

Adaptive Agent-based Service Composition for Wireless Terminals

Sasu Tarkoma¹ and Mikko Laukkanen²

¹ Helsinki Institute for Information Technology
P.O. Box 9800, FIN-02015 HUT, Finland
sasu.tarkoma@hiit.fi

² TeliaSonera Finland
P.O.Box 970 (Teollisuuskatu 13)
FIN-00051 SONERA
+358 40 5073358
mikko.laukkanen@teliasonera.com

Abstract. Software agents are one of the building blocks of ambient intelligence and pervasive computing. Adaptation to changes in the execution context is necessary in order to provide continuous and smooth provision of services for wireless clients. In this paper, we present a system for adapting a service consisting of multiple agents based on application, terminal and communication models and profiles. The system adapts and composes the service by finding the best combination of local and external agents to contact. The context models are based on experimentation with actual devices and environments. We present a theoretical cost model for runtime service composition, and examine experimental results that are based on an example scenario.

1 Introduction

Software agents are one of the building blocks of ambient intelligence and pervasive computing [13]. Agents are autonomous and proactive entities that are capable of reacting and adapting to changes in their environment. Agent technology relies on asynchronous messaging that is used in order to facilitate agent communication in multi-agent systems [6]. Therefore, improving messaging and minimizing communication latency are key requirements for improving the performance and response time of agent-based services and applications. This optimisation requires that agents adapt to their communication context, and react to changes perceived in the communication link, message transport and the agent execution environment.

Adaptation to changes in the execution context is necessary in order to provide continuous and smooth provision of services for wireless clients. In this paper, we present a system for adapting a service that consists of multiple agents for wireless clients. The adaptation mechanism uses application, terminal and communication models that allow the system to examine different usage strategies for contacting the agents of the service. The system adapts the service by finding the best combination

of local and external agents to contact. Local agents are available on the client terminal, and external agents reside on fixed-network systems, and require the use of a communication mechanism that may or may not be limited in terms of bandwidth and latency. The context models are based on experimentation with devices and environments. We present a theoretical cost model for service composition. We have focused on the dynamic service composition or partitioning decision, whether to request part of a service from a local agent or an external agent. Mobile agents or agent download is not considered, and we assume that the agents are available locally and that the service requests are stateless. Stateless operation allows the system to change the usage strategy without costly state transfer between agents.

In this paper service composition and partitioning are synonymous; however, service composition could also have more requirements, such as application requirements that would need to be considered. Moreover, we have assumed that the structure of the service being composed is available; however, in multi-vendor scenarios the client or initial service provision agent may need to negotiate the service structure by, for example, participating in auctions.

This paper is structured as follows: Section 2 presents the system overview and examines various models and how they are used in service composition. Section 3 presents the experimental application scenario: the recommendation service. Finally, Section 4 presents the conclusions.

1.1 Background and Related Work

Service composition has been the subject of many research projects, such as the Ninja project [5] and SAHARA [12]. In addition, W3C is specifying the Semantic Web, which includes specifications for WSDL, SOAP and other protocols that may be used to describe, access, execute, and discover services on the Web [8]. The Semantic Web effort provides mechanisms that may be used to realize automatic service composition and they are a building block for interoperable multi-agent systems and complement the existing agent standards, such as the FIPA architecture [3].

Agents have been considered for service composition both in fixed-network environments and in mobile environments [1,2]. Mobile agents and fixed-network proxies have been considered for application partitioning [14]. In addition, service partitioning in wireless and mobile environments with FIPA agents has been examined in [9]. A theoretical examination of selecting the best communication protocol has been presented in [11] in the context of service provision. One of the conclusions of the paper is that local decisions by agents in minimizing load lead to globally good behaviour. Agent-based service composition based on negotiation in multiple auctions has been examined in [10].

The work presented in this paper is based on the FIPA architecture [3], which aims to improve agent interoperability by standardizing the agent platform and the external interfaces of agents and platforms, such as: message transport protocols, message envelopes, agent communication language (ACL), content languages and interaction protocols.

MicroFIPA-OS is an agent development toolkit and platform based on the FIPA-OS agent toolkit. The system targets medium to high-end PDA devices that have sufficient resources to execute a PersonalJava compatible virtual machine. MicroFIPA-OS allows the use of FIPA-OS components such as AMS and DF, and supports the rapid prototyping of agents in PDA environments without modifying agent code with tradeoffs between portability, and performance and resource consumption [15].

