Formal Specification of Matchmakers, Front-agents, and Brokers in Agent Environments using FSP

Amelia Bădică[1] and Costin Bădică[2]

[1] University of Craiova, Business Information Systems Department
A. I. Cuza 13, Craiova, RO-200585, Romania

[2] University of Craiova, Software Engineering Department
Bvd. Decebal 107, Craiova, RO-200440, Romania

Abstract. The aim of the paper is to precisely characterize types of middle-agents – matchmakers, brokers and front-agents by formally modeling their interactions with requesters and providers. Our approach is based on conceptualizing these interactions by formal specification using FSP process algebra. The main result is the development of formal specification models of middle-agents that help designers and developers by improving their understanding of available types of middle-agents (including matchmakers, front-agents and brokers) and enable the formal analysis of software agent systems that incorporate middle-agents.

1 Introduction

Development of agents that enable dynamic interaction between parties that request and provide information, services or resources requires a careful analysis and understanding of their capabilities. Our recent work in this area was focused on i) implementation of middle-agents for connecting requesters with providers in e-commerce scenarios [2] and ii) formal modeling of middle-agents with the goal to precisely characterize their interactions with other agents in the system [3, 4].

Connecting requesters with providers is a crucial problem in an agent environment and its solution requires the use of middle-agents ([5]). Typical use of middle-agents is encountered in e-commerce. The model agent-based e-commerce system discussed in [2] uses middle-agents to connect user buyers on the purchasing side with shops on the selling side in a distributed marketplace. Each user buyer is represented by a *Client* agent and each shop is represented by a *Shop* agent. User buyer submits an order to the system for purchasing a product via his or her *Client* agent. *Client* agent acts as a *Front-agent* with respect to connecting the user buyer with an appropriate *Shop* agent that provides the requested product. Moreover, *Client* agent uses a special agent called *Client Information Center – CIC* that is responsible for providing information which shop in the system sells which products. So, it can be easily noticed that *CIC* is in fact a *Matchmaker* with respect to connecting *Client* agent with an appropriate *Shop* agent.

In this paper we consider formal models of domain independent interaction patterns between agents involved in intermediation – i.e. we omit in our models i) domain

Bădică A. and Bădică C. (2008).
Formal Specification of Matchmakers, Front-agents, and Brokers in Agent Environments using FSP.
In *Proceedings of the 6th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 9-18
DOI: 10.5220/0001729100090018
Copyright © SciTePress

dependent details of the environment and ii) details of the content languages used for representing requests, responses, preferences, and capabilities. Thus we assume that these interaction patterns are a defining characteristic of each type of middle-agent. In this work we employ our framework firstly proposed in [3]. This framework is utilizing the *finite state process algebra* – FSP ([11]) modeling language.

We start in section 2 with an overview of middle-agents focusing on *Matchmaker*, *Front-agent*, and *Broker*. In section 3 we introduce FSP and some guidelines of modeling agent interactions with FSP. Then we present detailed FSP models of *Matchmaker*, *Front-agent*, and *Broker* middle-agents. We follow in section 4 with experimental evaluation of the models. The last part of the paper contains related work and conclusions.

2 Background on Middle-Agents

The starting point of our work is the classification of middle-agents introduced in seminal work [5]. Based on assumptions about what it is initially known by the requesters, middle-agent, and providers about requester preferences and provider capabilities, authors of [5] proposed 9 types of middle-agents: *Broadcaster*, *Matchmaker*, *Front-agent*, *Anonymizer*, *Broker*, *Recommender*, *Blackboard*, *Introducer*, and *Arbitrator*. Based on literature overview (see survey from [10]) we noticed that frequently utilized middle-agents for connecting provider and requester agents are *Matchmaker*, *Front-agent*, and *Broker*. Additionally, quoting [10], observe that "notions of middle-agents, matchmakers, brokers [...] are used freely in the literature [...] without necessarily being clearly defined". Therefore in this paper we focused on improving this state-of-affair by presenting and discussing formal models of *Matchmaker*, *Front-agent*, and *Broker* middle-agents with the goal of highlighting their similarities and differences. **Matchmaker**. A *Matchmaker* middle-agent assumes that requester preferences are initially known only to the requester, while provider capabilities are initially known to all interaction participants. This means that a provider will have to advertise its capabilities with *Matchmaker* and *Matchmaker* has responsibility to match a request with registered capabilities advertisements. However, the fact that provider capabilities are initially known also by the requester means that the result of the matching (i.e set of matching providers) is returned by *Matchmaker* to requester (so provider capabilities become thus known to the requester), and the choice of the matching provider is the responsibility of the requester. Consequently the transaction is not intermediated by *Matchmaker*, as would be the case for example with a *Broker* or *Front-agent*.

