# Controlling and Sharing Distributed Data for Implementing Service Alliance

Yasuhito Asano*, Zhenjiang Hu†, Yasunori Ishihara‡, Hiroyuki Kato†, Makoto Onizuka§ and Masatoshi Yoshikawa*
*Kyoto University, Kyoto, Japan †National Institute of Informatics, Tokyo, Japan
‡Nanzan University, Nagoya, Japan §Osaka University, Suita, Japan

***Abstract*—According to the increase of venture companies providing similar services, alliances of them have become popular in recent years. Alliances might open a new business chance as explained above, although implementing an alliance has several problems in the aspect of distributed data management. One of the possible solutions is Dejima architecture proposed by Ishihara et al. based on the bidirectional transformation techniques. In this paper, focusing on a ridesharing alliance, we first clarify properties desired for controlling and sharing distributed data to implement a service alliance. We then propose an idea for implementing a ridesharing alliance utilizing the Dejima architecture, and discuss which properties can be satisfied by the implementation.**

***Index Terms*—data integration, distributed data management, bidirectional transformation, ridesharing, service alliance**

## I. Introduction

Alliances have been observed in any era. For example, a marketplace platform, such as e-bay and Amazon marketplace, can be considered as an alliance of companies providing shopping services. Companies that participate in the alliance can display their products on the marketplace. A customer can choose a preferable product among those provided by the participant companies. A crowdsourcing platform can also be regarded as a kind of an alliance, while its flow of money is opposite to that in a marketplace platform. According to the increasing of startup companies providing similar services, alliances of them have also become popular in recent years. For example, Licorish [1] reported there are many ridesharing companies in Canada, and Tom [2] reported partnerships between ridesharing companies. Let us call such an alliance of similar service providers a *service alliance*, which implements a unified place to match customers to their services. In an service alliance, service data published by providers and customer data are shared in the place so that customers choose services or providers choose customers among them.

The advantage of an alliance is to increase the chance of matching providers to consumers. From the viewpoint of customers, too many similar services to choose cause a waste of time for them. If an alliance enables customers to search preferable ones for them, it would be a great help. On the other hand, because an alliance would obtain more customers than a single company within it, it would enables providers to increase the chance to be known by customers.

Alliances might open a new business chance as explained above, although implementing an alliance has several problems in the aspect of data management. One of the easiest ways of implementing an alliance is an approach that a single representative company manages global data of all the participant provider companies and each provider manages local data by itself. An example is Amazon marketplace; Amazon has a database managing all the products, and each provider sends Amazon the information about a product which it selects to sell; if the product is sold on Amazon marketplace, then the customer information is sent to the provider and it updates its local database.

Methods for controlling and sharing data in this approach tend to be fairly primitive; actually, each provider in the example above has to write a program to manage its local database to select products and update it according to the information sent from Amazon. If a provider takes part in multiple alliances, then such a program could be more complicated. On the other hand, it is difficult for an alliance to prepare a program for sharing data between its global database and the local database of every participant provider, because the local schema of each provider would be completely different from each other. Therefore, more sophisticated data sharing methods are desired.

The another problem of the approach above is that the representative, such as e-bay and Amazon, managing the global database has much power than the other participants. Several providers, e.g. startup companies competing with each other, do not prefer such a situation.

Establishing an alliance between companies might cause an information leak that never occurs in a company alone. For example, a careless program in an alliance might allow a secret of a company to propagate to another company, or allow customer information to propagate to companies which should not obtain it. This can be a serious problem for alliances of companies providing sharing economy services; because people who are not professional are engaged in service (e.g. drivers in ridesharing services), such people and customers tend to be sensitive to their privacy with each other. Therefore, an architecture is desired which can prevent unintentional information leak due to an alliance.

One of the possible solutions is Dejima architecture [3],

This work is partially supported by the Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (S) No. 17H06099, (A) No. 18H04093, and (C) No. 18K11314.

[4] based on the bidirectional transformation (BX, for short) techniques. The Dejima architecture enables peers to write a BX for sharing and controlling data according to the own publishing and update policy of each peer. Thus, it presents a more flexible way for establishing a group of peers than previously proposed data sharing architectures including Piazza [5], [6] and ORCHESTRA (CDSS) [7], [8]. It also supports the global consistency among peers, whereas the previous ones handle the local consistency only inside of each peer.

In this paper, focusing on a ridesharing alliance, we first clarify properties desired for controlling and sharing distributed data to implement a service alliance. We then propose a fundamental idea for implementing the core part of a ridesharing alliance on top of the Dejima architecture. We finally discuss which properties can be satisfied by the implementation, and which ones remain unsatisfied.