Previous performance measurements [15] with MicroFIPA-OS indicate that high-end PDAs are capable of supporting multiple agents on a single device; however, with a cost in system latency and memory. The wireless connections are the critical part of the system being considerably slower than messaging within devices. Since the messaging environment is constrained by the capabilities of the device and the limitations of the wireless link, the system needs to decide what is transmitted and when [7,15].

2 Adaptive Service Composition

Service composition and adaptation to changes in the client communication context requires that the agent or agent execution environment on the terminal has a model of the environment, and the different components that are available. The model is used as the basis for decision-making in selecting the best way of using the components. Fig. 1 presents an overview of different aspects of the environment. The changeable parameters that may be used for optimization are illustrated using italics. Observable parameters are highlighted using the bold type. In this paper, a service is composed of a number of agents, which are available locally, remotely or both to the service-requesting agent on a terminal device.

We propose a system that finds an optimal usage pattern for the agents that can be used either locally or externally. The primary goal of this system is to minimize application-level latency, which is important in order to provide reasonable response times when slow and unreliable communication links are used. The system provides a flexible way to partition a service between terminals and servers thus reducing server load when clients have enough processing power to execute parts of the service.

We assume that one agent requests service from the other agents. The service experienced by this agent is composed of one or more agents. Some of the agents may or must be local to the requesting agent, and some agents may or must be external.

There are many different possible adaptation techniques. Table 1 identifies three possible adaptation techniques in an agent execution environment. We focus on service composition. Dynamic selection of the MTP and different message encodings create a more complex scenario.

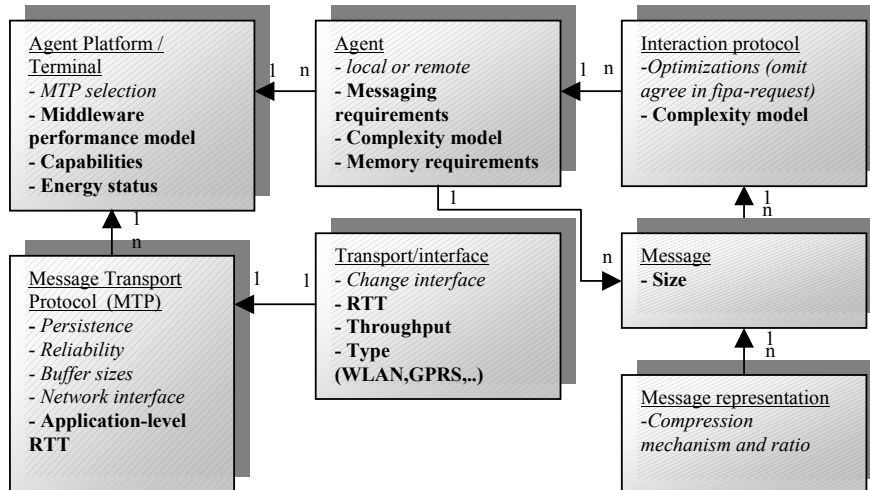


Fig. 1. Dependencies of various components of an agent system and an overview of the observable (bold) and changeable (italics) parameters

Table 1. Adaptation techniques in an agent execution environment

Adaptation Decision	Description
Service composition	Find the optimal location of agents, and their parameters. Agents are non-mobile, stateless and available locally, or remotely. The execution environment may support this transparently using information stored in application profiles.
Selection of MTP	The system can change MTP to a more suitable protocol when the link properties change. In addition, MTP may support message prioritization, and various messaging strategies (push/pull). Selection of MTP is not addressed in this paper.
Message representation	The message representation and compression can be changed to optimize the number of bits transmitted and the communication latency. Message representation issues are not addressed in this paper.

2.1 Service Composition

Agent-based services can be implemented using a number of agents that co-operate in the realization of the service functionality. An agent wishing to use the service needs to know at least one agent that provides the service. It is assumed that a service consists of N agents, and some may be local and some external. External agents are required, because many service elements cannot be provided locally because of their nature. On the other hand, it is not reasonable to use some components externally if they are locally available and the execution environment is fast enough.