Front-agent assumes that requester preferences are initially known only to the requester, while provider capabilities are initially known both to provider and middle-agent. This means that a provider will have to advertise its capabilities with *Front-agent* and *Front-agent* is responsible to match a request with registered capabilities advertisements. Additionally, as the provider capabilities are not initially known to the requester, *Front-agent* also has the responsibility of intermediating the transaction between the requester and the matching provider (this is why often this type of middle-agent is called *Broker* rather than *Front-agent*; in our opinion a true *Broker* is different, see below).

Broker assumes that requester preferences are initially known only to the requester and the middle-agent and provider capabilities are initially known only to the provider

and the middle-agent. The crucial point is that, however, requester preferences are not initially known to the provider and provider capabilities are not initially known to the requester. This means that a *Broker* will truly intermediate transactions between providers and requesters in *both* directions: i) if a requester submits a request either it cannot be matched and it is registered with the *Broker* or it is matched with a provider capability and then transaction is intermediated by the *Broker*, and ii) if a provider advertises a capability then capability is registered with the *Broker* and also capability is matched against registered requests; neither match is found and nothing more happens or matches are found and corresponding transactions are intermediated by the *Broker*.

3 FSP Models

Overview of FSP. FSP is an algebraic specification technique of concurrent and cooperating processes that allows a compact representation of a finite state labeled transition system (LTS hereafter), rather than describing it as a list of states and transitions.

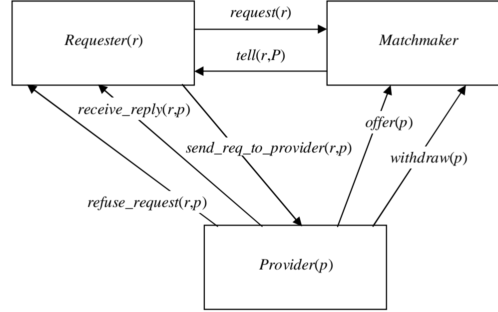
A FSP model consists of a finite set of sequential and/or composite process definitions. Additionally, a sequential process definition consists of a sequence of one or more definitions of local processes. A process definition consists of a process name associated to a process term. FSP uses a rich set of constructs for process terms (see [11] for details). For the purpose of this paper we are using the following constructs: action prefix ($a \rightarrow P$), nondeterministic choice ($P|Q$), and process alphabet extension ($P + \{a_1, \ldots, a_n\}$) for sequential process terms and parallel composition ($P\|Q$) and relabeling ($P/\{new_1/old_1, \ldots, new_k/old_k\}$) for composite process terms. The modeling that we propose here follows the general guidelines outlined in [3]. Briefly: i) agents are modeled as FSP processes and a multi-agent system is modeled as a parallel composition of processes; ii) sets $\mathcal{R}$ of requesters and $\mathcal{P}$ of providers are assumed to be initially given and agent requests and replies are indexed with requester and/or provider identifiers; iii) matching operation is modeled as a relation $\mathcal{M} \subseteq \mathcal{R} \times \mathcal{P}$.

We assume that our system contains providers, requesters and a middle-agent. So in our models we have *Provider*, *Requester* and middle-agent FSP processes. Additionally we assume that *Provider* offers are reproducible and *Requester* requests are non-reproducible (see [3] for a more detailed discussion on this topic).

***Matchmaker* Middle-Agent in FSP.** The block diagram and the FSP specification of a system with requesters, providers and a *Matchmaker* are shown in figure 1.

Provider agent registers its capability offer (action $offer$) with the *Matchmaker* and then enters a loop where it receives requests from *Requester* agents via action $receive_request$ and processes and replies accordingly via action $send_reply$. Note that a *Provider* can also withdraw a registered capability offer and while its capability is not registered it always refuses to serve a request (action $refuse_request$).

