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

Amelia Bădică and Costin Bădică

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
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 \(R\) of requesters and \(P\) 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 \(M \subseteq R \times P\).

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 \(M(r) \cap P\) representing the set of matches; here \(P\) is the set of registered providers and \(M(r)\) 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-
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 Requester. This ability of the Provider is modeled with action refuse_request. Note that this situation cannot occur with the 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.

Fig. 1. System with Matchmaker middle-agent.

sender 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_provider). 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 \( \text{tell}(r' \in R, P' \subseteq P) \)) in figure 1 to model correctly communication between Matchmaker and Requester.

```
Provider = (offer → ProcessRequest)
            | receive_request → refuse_request → Provider).
ProcessRequest = (receive_request → send_reply → ProcessRequest) | withdraw → Provider).
Requester = (send_request → WaitReply).
WaitReply = (tell \( p \in P \) → if \( P \neq \emptyset \) then ContactProvider(\( P \) else Requester),
ContactProvider(\( P \subseteq P \)) = (while \( P \neq \emptyset \) send_request, \( p \) \( \in P \) →
            \{receive_reply(\( p \), refuse_request(\( p \)) → Requester).
Matchmaker = Matchmaker(\( 0 \)),
Matchmaker(\( P \subseteq P \) = (request(\( r \in R \) MatchReq(\( P \)
            offer(\( p \in P \) \( P \) → Matchmaker(\( P \cup \{ p \} \)
            | withdraw(\( p \in P \) → Matchmaker(\( P \setminus \{ p \} \))).
MatchReq(\( r \in R, P \subseteq P \)) = (tell(\( r, M(\( r \) \( P \) → Matchmaker(\( P \) + tell(\( r' \in R, P' \subseteq P \))
Requester(\( r \in R \) = Requester(\( \{ \text{request}(\( r \) send_request, \( p \in P \) \( P \) tell(\( p \),
            send_request, \( p \) \( \in P \) ) → send_request, \( p \) \( \in P \) | receive_reply(\( p \), refuse_request(\( p \) ) → Requester).
Provider(\( p \in P \) = Provider(\( \{ \text{offer}(\( p \) \( P \) \( P \) ) \text{offer},
            send_request, \( p \) \( \in P \) ) → send_request, 
            receive_reply(\( p \), refuse_reply(\( r \in R, p \) ) refuse_reply).
System = Matchmaker(\( \{ \text{request}(\( r \) (\( R \) Provider(\( p \) )

```
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 \( M(r) \cap P \) (here \( P \) is the set of registered providers and \( M(r) \) is the set of matching providers) using action \( \text{fragent}_{request} \) and to resolve the request (action \( \text{fragent}_{reply} \)). Finally the result is passed to the Requester agent using action success. In conclusion, the actual Provider that fulfills 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 a request can be served based on available matching offers then Broker behaves similarly with Front-agent. Differently from Front-agent, if a request cannot be
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 and Requester processes. Any progress from $S$ will be caused 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 $p$ – ContactProviderOff $f$ 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.
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 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 \( S \) 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 \( s \in S \) is encoded as \( [s] \); ii) an index \( S \subseteq S \) 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 \( \text{ResolveReq}(2, \{1\}, \{1, 3\}) \) is mapped to \( \text{ResolveReq}[2][1][0][1][0][1] \) and \( \text{tell}(1, \{2, 3\}) \) is mapped to \( \text{tell}[1][0][1][1] \).

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

\[
\text{Proc}(S \subseteq \mathcal{S}) = (\text{while } S \neq \emptyset \text{ action}(x \in S) \ldots)
\]

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

\[\ldots\]

\(^3\)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.

<table>
<thead>
<tr>
<th># requesters</th>
<th># providers</th>
<th># states</th>
<th># transitions</th>
<th># states after minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>608</td>
<td>2272</td>
<td>376</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3392</td>
<td>14720</td>
<td>2064</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>166400</td>
<td>1010176</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>&gt;1000000</td>
<td>&gt;7000000</td>
<td>N/A</td>
</tr>
</tbody>
</table>

\[
\text{Proc}[s1:0..1][s2:0..1][s3:0..1] = ( \\
\text{while } s1 == 1 \text{ action}[1] ... | \\
\text{while } s2 == 1 \text{ action}[2] ... | \\
\text{while } s3 == 1 \text{ action}[3] ...) \\
\]

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

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

\[
\text{Proc}_i(S \subseteq S) = (\text{while } S \neq \emptyset \text{ Proc}_i(S)...) \\
\]

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

\[
\text{Proc1}[s1:0..1][s2:0..1][s3:0..1] = ( \\
\text{if } s1 == 1 || s2 == 1 || s3 == 1 \\
\text{then Proc2}[s1:0..1][s2:0..1][s3:0..1] ...) \\
\]

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

If \(|S_1| = m\) and \(|S_2| = n\) are sets, \(M \subseteq S_1 \times S_2\) is a relation then the construct:

\[
\text{Proc}_1(s \in S_1, S \subseteq S_2) = (\text{if } M(s) \cap S = \emptyset \text{ then } ... \text{ else } \text{Proc}_2(M(s) \cap S)...) \\
\]

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

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

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 \(M = \{(i, i + 1)|1 \leq i \leq n| \cup \{(i, i + 2)|1 \leq i \leq n - 1\} \cup \{n, 1\}\}^4\).

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.

\[\text{FSP models used in experiments are available at } \text{http://software.ucv.ro/~badica_costin/fsp/msvveis08_models.zip}\]
Table 2. LTS size of the system with Front-agent middle-agent.

<table>
<thead>
<tr>
<th># requesters</th>
<th># providers</th>
<th># states</th>
<th># transitions</th>
<th># states after minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>36</td>
<td>26</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>160</td>
<td>268</td>
<td>128</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>416</td>
<td>704</td>
<td>384</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1024</td>
<td>1744</td>
<td>944</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th># requesters</th>
<th># providers</th>
<th># states</th>
<th># transitions</th>
<th># states after minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>293</td>
<td>464</td>
<td>281</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1788</td>
<td>2997</td>
<td>1718</td>
</tr>
</tbody>
</table>

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 $\pi$-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