The service composition decision can be made at start-time or at run-time. Start-time decisions may become unoptimal when the environment changes. Run-time composition can be further categorized based on the temporal properties of the composition decision: one-shot or continuous. One-shot decisions are evaluated separately for each service request. Service requests are typically modeled using different interaction protocols. Interaction protocols specify the message flow and possible states of an agent interaction and simplify the development of agent systems. One-shot decisions require that the decision algorithm is executed for each interaction and the algorithm needs to take into account also the other interactions. The second category assumes that the decision holds for all interactions for a certain period of time, usually the time between context changes. The latter approach requires information about context changes in order to start the re-evaluation of the service access strategy. Fig. 2 illustrates the continuous and dynamic service composition and adaptation process. When the execution environment changes, the system evaluates the current service access strategy and possibly re-configures the system using a control mechanism.

In order to make a composition decision, we need to have a model of the application behaviour: the type, volume and frequency of interactions. The modelling of application behaviour is difficult and may require input from the authors of the application and run-time performance monitoring and modelling. The execution environment may support service composition transparently using information stored in application profiles. The applications profiles need to include information about the location, availability and requirements of various components.

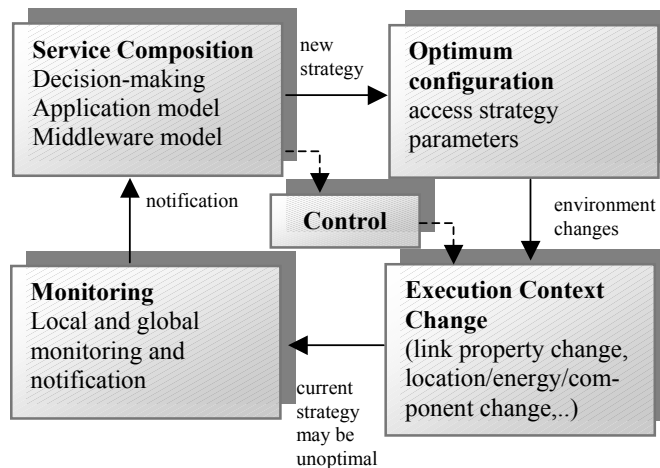


Fig. 2. Service composition in a dynamic environment

2.2 Application Model

For one-shot interactions, we need to model the expected number of bytes that are sent and received, and the expected processing time of each request. Complex functionality or interactions require more complex models for the communication. The adaptation engine uses this model in deciding whether external or local agent is used. We assume that all agents that are components of a service are available on the local system, so the download time is not taken into account, which is a factor in real-life situations. The main metric is the total time spent in messaging, which should be minimized. Additional metrics are memory and resource use on the local system. The experimental model is presented in Section 3.

2.3 Theoretical Cost Model

Equation 1 presents the one-shot external cost in seconds for accessing an external agent using the fipa-request interaction protocol [4] with the agree performative omitted. Srv denotes the processing time of the request on the server in seconds, M_{req} is the request size in bytes and M_{res} is the response size in bytes, τ is the bandwidth (bps) and δ is the link latency in seconds. We assume symmetric communication links and all variables are non-negative and finite. In addition, there may be only one service request active at a time or the effect of existing messaging needs to be taken into account in the bandwidth:

$$E(C_E) = E(Srv) + 2\delta + \frac{8(E(M_{req}) + E(M_{res}))}{\tau}. \quad (1)$$

Equation 2 presents the cost of accessing a local agent with the same interaction protocol. L_{cl} denotes the expected local processing time of the agent to be contacted, and C_{Lreq} is the expected cost in seconds for the local request operation that is middleware specific:

$$E(C_L) = E(L_{cl}) + E(C_{Lreq}). \quad (2)$$

2.4 Composition Decision

Context-aware service composition may be represented as a decision theoretic problem. Let $A = \{A_i \text{ for } i = 1, 2, \dots, M\}$ be a finite set of decision alternatives, different ways that agents of a service may be contacted, and $G = \{G_j \text{ for } j = 1, 2, \dots, N\}$ a finite set of goals that represent the agents that need to be contacted. A decision matrix is used to represent the decision problem. An element a_{ij} of $(M \times N)$ decision matrix represents the performance of alternative A_i when it is evaluated in terms of the goal G_j . In order to solve the problem and find A^* , the optimal

configuration of local and external components, we use the weighted sum model (WSM), which is a common approach for single dimensional problems under the additive utility assumption [16]:

$$A^* = \underset{i}{\text{Min}} \sum_{j=1}^N a_{ij} . \quad (3)$$

Since each agent may be used locally or remotely, each goal may take either the value of 0 (local) or 1 (remote). There are $M = 2^N$ possible configurations, and therefore M rows in the decision matrix. Let a' denote an element of a second $M \times N$ matrix that represents the different decision alternatives, the different binary N-strings. Equation 4 presents how an element a_{ij} of the decision matrix is calculated:

$$a_{ij} = a'_{ij} E(C_E(j)) + (1 - a'_{ij}) E(C_L(j)) . \quad (4)$$

In equation 4, functions $C_E(j)$ and $C_L(j)$ represent the cost of requesting service from the agent j externally and locally, respectively. In addition to response time, we may also consider the overhead of using the components locally. Let β denote the threshold, in percentage, that the cumulative local resource overhead must not exceed, and let R_{local} denote the local resource overhead increment by using the given resource locally. Equation 5 presents the resource overhead for the decision alternative i , which must be below β :

$$\varphi(i) = \sum_{j=1}^N (1 - a'_{ij}) R_{local}(j) . \quad (5)$$

A^* may be found by exhaustion by going through all the 2^N subsets and finding the sequence that solves the problem with the smallest response time, possibly taking into account the overhead as well. Equation 5 allows us to remove rows of the decision matrix that do not satisfy the inequality ($\varphi(i) < \beta$). The decision matrix and A^* are found using the following steps:

1. Create a temporary (M, N) matrix T' and each row a'_i of T' contains a binary decision alternative for N goals. If a component j can only be used remotely, its value in $a'_{ij} \in T'$ is set to 1. Duplicate rows are removed from T' .
2. Let E denote the set of rows that do not satisfy the inequality; thus $x \in E$, $\varphi(x) \geq \beta$, and it follows $M = |A \setminus E|$. Rows in E are removed from T' .
3. Create a (M, N) matrix using Equation 4.
4. Sum all rows of the decision matrix.
5. Select the row (A^*) with the minimum sum (Equation 3).

2.5 Discussion

The dynamic service composition and partitioning problem is challenging, because the environment changes over time. The proposed model is relatively simple and more complex models would reflect service usage better. We have focused on decisions based on the measurements of interactions for stateless agents one strategy at a time. The evaluation of longer and more complicated communication patterns may require support for context prediction and communication pattern duration estimation, because the system needs to know for what period of time the strategy is evaluated. Stateful agents require that a certain adopted communication pattern is maintained as long as the state is maintained by the service agent.

This approach requires that the service requests are sequential in nature and they can utilize the given bandwidth of the link. If the requests are not sequential the usage decision may become unoptimal, because the link characteristics do not reflect the current situation of the communication link, which may be congested and have additional latencies due to message buffering. Simultaneous requests require additional information in making the usage decision, such as the average request rate for each interaction. Another approach is to monitor the bandwidth and latency, and when a long-term change is perceived a context change event is fired and a new usage strategy is formulated using the new estimates of the link characteristics.

3. Example Scenario: Recommendation Service

We have created a simple adaptive service scenario based on the ideas presented in Section 2. The experimental scenario used to examine service composition in wireless environments consists of a location-based recommendation service. The service takes the location of the client, accesses an external database that contains location specific information, such as nearby restaurants and their menus, and creates a map image of the neighborhood with the nearby location highlighted. The results presented in this section are based on a prototype Java implementation of the components of the map service, and the MicroFIPA-OS platform [15]. The service configuration is illustrated Fig. 3 and the system consists of the following components:

- Client agent that orchestrates the service usage and has knowledge of its location,
- Proxy DB-agent that caches map information and retrieves information from location-information database located on the fixed network,
- Map-agent that takes the location-information from the database and builds an image of the neighbourhood,
- Proximity agent that calculates distances between objects and returns the identifiers of nearby objects, and
- Database-agent that manages location-based information.