Requester agent submits a request to the *Matchmaker* (action $send_request$) and then waits for a reply. The *Matchmaker* replies with a set of matching providers (action *tell* with argument $\mathcal{M}(r) \cap P$ representing the set of matches; here P is the set of registered providers and $\mathcal{M}(r)$ is the set of matching providers). Next *Requester* has the option to choose what provider from set P to contact for performing the service (ac-



$$
\begin{array}{lcl}
\mathit{Provider} & = & (\mathit{offer} \rightarrow \mathit{ProcessRequest}\,| \\
 & & \mathit{receive_request} \rightarrow \mathit{refuse_request} \rightarrow \mathit{Provider}), \\
\mathit{ProcessRequest} & = & (\mathit{receive_request} \rightarrow \mathit{send_reply} \rightarrow \mathit{ProcessRequest}\,| \\
 & & \mathit{withdraw} \rightarrow \mathit{Provider}). \\
 & & \\
\mathit{Requester} & = & (\mathit{send_request} \rightarrow \mathit{WaitReply}), \\
\mathit{WaitReply} & = & (\mathit{tell}(P \subseteq \mathcal{P}) \;\rightarrow\; \textbf{if } P \neq \emptyset \textbf{ then } \mathit{ContactProvider}(P) \textbf{ else } \mathit{Requester}), \\
\mathit{ContactProvider}(P \subseteq \mathcal{P}) & = & (\textbf{while } P \neq \emptyset\ \mathit{send_request_to_provider}(p \in P) \rightarrow \\
 & & \{\mathit{receive_reply}(p), \mathit{refuse_request}(p)\} \rightarrow \mathit{Requester}). \\
 & & \\
\mathit{Matchmaker} & = & \mathit{Matchmaker}(\emptyset), \\
\mathit{Matchmaker}(P \subseteq \mathcal{P}) & = & (\mathit{request}(r \in \mathcal{R}) \rightarrow \mathit{MatchReq}(r, P)\,| \\
 & & \mathit{offer}(p \in \mathcal{P} \setminus P) \rightarrow \mathit{Matchmaker}(P \cup \{p\})\,| \\
 & & \mathit{withdraw}(p \in P) \rightarrow \mathit{Matchmaker}(P \setminus \{p\})), \\
\mathit{MatchReq}(r \in \mathcal{R}, P \subseteq \mathcal{P}) & = & (\mathit{tell}(r, \mathcal{M}(r) \cap P) \rightarrow \mathit{Matchmaker}(P)) + \{\mathit{tell}(r' \in \mathcal{R}, P' \subseteq \mathcal{P})\}. \\
 & & \\
\mathit{Requester}(r \in \mathcal{R}) & = & \mathit{Requester}/\{\mathit{request}(r)/\mathit{send_request}, \mathit{tell}(r, P \subseteq \mathcal{P})/\mathit{tell}(P), \\
 & & \mathit{send_request_to_provider}(r, p \in \mathcal{P})/\mathit{send_request_to_provider}(p), \\
 & & \mathit{receive_reply}(r, p \in \mathcal{P})/\mathit{receive_reply}(p), \mathit{refuse_request}(r, p \in \mathcal{P})/\mathit{refuse_request}(p)\}. \\
\mathit{Provider}(p \in \mathcal{P}) & = & \mathit{Provider}/\{\mathit{offer}(p)/\mathit{offer}, \\
 & & \mathit{send_request_to_provider}(r \in \mathcal{R}, p)/\mathit{receive_request}, \\
 & & \mathit{receive_reply}(r \in \mathcal{R}, p)/\mathit{send_reply}, \mathit{refuse_reply}(r \in \mathcal{R}, p)/\mathit{refuse_reply}\}. \\
\mathit{System} & = & \mathit{Matchmaker} \,\|\, (\|_{r \in \mathcal{R}} \mathit{Requester}(r)) \,\|\, (\|_{p \in \mathcal{P}} \mathit{Provider}(p)).
\end{array}
$$

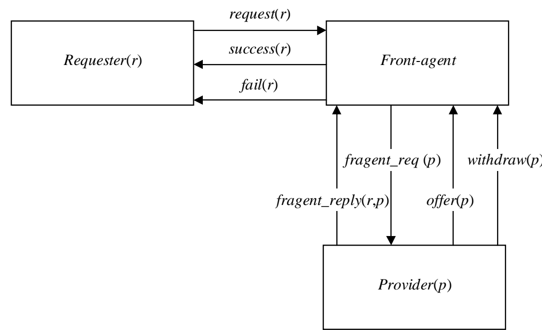
Fig. 1. System with *Matchmaker* middle-agent.

tion *send_request_to_provider* with argument $p \in P$ representing the chosen provider). Finally, *Requester* waits for a reply from the contacted provider (action *receive_reply*).

Matchmaker agent registers and deregisters *Provider* offers and answers *Requester* requests for matching offers. *Matchmaker* informs *Requester* about available registered offers (action *tell*). Note that *Requester* is responsible to choose an appropriate matching offer from the available matching offers (action *send_request_to_pro- vider*). This complicates a bit *Requester* behavior as compared with *Front-agent* and *Broker* cases.

