1, which updates the order state and delegates book delivery to the mobile agent `carrierAgent`, which goes to the selected carrier's node and interacts with the local services as appropriate. From the carrier's node, `carrierAgent` notifies the VE about the delivery status by producing a `book_dispatched` tuple in the home tuple centre when the book is picked up from the bookseller's, and a `book_delivered` tuple when the book is successfully delivered to the customer. The payment activity is then started (agent `paymentAgent`, spawned by the fourth reaction of Table 1): when it succeeds, a logging agent (`loggingAgent`, fifth reaction) is spawned to backup the business process history on a safe storage.

4.2. Purchase of a book set

This example shows how to manage a set of activities to be executed concurrently, whose results must be collected and synchronised, according to the SPLIT(BRANCH) and AND-JOIN workflow rules. Again, the business process starts when a new order tuple is placed; here, the tuple describes the purchase of a multiplicity of books, possibly from different booksellers. In this case, one buyer mobile agents is spawned for each book to be ordered, reflecting a branching of activities (first three reactions of Table 2). For each concurrent activity, a `suborder_state` tuple describes the state of the specific sub-order. When a buyer agent communicates the success of a sub-order activity (placing a tuple `book_ready`) the sub-order state is updated, and a check about order completion is done (fourth to sixth reaction of Table 2). When all sub-orders have been processed successfully (AND-join workflow rule) the `carrierAgent` mobile agent is spawned, and everything proceeds as in the single book case (last reactions of Table 2).

Table 1. Purchase of a single book

```
% a new order is placed
reaction(out(new_order(Customer,[(Book,Seller)],Carrier)),(
    in_r(new_order(Customer,[(Book,Seller)],Carrier)),
    % generate a new transaction ID
    in_r(trans_id_counter(ID)),
    NextID is ID + 1,
    out_r(trans_id_counter(NextID)),
    % set up order info
    out_r(order_info(ID,Customer,[(Book,Seller)],Carrier)),
    out_r(order_state(ID,ordering)),
    % spawn agent monitoring the business process
    spawn(monitorAgent(ID)),
    % execute book order from specified seller
    spawn(buyerAgent(ID,Book,Seller)))).
% book ready, execute dispatching
reaction(out(book_ready(ID)),(
    in_r(book_ready(ID)),
    % change order status
    in_r(order_state(ID,ordering)),
    out_r(order_state(ID,ready)),
    % activate book dispatching
    rd_r(order_info(ID,Customer),[(Book,Seller)],Carrier)),
    spawn(carrierAgent(
            ID,Carrier,[(Book,Seller,Customer)]))))).
% book has been collected from sender (seller)
reaction(out(book_dispatched(ID)),(
    in_r(book_dispatched(ID)),
    % change order status to dispatching
    in_r(order_state(ID,ready)),
    out_r(order_state(ID,dispatching)))).
% book delivered to customer, execute payment
reaction(out(book_delivered(ID)),(
    in_r(book_delivered(ID)),
    % change order status to delivered
    in_r(order_state(ID,dispatching)),
    out_r(order_state(ID,delivered)),
    % execute payment activity
    spawn(paymentAgent(ID)))).
% payment successful:  backup transaction
reaction(out(payment_ok(ID)),(
    in_r(payment_ok(ID)),
    % change order status to closed
    in_r(order_state(ID,_)),
    out_r(order_state(ID,closed)),
    % execute transaction backup activity
    in_r(order_info(ID,Customer,BookList,Carrier)),
    spawn(loggingAgent(order(ID,Customer,
            BookList,Carrier))))).
```

5. Related Works

Several types of workflow systems have been developed both at the research level (for instance WIDE, Regatta, APM, MILANO, TriGS) and the commercial level (ATI Action Workflow, XSoft InConcert, TeamWARE Flow). The above systems face the requirements for classic Workflow Managements, but do not easily fit Inter-Organisational Workflow systems and Web-based workflow management systems [5]. As discussed in this article, these requirements seem to be properly manageable by a general-purpose agent coordination infrastructure like TuCSoN. The METEOR$_2$ system [16] goes beyond the classic workflow systems, towards Web-based workflow, workflow adaptation and integral support for collaboration. In [26], the need for higher level forms of middleware that support coordination, collaboration and information management clearly emerges. TuCSoN is meant to play this role, since it is general-purpose enough to support distributed workflow management, as well as interaction, coordination and collaboration among heterogeneous (both human and software) agents.