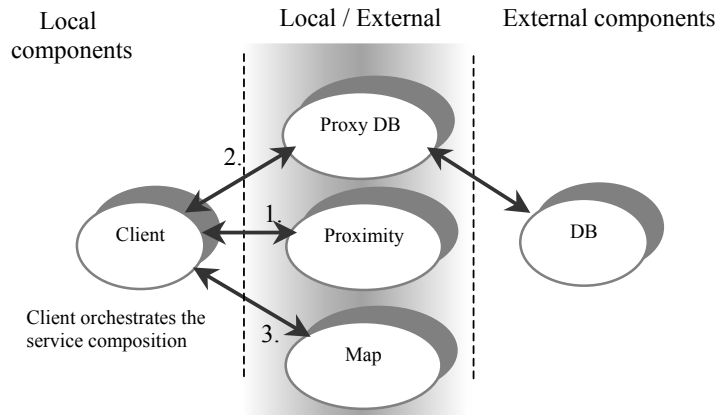


Fig. 3. Overview of the service configuration space

In Fig. 3, the client agent uses the other agents in order to create a map of nearby locations. First, the client sends the coordinates to the Proximity agent that returns the identifiers of the nearby locations (1). After this, the client sends the identifiers of the locations to the Proxy DB/Database-agent (2), which returns their names and other relevant information. Finally, the client sends the data to the Map-agent, which creates an image of the nearby locations with the coordinates highlighted on the map. Table 2 presents the equipment that was used in the experimentation. All the local processing times were measured with a laptop and a Compaq iPAQ, and in the network performance tests the peer computer was another laptop. The client system used Nokia D211 for WLAN/GPRS connectivity.

Table 2. The equipment used in the measurements

Device	Hardware	Software
Windows terminal	HP Omnibook 6100 512MB RAM 1GHz CPU	Windows 2000 Professional Service Pack 3, RC 3.136 JDK1.1.8
Linux terminal	HP Omnibook 6100 512MB RAM 1GHz CPU	Linux 2.4.18 JDK1.1.8
Handheld terminal	Compaq iPAQ H3630 32MB RAM 206 MHz CPU	Linux 2.4.3 JDK1.1.8
Network peer	Toshiba Portege 7020 128MB RAM 366 MHz CPU	Linux 2.2.18 JDK1.1.8

Table 3 presents a summary of the output data of the components and their average processing times. These measurements were made using 1000 replications. The input arguments to the Proximity-agent were longitude, latitude, and the diameter, and the output was a list of identifiers. The identifiers were the input to the Database-agent, which returned a list of strings. The Map-agent input was a list of strings and coordinates, and the output was a byte array consisting of the compressed image (GIF-format). In the experiments we omitted the overhead caused by the agent platform, because the components are generic service objects. The network performance measurements were performed in two environments: WLAN and GPRS. The WLAN was a private network consisting of a WLAN access point connecting the terminal with the network peer. In the case of GPRS, on the other hand, a live and public production network was used. Table 4 presents the results of wireless access of the same components using WLAN and GPRS on top of the MicroFIPA-OS execution environment. The wireless measurements were made using 20 replications.

This experimental data is the basis for the application profile that is used in deciding what components are used and where. The application profile consists of the FIPA-request protocol information: the types of FIPA-request messages sent and received, their estimated frequency, processing times locally and remotely, and an estimated size for the request and reply messages of each type. This information is used to evaluate the cost of local and external communication.

Table 3. Summary of output data and average processing time of the three components on three different devices

Case	Setup	Output vector size	Output size (bytes)	Avg. Processing time (ms)
Proximity	Laptop (Win)	1	22	0.0014
		50	1103	0.06439
		100	2234	0.13019
	Laptop (Linux)	1	22	0.0032
		50	1103	0.15486
		100	2234	0.31019
	iPAQ (Linux)	1	22	1.408
		50	1103	73.263
		100	2234	144.042
Database	Laptop (Win)	1	16	0.0018
		50	802	0.0698
		100	1533	0.1381
	Laptop (Linux)	1	16	0.02405
		50	802	0.50064
		100	1533	0.98662
	iPAQ (Linux)	1	16	0.332
		50	802	5.702
		100	1533	11.19
Map	Laptop (Win)	50x50	3639	9.474
		200x200	46159	46.096
	Laptop (Linux)	50x50	3639	16.575
		200x200	46159	198.979
	iPAQ (Linux)	50x50	3639	35151
		200x200	46159	121981

Table 4. Remote performances of the three components using WLAN and GPRS on MicroFIPA-OS with FIPA-request (agree message omitted)

Case	Network	Output	Avg. time (ms)	Standard Deviation
Proximity	GPRS	1	3713.9	110.27
		50	4387.58	63.51
		100	4615.05	201.11
	WLAN	1	37.8	7.25
		50	55	5.55
		100	72.4	6.88
Database	GPRS	1	3715.58	121.87
		50	3815.21	436.95
		100	4418.26	696.69
	WLAN	1	38.55	6.45
		50	47.8	4.94
		100	59.25	5.38
Map	GPRS	50x50	5454.95	406.21
		200x200	19705.79	409.57
	WLAN	50x50	122.75	5.24
		200x200	1223.8	15.86