Special care is taken to accurately model agent communication using FSP synchronization. *Matchmaker* model requires alphabet extension (construct $\{tell(r' \in R, P' \subseteq \mathcal{P})\}$ in figure 1) to model correctly communication between *Matchmaker* and *Requester*.

A critical situation may occur when a matching offer is found but the matching *Provider* chooses to cancel its offer by deregistering it with the *Matchmaker* before it is actually contacted by the *Requester*. This ability of the *Provider* is modeled with action *refuse_request*. Note that this situation cannot occur with the *Front-agent* and *Broker* (remember that both *Front-agent* and *Broker* intermediate the request on behalf of the *Requester*) so we did not have to model this ability of the *Provider* in those cases.



$$
\begin{aligned}
&Provider && = (offer \rightarrow ProcessRequest),\\
&ProcessRequest && = (receive_request \rightarrow send_reply \rightarrow ProcessRequest \mid\\
& && \quad withdraw \rightarrow Provider).\\[6pt]
&Requester && = (send_request \rightarrow WaitReply),\\
&WaitReply && = (\{sucess, fail\} \rightarrow Requester).\\[6pt]
&Frontagent && = Frontagent(\emptyset),\\
&Frontagent(P) && = (request(r \in R) \rightarrow ResolveReq(r, P) \mid\\
& && \quad offer(p \in \mathcal{P} \setminus P) \rightarrow Frontagent(P \cup \{p\}) \mid\\
& && \quad withdraw(p \in P) \rightarrow Frontagent(P \setminus \{p\})),\\
&ResolveReq(r, P) && = (\textbf{if } \mathcal{M}(r) \cap P = \emptyset \textbf{ then } fail(r) \rightarrow Frontagent(P)\\
& && \quad \textbf{else } ContactProvider(r, \mathcal{M}(r) \cap P, P)\\
&ContactProvider(r, P', P) && = (\textbf{while } P' \neq \emptyset\ fragent_req(p \in P') \rightarrow fragent_reply(p) \rightarrow\\
& && \quad success(r) \rightarrow Frontagent(P)).\\[6pt]
&Requester(r \in \mathcal{R}) && = Requester/\{request(r)/send_request, fail(r)/fail, success(r)/success\}.\\
&Provider(p \in \mathcal{P}) && = Provider/\{offer(p)/offer, fragent_req(p)/receive_request, fragent_reply(p)/send_reply,\\
& && \quad withdraw(p)/withdraw\}.\\
&System && = Frontagent \parallel (\parallel_{r \in \mathcal{R}} Requester(r)) \parallel (\parallel_{p \in \mathcal{P}} Provider(p)).
\end{aligned}
$$

Fig. 2. System with *Front-agent* middle-agent.

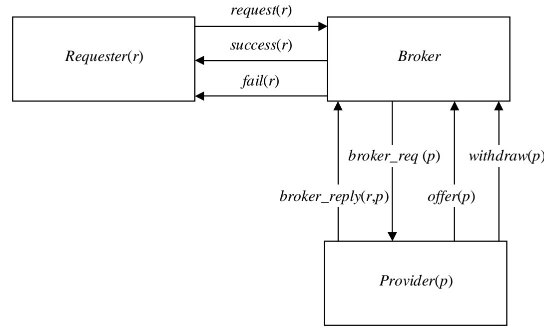
***Front-agent* Middle-Agent in FSP.** The block diagram and the FSP specification of a system composed of requesters, providers and a *Front-agent* are shown in figure 2.

Provider agent is similar to *Matchmaker* case. *Requester* agent is simpler than *Matchmaker* case: it submits a request to *Front-agent* (action *send_request*) and then waits for *Front-agent* to either resolve that request (action *success*) or fail (action *fail*).

Front-agent agent processes requests from *Requester* agents and registers offers from *Provider* agents. Note that, differently from the *Matchmaker*, *Front-agent* has the responsibility to choose an appropriate matching provider from the available matching providers $\mathcal{M}(r) \cap P$ (here P is the set of registered providers and $\mathcal{M}(r)$ is the set of matching providers) using action *fragent_request* and to resolve the request (action *fragent_request*). Finally the result is passed to the *Requester* agent using action *success*. In conclusion, the actual *Provider* that fulfils the request for the *Requester* on behalf of the *Front-agent* is hidden from the *Requester* that issued the request.