Some approaches support VE management using mobile and intelligent agent technologies [31, 7, 12], while others support VE management through WFMS using a multi-agent system with mobile agents [2, 15]. While the first do not provide an explicit model for WFMS, the second support business processes execution with subjective coordination, that is, encapsulating the social rules into agents. We found objective coordination more effective than subjective coordination for WFMS, because of the engineering problems endorsed by subjective coordination with respect to dynamism, flexibility, heterogeneity and traceability issues.

References

[1] G. Cabri, L. Leonardi, and F. Zambonelli. Mobile-agent coordination models for internet applications. *IEEE Computer*, 33(2):82–89, February 2000.

[2] P. K. Chrysanthis, T. Znati, S. Banerjee, and S.-K. Chang. Establishing virtual enterprises by means of mobile agents. In *Workshop on Research Issues in Data Engineering (RIDE 1999)*, pages 116–125. IEEE CS, March 1999.

[3] M. Cremonini, A. Omicini, and F. Zambonelli. Ruling agent motion in structured environments. In *High Performance Computing and Networking*, volume 1823 of *LNCS*, pages 187–196. Springer-Verlag, 2000.

[4] E. Denti, A. Natali, and A. Omicini. On the expressive power of a language for programming coordination media. In *1998 ACM Symposium on Applied Computing (SAC'98)*, pages 169–177, Atlanta (GA), 27 Feb. – 1 Mar. 1998. ACM.

[5] M. Divitini, C. Hanachi, and C. Sibertin-Blanc. Inter-organizational workflows for enterprise coordination. In *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 15, pages 369–398. Springer-Verlag, Mar. 2001.

[6] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *ACM Conference on Organizational Computing Systems*, pages 10–21. ACM, 1995.

[7] K. Fischer, J. Muller, I. Heimig, and A.-W. Scheer. Intelligent agents in virtual enterprises. In PAAM96 [20].

[8] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[9] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communic. of the ACM*, 35(2), 1992.

[10] M. Huns and M. Singh. Social abstractions for information agents. In *Intelligent Information Agents: Agemt-Based Information Discovery and Management on the Internet.* Springer-Verlag, 1999.

[11] S. Jablonski and C. Bussler. *Workflow-Management: Modelling Concepts, Architecture and Implementation.* International Thomson Computer Press, 1996.

[12] N. R. Jennings, P. Farantin, P. Johnson, M.J. O'Brien, and M. Wiegand. Using intelligent agents to manage business processes. In PAAM96 [20], pages 345–360.

[13] G. Kappel, S. Rausch-Scott, and W. Retschitzegger. A framework for workflow management systems based on objects, rules and roles. *ACM Computing Surveys*, 32(1), 2000.

[14] T. Malone and K. Crowstone. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1), 1994.

[15] M. Merz, B. Liberman, and W. Lamersdorf. Using mobile agents to support interorganizational workflow-management. *Applied Artificial Intelligence*, 6(11), 1997.

[16] J. A. Miller, D. Palaniswami, K. J. Sheth, Amit P. Kochut, and H. Singh. Webwork: Meteor2's web-based workflow management system. *Journal of Intelligent Information Systems*, 10(2):185–215, 1998.

[17] A. Omicini. Hybrid coordination models for handling information exchange among Internet agents. In *AI*IA 2000 Workshop "Agenti intelligenti e Internet: teorie, strumenti e applicazioni"*, pages 1–4, Milano (Italy), 13 Sept. 2000.

[18] A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 2001.

[19] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999.

[20] *Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 96)*, Apr. 1996.

[21] G. Picco, A. Fuggetta, and G. Vigna. Understanding code mobility. *IEEE Trans. Soft. Eng.*, 24(5), May 1998.

[22] G. Picco, A. Murphy, and G. C. Roman. LIME: Linda meets mobility. In *21st International Conference on Software Engineering (ICSE'99)*, Los Angeles, May 1999. ACM Press.

[23] A. P. Rocha and E. Oliveira. An electronic market architecture for the formation of virtual enterprises. In *PRO-VE 99 IFIP/PRODNET Conference on Infrastructures for Industrial Virtual Enterprises*, October 1999.

[24] P. Santanu, P. Edwin, and C. Jarir. Essential requirements for a workflow standard. In *Proceeding of OOPSLA 1997 – Business Object Workshop III*. ACM, October 1997.

[25] M. Schumacher. *Objective Coordination in Multi-Agent System Engineering – Design and Implementation*, volume 2039 of *LNAI*. Springer-Verlag, Apr. 2001.