Table 5 presents the decision matrix with the 2^3 decision alternatives and the approximate response time. The number 0 denotes that the component is used locally, and 1 that it is used externally. The database column denotes the database proxy agent when the local component is used, and the actual database agent when the external component is used. For these results we have used the laptop-based network measurements, because the wireless link is slower than local processing. We have previously measured the approximate local fipa-request latency of the MicroFIPA-OS system on the iPAQ as 144 ± 22 ms using the interoperable FIPA-OS API, and 37.3 ± 0.48 ms using the minimal messaging API that lacks conversation management [15]. Table 5 includes the approximate response time calculated using the latter latency.

In runtime operation, the system may use Equation 1 to calculate an estimate for the remote interaction if the application profile contains information about processing times and message sizes. We plan to take the memory overhead of the components into account in the future.

The results of Table 5 indicate that we may gain considerable benefits by finding the best access pattern for the components of a service. The smallest response times are highlighted using underlining and bold type. The selected strategy for WLAN offers 18.7% (2% if local latency is taken into account) improvement in response time when compared with the strategy in which all components are external, and it is 192 (160) times faster than all-local operation. Similarly, the selected strategy for GPRS offers 60% (59%) improvement from the all-external scenario, and it is 6.4 (6.3) times faster than all-local operation. The service latency is lowest for the iPAQ when the client does not orchestrate the service, but only fetches the image from the fixed-network side; however, this kind of approach does not provide any functionality when the client is disconnected.

Table 5. Decision matrix for the experimental scenario on iPAQ with WLAN and GPRS (50x50 image with output vector size 50)

	Proximity	Database	Map	Response time (ms)	Response time with local latency (ms)
1	WLAN	0 (73.263)	0 (35151)	35229.97	35341.87
	GPRS	0 (73.263)	0 (35151)	35229.97	35341.87
2	WLAN	0 (73.263)	1 (122.75)	201.715	276.315
	GPRS	0 (73.263)	1 (5454.95)	5533.915	5608.5
3	WLAN	0 (73.263)	1 (47.8)	35272.06	35346.66
	GPRS	0 (73.263)	1 (3815.21)	39039.47	39114.07
4	WLAN	1 (55)	0 (35151)	35211.7	35286.3
	GPRS	1 (4387.58)	0 (35151)	39544.28	39618.88
5	WLAN	0 (73.263)	1 (122.75)	243.813	281.113
	GPRS	0 (73.263)	1 (5454.95)	9343.423	9380.723
6	WLAN	1 (55)	1 (47.8)	35253.8	35291.1
	GPRS	1 (4387.58)	1 (3815.21)	43353.79	43391.09
7	WLAN	1 (55)	1 (122.75)	183.452	220.752
	GPRS	1 (4387.58)	1 (5454.95)	9848.232	9885.532
8	WLAN	1 (55)	1 (122.75)	225.55	225.55
	GPRS	1 (4387.58)	1 (5454.95)	13657.74	13657.74

In this scenario, the bottleneck component is the map component, which is slow in local operation. The results are similar with 200x200 images and with output vector size of 100 on the iPAQ. On a laptop, however, the all-local configuration has the lowest response time and the all-external the highest response time, because wireless communication is slow. These results show that neither the all-external nor the all-local configurations are optimal for a small device given that the service is orchestrated from the client-side. In addition, the results motivate using local components and service logic with slow links and reasonably powerful client systems, such as GPRS and the laptop used in this study.

4. Conclusions

One of the key requirements for applications and services is to minimize the delay experienced by the user. In this paper, we have investigated how the latency of services that consist of multiple agents can be improved by adaptive service composition. In the proposed model, the agent responsible for service composition decides whether to contact an agent externally or locally. This decision is made when the execution context changes. We have assumed that the agents are locally available, and that they are stateless. We presented a method for formulating a decision matrix and making the usage decision, and an example service scenario. The experimental results show that the client agent may gain substantial improvement in latency by selecting the best combination of local and external resources.

This kind of approach may also be used to implement partial fault tolerance, in which part of the service functionality can still be provided even if network

connectivity is lost. In the future we plan to examine service composition and reconfiguration in mobile and wireless environments, context events for triggering reconfiguration, and sensitivity analysis of the composition and partitioning decision.