***Broker* Middle-Agent in FSP.** The block diagram and the FSP specification of a system composed of requesters, providers and a *Front-agent* are shown in figure 2. *Provider* and *Requester* agents are similar to the *Front-agent* case.

Broker processes requests from *Requesters* and processes and registers offers from *Providers*. If arequest can be served based on available matching offers then *Broker* behaves similarly with *Front-agent*. Differently from *Front-agent*, if a request cannot be



$$
\begin{aligned}
&\mathit{Provider} &&= (\mathit{offer} \rightarrow \mathit{ProcessRequest}),\\
&\mathit{ProcessRequest} &&= (\mathit{receive_request} \rightarrow \mathit{send_reply} \rightarrow \mathit{ProcessRequest}\,|\\
& &&\quad \mathit{withdraw} \rightarrow \mathit{Provider}).\\[6pt]
&\mathit{Requester} &&= (\mathit{send_request} \rightarrow \mathit{WaitReply}),\\
&\mathit{WaitReply} &&= (\{\mathit{sucess}, \mathit{fail}\} \rightarrow \mathit{Requester}).\\[6pt]
&\mathit{Broker} &&= \mathit{Broker}(\emptyset, \emptyset),\\
&\mathit{Broker}(P, R) &&= (\mathit{request}(r \in \mathcal{R} \setminus R) \rightarrow \mathit{ResolveReq}(r, R, P)\,|\\
& &&\quad \mathit{offer}(p \in \mathcal{P} \setminus P) \rightarrow \mathit{ResolveOff}(p, R, P)\,|\\
& &&\quad \mathit{withdraw}(p \in P) \rightarrow \mathit{Broker}(R, P \setminus \{p\})\,|\\
& &&\quad \textbf{while } R \neq \emptyset\ \mathit{fail}(r \in \mathcal{R}) \rightarrow \mathit{Broker}(R \setminus \{r\}, P)),\\
&\mathit{ResolveReq}(r, R, P) &&= (\textbf{if } \mathcal{M}(r) \cap P = \emptyset \textbf{ then } \mathit{Broker}(R \cup \{r\}, P)\\
& &&\quad \textbf{else } \mathit{ContactProviderReq}(r, R, \mathcal{M}(r) \cap P, P)),\\
&\mathit{ContactProviderReq}(r, R, P', P) &&= (\textbf{while } P' \neq \emptyset\ \mathit{broker_req}(p \in P') \rightarrow \mathit{broker_reply}(p) \rightarrow \mathit{success}(r) \rightarrow \mathit{Broker}(R, P)),\\
&\mathit{ResolveOff}(p, R, P) &&= (\textbf{if } \mathcal{M}^{-1}(p) \cap R = \emptyset \textbf{ then } \mathit{Broker}(R, P \cup \{p\})\\
& &&\quad \textbf{else } \mathit{ContactProviderOff}(p, \mathcal{M}^{-1}(p) \cap R, R, P)),\\
&\mathit{ContactProviderOff}(p, R', R, P) &&= (\textbf{if } R' \neq \emptyset \textbf{ then } \mathit{broker_req}(p) \rightarrow\\
& &&\quad \mathit{broker_reply}(p) \rightarrow \mathit{success}(r \in R') \rightarrow \mathit{ContactProviderOff}(p, R' \setminus \{r\}, R \setminus \{r\}, P)\\
& &&\quad \textbf{else } \mathit{Broker}(R, P)).\\[6pt]
&\mathit{Requester}(r \in \mathcal{R}) &&= \mathit{Requester}/\{\mathit{request}(r)/\mathit{send_request}, \mathit{fail}(r)/\mathit{fail}, \mathit{success}(r)/\mathit{success}\}.\\
&\mathit{Provider}(p \in \mathcal{P}) &&= \mathit{Provider}/\{\mathit{offer}(p)/\mathit{offer}, \mathit{fragent_req}(p)/\mathit{receive_request},\\
& &&\quad \mathit{fragent_reply}(p)/\mathit{send_reply}, \mathit{withdraw}(p)/\mathit{withdraw}\}.\\
&\mathit{System} &&= \mathit{Broker} \parallel (\parallel_{r \in \mathcal{R}} \mathit{Requester}(r)) \parallel (\parallel_{p \in \mathcal{P}} \mathit{Provider}(p)).
\end{aligned}
$$

Fig. 3. System with *Broker* middle-agent.

served based on currently registered offers then, rather than reporting failure, the request is recorded until: i) either a new matching offer is registered with the *Broker* or ii) the request is deemed failed. Note that when a new offer is registered the *Broker* determines the set of recorded (i.e. not yet served) matching requests – $\mathcal{M}^{-1}(p) \cap R$ ($\mathcal{M}^{-1}(p)$ is the set of matching requesters and R is the set of recorded requesters) and serves them using the matching provider p – $\mathit{ContactProviderOff}$ sub-process in figure 3.