[26] A. P. Sheth and K. J. Kochut. Workflow applications to research agenda: Scalable and dynamic work co-ordination and collaborative systems. In *Adv. in Workflow Management Systems and Interoperability*, Istanbul (Turkey), 1997.

[27] R. Tolksdorf. Models of coordination. In *Engineering Societies in the Agents World*, volume 1972 of *LNAI*, pages 78–92. Springer-Verlag, Dec. 2000.

[28] TuCSoN. `http://lia.deis.unibo.it/rsrc/tucson/`.

[29] The Workflow Management Coalition. `http://www.wfmc.org`.

[30] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[31] H. Wortmann, A. Aerts, N. Szirbik, D. Hammer, and J. Goossenaerts. On the design of a mobile agent web for supporting virtual enterprises. In *WET ICE 2000*, 2000.

Table 2. Book set case

```
% new order is placed
reaction(out(new_order(Customer,BookList,Carrier)),(
    in_r(new_order(Customer,BookList,Carrier)),
    % generate a new transaction ID
    in_r(trans_id_counter(ID)), NextID is ID + 1,
    out_r(trans_id_counter(NextID)),
    % setup order info
    out_r(order_info(ID,Customer,BookList,Carrier)),
    out_r(order_state(ID,ordering)),
    % spawning monitoring agent
    spawn(monitorAgent(ID)),
    % execute parallel buying activities
    out_r(order_all(ID,BookList)),
    out_r(suborder_counter(ID,0)))).
% execute buying activity for each sub order
reaction(out_r(order_all(ID,[(Book,Seller)|L])),(
    in_r(order_all(ID,_)),
    % execute sub-order activity with C as sub ID
    in_r(suborder_counter(ID,C)),
    out_r(suborder_state(ID,C,ordering)),
    spawn(buyerAgent(ID,C,Book,Seller)),
    % iterate for next sub-order
    C1 is C+1,
    out_r(suborder_counter(ID,C1)),
    out_r(order_all(ID,L)))).
% no more sub-order to process
reaction(out_r(order_all(ID,[])),(
    in_r(order_all(ID,[])),
    in_r(suborder_counter(ID,_)))).
% a book is ready at a seller
reaction(out(book_ready(ID,SubCode)),(
    in_r(book_ready(ID,SubCode)),
    % update sub-order state
    in_r(suborder_state(ID,SubCode,ordering)),
    out_r(suborder_state(ID,SubCode,ready)),
    % waiting for other sub-orders?
    out_r(check_order_complete(ID)))).
% not all sub-orders have been completed
reaction(out_r(check_order_complete(ID)),(
    in_r(check_order_complete(ID)),
    rd_r(suborder_state(ID,_,ordering)))).
% all sub-orders processed:  prepare dispatching
reaction(out_r(check_order_complete(ID)),(
    in_r(check_order_complete(ID)),
    no_r(suborder_state(ID,_,ordering)),
    out_r(clear_suborder(ID)))).
% clean information on closed sub-orders
reaction(out_r(clear_suborder(ID)),(
    in_r(clear_suborder(ID)),
    in_r(suborder_state(ID,_,_)),
    out_r(clear_suborder(ID)))).
reaction(out_r(clear_suborder(ID)),(
    in_r(clear_suborder(ID)),
    no_r(suborder_state(ID,_,_)),
    out_r(list_for_carrier(ID,[])),
    rd_r(order_info(ID,Customer,Order,_)),
    out_r(collect_info_for_carrier(ID,Customer,Order)))).
% prepare information for dispatching
reaction(out_r(collect_info_for_carrier(ID,Customer,
        [(Book,Seller)|L])),(
    in_r(collect_info_for_carrier(ID,_,_)),
    in_r(list_for_carrier(ID,List),
    out_r(list_for_carrier(ID,
        [(Book,Seller,Customer)|List])),
    out_r(collect_info_for_carrier(ID,Customer,L)))).
% execute dispatching with collected information
reaction(out_r(collect_info_for_carrier(ID,_,[])),(
    in_r(collect_info_for_carrier(ID,_,[])),
    % update order status
    in_r(order_state(ID,ordering)),
    out_r(order_state(ID,ordered)),
    % execute dispatching activity
    in_r(list_for_carrier(ID,OrderList)),
    rd_r(order_info(ID,_,_,Carrier)),
    spawn(carrierAgent(ID,Carrier,OrderList)))).
```