Acknowledgements

This work has been funded by the National Technology Agency of Finland (Tekes), Elisa Communications, Ericsson, Nokia, TeliaSonera, More Magic Software, and Movial. We thank Heikki Helin and Jaakko Kangasharju for valuable comments.

References

1. Chakraborty, D., Joshi, A. Dynamic Service Composition: State-of-the-Art and Research Directions. Technical Report TR-CS-01-19. CSEE. University of Maryland, Baltimore County. December 2001
2. Chen, H., Joshi, A., Finin, T. Dynamic Service Discovery for Mobile Computing: Intelligent Agents Meet Jini in the Aether. Cluster Computing, Volume 4, Number 4, pages 343-354, 2001
3. Foundation for Intelligent Physical Agents (FIPA). The FIPA Architecture, Geneva, Switzerland, 2003. Available at <http://www.fipa.org>
4. Foundation for Intelligent Physical Agents (FIPA). FIPA Request Interaction Protocol Specification. Available at <http://www.fipa.org/specs/fipa00026/>
5. Gribble, S. D., Welsh, M., von Behren, R., Brewer, E. A., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, A. D., Katz, R. H., Mao, Z., Ross, S., and Zhao, B. The Ninja Architecture for Robust Internet-Scale Systems and Services, Special Issue of Computer Networks on Pervasive Computing, March 2001, 473-497
6. Jennings, N., Sycara, K., Wooldridge, M. A Roadmap of Agent Research and Development, Autonomous Agents and Multi-Agent Systems (275-306), Kluwer Academic Publishers, Boston, 1998
7. Laukkanen, M., Helin, H., Laamanen, H. Supporting Nomadic Agent-based Applications in the FIPA Agent Architecture. In Cristiano Castelfranci and W. Lewis Johnson, editors, Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2002) pages 1348-1355, July, 2002, Bologna, Italy
8. McIlraith, S., Son T., Zeng, H. Semantic Web Services. IEEE Intelligent Systems, Special Issue on the Semantic Web, Volume 16, No. 2, pp. 46-53, March/April, 2001
9. Koskimies, O., Raatikainen, K. Partitioning Applications with Agents. In the Second International Workshop on Mobile Agents for Telecommunication Applications (MATA2000), pages 79-93. Lecture Notes in Computer Science, Springer Verlag, September 2000
10. Preist, C., Byde, A., Bartolini, C., Piccinelli, G. Towards Agent-Based Service Composition through Negotiation in Multiple Auctions. HP report, HPL-2001-71
11. Preist, C., Pearson, S. An Adaptive Choice of Messaging Protocol in Multi Agent Systems. Foundations and Applications of Multi-Agent Systems, LNAI 2403, Springer, 2002

12. Raman, B., Agarwal, S., Chen, Y., Caesar, M., Cui, W., Johansson, P., Lai, K., Lavian, T., Machiraju, S., Mao, Z.M., Porter, G., Roscoe, T., Seshadri, M., Shih, J., Sklower, K., Subramanian, L., Suzuki, T., Zhuang, S., Joseph, A.D., Katz, R.H., and Stoica, I. The SAHARA Model for Service Composition Across Multiple Providers. *Pervasive Computing*, August 2002. Lecture Notes in Computer Science LNCS 2414, Springer, 2002
13. Satyanarayanan, M. Pervasive computing: vision and challenges. *IEEE Personal Communications*, August 2001
14. Schill, A., Held, A., Ziegert, T., Springer, T. A Partitioning Model for Applications in Mobile Environments. In Todd Papaioannou and Nelson Minar, editors, *Proceedings. Mobile Agents in the Context of Competition and Cooperation (MAC3)*, a workshop at *Autonomous Agents '99*, pages 34-41, 1999
15. Tarkoma, S., Laukkanen, M. Facilitating Agent Messaging on PDAs. *Fourth International Workshop on Mobile Agents for Telecommunication Applications (MATA-2002)*, Barcelona, Spain. Lecture Notes in Computer Science (LNCS) 2521, Springer 2002
16. Triantaphyllou, E., Shu, B., Nieto Sanchez, S., and Ray, T. Multi-Criteria Decision Making: An Operations Research Approach. *Encyclopedia of Electrical and Electronics Engineering*, (J.G. Webster, Ed.), John Wiley & Sons, New York, NY, Vol. 15, pp. 175-186, 1998