System Properties. A basic desirable property of systems with middle-agents is that they are free of deadlocks. The result is formally stated as follows.

Proposition 1. *The systems with Matchmaker, Front-agent and Broker shown in figures 1, 2 and 3 are deadlock free.*

Proof. We consider the *Matchmaker* system proof (other proofs follow the pattern).

Let us consider an arbitrary system state S. As the system is a parallel composition of processes, state S is composed of sub-states corresponding to the *Matchmaker* and to each of the *Provider* an *Requester* processes. Any progress from S will be caused

by an interaction between two processes: *Matchmaker* with a *Provider*, *Matchmaker* with a *Requester* or a *Requester* with a *Provider*. Note that *Matchmaker* can be in one of two states: i) $Matchmaker(P)$ with $P \subseteq \mathcal{P}$; ii) $MatchReq(r, P)$ with $r \in \mathcal{R}$ and $P \subseteq \mathcal{P}$.

In the first case it means that *Matchmaker* finalized to process a request and it is waiting for a new one. If a new request is available we are done. If a *Provider* is available to register/deregister a capability offer, we are are again done. Otherwise it means that all requesters submitted requests to providers and wait for service. In this case we pick randomly a pair $Requester(r)$ and $Provider(p)$ with $r \in \mathcal{R}$ and $p \in \mathcal{P}$ and system will proceed with interaction between $Requester(r)$ and $Provider(p)$.

In the second case progress will occur through interaction between *Matchmaker* and $Requester(r)$, as $Requester(r)$ is definitely in state *WaitReply*.

4 Experimental Results

We conducted a series of experiments to check correctness of our models introduced in section 3. As a side effect we have also recorded the size of the state model expressed as number of states and transitions, depending on the number of requesters and providers.
Experimental Setup. Firstly we had to express the general models shown in figures 1, 2 and 3 using the FSP language supported by the LTSA tool [11]. The main difficulty is encoding of processes indexed with sets in the language supported by LTSA.

Assuming that $\mathcal{S}$ is a set with n elements, mapping of processes indexed with sets and/or set elements to the FSP notation supported by LTSA follows the guidelines: i) an index $s \in \mathcal{S}$ is encoded as $[s]$; ii) an index $S \subseteq \mathcal{S}$ is encoded as $[s_1] \ldots [s_n]$ s.t. $s_i = 1$ if $i \in S$ and $s_i = 0$ if $i \notin S$. For example $ResolveReq(2, \{1\}, \{1, 3\})$ is mapped to `ResolveReq[2][1][0][1][0][1]` and $tell(1, \{2, 3\})$ is mapped to `tell[1][0][1][1]`.

Note that a mapping can be defined such that the size of resulting FSP specification is linear in the product of number of providers with number of requesters[3]. This is an important desiderata to make the resulting FSP specification of a practical value.

Proposition 2. *Let $m = |\mathcal{R}|$ and $n = |\mathcal{P}|$. The systems with Matchmaker, Front-agent and Broker shown in figures 1, 2 and 3 can be mapped to the FSP language supported by LTSA tool such that size of resulting specification is $O(m \times n)$.*

Proof. First note that mapping of set indexed names of processes and actions produces new names of size linear with m and n..

Second, application of the following mapping rules produces parts of FSP specification that clearly have a size $O(m \times n)$.

If $Proc(S \subseteq \mathcal{S})$ is a set indexed processes and $|\mathcal{S}| = n$ then the construct:

$$Proc(S \subseteq \mathcal{S}) = (\textbf{while}\ S \neq \emptyset\ action(s \in S) \ldots)$$

is mapped to (here assuming $n = 3$):

[3] Do not confuse size of FSP specification (i.e. size of FSP code measured for example as the number of nodes of its syntax tree) with size of LTS corresponding to this specification.

Table 1. LTS size of the system with *Matchmaker* middle-agent.

# requesters	# providers	# states	# transitions	# states after minimization
2	3	608	2272	376
3	4	3392	14720	2064
4	5	166400	1010176	N/A
5	6	> 1000000	> 7000000	N/A

```
Proc[s1:0..1][s2:0..1][s3:0..1] = (
  while s1 == 1 action[1] ... |
  while s2 == 1 action[2] ... |
  while s3 == 1 action[3] ...)
```

Note that the size of the resulted specification is $O(n)$.

If $Proc_i(S \subseteq \mathcal{S})$, $i = 1, 2$ are set indexed processes and $|\mathcal{S}| = n$ then the construct:

$$Proc_1(S \subseteq \mathcal{S}) = (\textbf{while } S \neq \emptyset \; Proc_2(S) \ldots)$$

is mapped to (here assuming $n = 3$):

```
Proc1[s1:0..1][s2:0..1][s3:0..1] = (
  if s1 == 1 || s2 == 1 || s3 == 1
  then Proc2[s1:0..1][s2:0..1][s2:0..1] ...)
```

Note that the size of the resulted specification is $O(n)$.

If $|\mathcal{S}_1| = m$ and $|\mathcal{S}_2| = n$ are sets, $\mathcal{M} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is a relation then the construct:

$$Proc_1(s \in \mathcal{S}_1, S \subseteq \mathcal{S}_2) = (\textbf{if } \mathcal{M}(s) \cap S = \emptyset \textbf{ then } \ldots \textbf{ else } Proc_2(\mathcal{M})s) \cap S) \ldots)$$

is mapped to (here assuming $m = 2$, $n = 3$ and $\mathcal{M} = \{(1, 2), (1, 3), (2, 1), (2, 2)\}$):

```
Proc1[s:1..2][s1:0..1][s2:0..1][s3:0..1] = (
  if s == 1 then
     if s2 == 0 && s3 == 0 then ...
     else Proc2[0][s2][s3] ...
  else if s == 2 then ...
     if s1 == 0 && s2 == 0 then ...
     else Proc2[s1][s2][0] ...)
```

Note that the size of the resulted specification is $O(m \times n)$.

Results and Discussion. In the experiments we considered 3 systems composed of: i) n requesters and $n + 1$ providers, $2 \leq n \leq 5$; ii) 1 middle-agent per system (*Matchmaker*, *Front-agent* and respectively *Broker*), and iii) the matching relation $\mathcal{M} = \{(i, i + 1)|1 \leq i \leq n\} \cup \{(i, i + 2)|1 \leq i \leq n - 1\} \cup \{n, 1)\}$[4].

We utilized LTSA tool to analyze the resulting FSP models of systems with *Matchmaker*, *Front-agent* and *Broker* middle-agents. All the models are free of deadlocks (thus confirming theoretical results). Sizes in terms of number of states and transitions of their corresponding labeled transition systems are presented in tables 1, 2 and 3.

[4] FSP models used in experiments are available at `http://software.ucv.ro/~badica_costin/fsp/msvveis08_models.zip`

SciTePress Science and Technology Publications

Table 2. LTS size of the system with *Front-agent* middle-agent.

# requesters	# providers	# states	# transitions	# states after minimization
2	3	56	92	52
3	4	160	268	148
4	5	416	704	384
5	6	1024	1744	944

Table 3. LTS size of the system with *Broker* middle-agent.

# requesters	# providers	# states	# transitions	# states after minimization
2	3	293	464	281
3	4	1788	2997	1718

5 Related Work

Middle-agents were recently put forward in [7] in the context of applications of agents in e-commerce. [7] covers in some detail *Matchmaker*, *Broker*, *Broadcaster* and *Recommender* with a focus on interaction protocols and languages for describing provider capabilities. However, while *Matchmaker* description fits within our model, note that *Broker* described by [7] actually corresponds to our model of a *Front-agent*.

Our literature review indicates that while there is a growing interest in the subject of "middle-agents", only a few papers address the problem of a concise definition of middle-agents in terms of their interaction capabilities.

For example [1] suggests the use of LTSs for modeling agent types without providing definitions of middle-agents. [10] shows only models of matchmakers and brokers using input/output automata (following initial proposal in [14]), while [9] informally describes interactions of middle-agents with requesters and providers as sequences of message exchanges presented in natural language. Rather, many references to middle-agents focus in more or less detail on various applications and systems that use different types of middle-agents, most often brokers and/or matchmakers (see overview in [10]) and highlighting implementation and/or performance issues.

Some recent works propose the use of process algebras for concisely modeling middle-agent interactions. Paper [3] introduces a general framework for FSP modeling of middle-agents and shows how this framework can be used to model a *Recommender* agent. Using the same idea, paper [4] presents a model of an *Arbitrator* middle-agent for coordination of participants in single-item English auctions. Our present work is an extension of [3, 4] for modeling *Matchmaker*, *Front-agent* and *Broker* middle-agents, including both theoretical and experimental results.

Our "agents-as-processes" modeling approach is not entirely new. Paper [6] proposes the use of π-calculus ([12]) and shows models of a prototype agent system LOGOS for an unattended grounds operation-center using a fault resolution scenario. However, while this paper shows how to use a software tool for checking the proposed specifications, it does not provide any experimental results – only general guidelines are given. Paper [13] focuses also on the LOGOS agent system, but using a CSP [8]. While this paper discusses some benefits of the method (detecting race conditions, message omissions, and better understanding of the system) it still does not provide any experimental results and neither discusses the practical problems encountered.

6 Conclusions and Future Work

In this paper we applied a formal framework based on FSP process algebra for modeling a system that contains requesters, providers and middle-agents. We considered three types of middle-agents: *Matchmaker*, *Front-agent*, and *Broker*. For each system we checked the resulting model with the help of LTSA analysis tool.

We also identified some paths for continuing this research: i) modeling of more complex systems containing more types of middle-agents; ii) introduction and analysis of qualitative properties of agent systems.

References

1. Alagar, V., Holliday, J.: Agent types and their formal descriptions. Technical Report COEN-2002-09-19A, Santa Clara University. Computer Engineering Department, (2002).
2. Bădică, C., Ganzha, M., Paprzycki, M.: Developing a Model Agent-based E-Commerce System. In: *E-Service Intelligence: Methodologies, Technologies and Applications*, Studies in Computational Intelligence 37, Springer, (2007) 555–578.
3. Bădică, A., Bădică, C., Liţoiu, L.: Middle-Agents Interactions as Finite State Processes: Overview and Example: In *Proc.*16th *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2007)*, (2007) 12–17.
4. Bădică, A., Bădică, C.: Formal modeling of agent-based english auctions using finite state process algebra. In: N. Nguyen et al. (Eds.): *Agent and Multi-Agent Systems: Technologies and Applications. Proc. KES-AMSTA'2007*, LNAI 4496, Springer (2007) 248–257.
5. Decker, K., Sycara, K. P., and Williamson, M.: Middle-agents for the internet. In: *Proceedings of the* 15th *Int.Joint Conf.on Artif.Intel. IJCAI'97*, vol.1, Morgan Kaufmann, (1997) 578–583.
6. Esterline, A., Rorie, T., and Homaifar, A.: A Process-Algebraic Agent Abstraction. In: Rouff, C.A. et al. (Eds.): *Agent Technology from a Formal Perspective*, NASA Monographs in Systems and Software Engineering, Springer (2006) 88–137.
7. Fasli, M.: *Agent Technology For E-Commerce*. Wiley, (2007).
8. Hoare, C.A.R.: *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science, Hemel Hempstead. (1985).
9. Hristozova, M., Lister, K., and Sterling, L.: Middle-agents – towards theoretical standardization. In: I. J. Timm, M. Berger, S. Poslad, and S. Kirn (Eds.): *Proc. of the Int. Workshop on Multi-Agent Interoperability – MAI'02.* 25th *German Conference on Artif.Intel. (KI'2002)*, `http://www.informatik.uni-bremen.de/agki/www/astap02/mai02-ws.pdf`, (2002) 65–80.
10. Klusch. M., Sycara, K.P.: Brokering and matchmaking for coordination of agent societies: A survey. In Omicini, A., Zambonelli, F., Klusch, M., and Tolksdorf, R. (Eds.): *Coordination of Internet Agents. Models, Technologies, and Applications*, Springer (2001) 197–224.
11. Magee, J., Kramer, J.: *Concurrency. State Models and Java Programs (2nd ed.)*. John Wiley & Sons (2006).
12. Milner, R. Communicating and Mobile Systems: The π-calculus, Cambridge University Press, Cambridge. (1999).
13. Rouff, C., Rash, J., Hinchey, M., and Truszkowski, W.: Formal Methods at NASA Goddard Space Flight Center. In: Rouff, C.A. et al. (Eds.): *Agent Technology from a Formal Perspective*, NASA Monographs in Systems and Software Engineering, Springer (2006) 287–309.
14. Wong, H.C., Sycara, K.: A taxonomy of middle-agents for the internet. In *Proceedings of the* 4th *International Conference on MultiAgent Systems (ICMAS-2000)*, Washington, DC, USA, IEEE Computer Society. (2000) 465–466. An extended version of the paper is available at `http://www.cs.cmu.edu/~softagents/papers/ExtMiddleAgentsICMAS.pdf`.