## II. Desired Properties for Ridesharing Alliance

In this section, we first explain the stakeholders and their roles in a ridesharing alliance system. The stakeholders are passengers, the alliance, provider companies, and vehicles. Note that we regard a vehicle and its driver as the same stakeholder. We then clarify desired properties for a ridesharing alliance system, especially in the reservation process.

We describe the outline of the roles of stakeholders below.

**Passengers.** Each passenger first sends a request to the alliance to select and reserve a vehicle. Each request includes information required for a reservation, such as source/destination locations, the constraints of departure and arrival time, the number of people. Then, the passenger rides on and off the vehicle, or cancels the reservation. Finally, the passenger might evaluate the vehicle.

**Alliances.** The Alliance integrates the vehicle data received from the companies. It also responds to the request of a passenger, and tells the passenger's choice for reserving a vehicle to the corresponding company. Each response includes information required for selecting a vehicle, such as estimated arrival time and fee. It also sends the passenger the decision of the company about the reservation (see below).

**Companies.** Each company manages the data of vehicles which belong to it, and sends the data of vehicles satisfying the conditions it sets. For example, a company might disclose the data of vehicles in a specific area only to the alliance. Each company has a distinct business logic for determining fee and wage. It also receives the passenger's choice from the alliance, decides whether it accepts or denies the reservation according to its business logic, and sends the decision to the alliance.

**Vehicles.** Each vehicle sends its data, e.g. its location and the number of empty seats, to the company to which it belongs. It also picks up and drops off passengers. After a ride, it might evaluate the passenger.

The most complicated process is the reservation of a vehicle by a passenger. If this process can be implemented, then the remaining processes could be implemented also. Therefore, we below focus on this process. Figure 1 illustrates the flow of this process. Note that multiple alliances may exist, although we draw a single alliance here for simplicity,

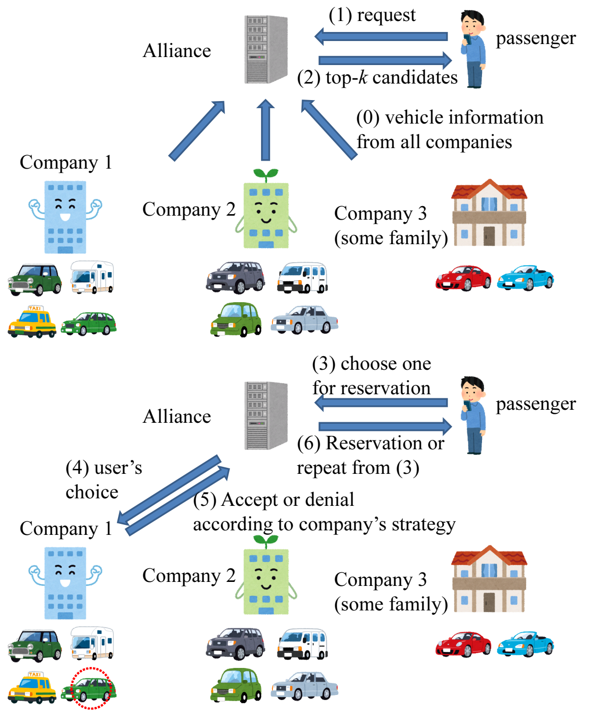

Fig. 1. The flow of a reservation in a ridesharing alliance. Note that the alliance is not required to be a trusted third party or a representative company.

We below enumerate desired properties for the reservation process.

(a) Sharing and controlling data between alliances and companies can be done automatically by writing a program according to simple and unified rules.
(b) The operation sequences for a reservation process must be consistent. For example, Any double-booking (i.e. reservations whose total number of passengers exceeds the number of seats in the same vehicle) should be avoided.
(c) It should be guaranteed that the stakeholders follow the specified protocol.
(d) A passenger can avoid a leak of his/her location as possible.
(e) A company can participate in multiple alliances.
(f) A company can have a local database having a different schema from that of the global database of the alliance, as long as its schema contains sufficient information for the reservation process.
(g) Similarly, a company can have a local database having a different schema from that of another company.
(h) A company can describe its policy for determining which vehicle's information should be published to the alliance.
(i) A company can describe its strategy for accepting and denying a request assigned by the alliance.
(j) A company can avoid a leak of its vehicle information as possible.

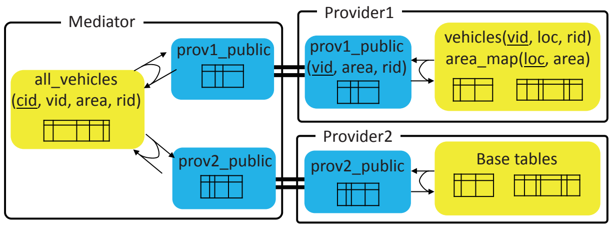

Fig. 2. An implementation of a ridesharing alliance on the Dejima architecture. Yellow rounded rectangles represent sets of tables, while blue ones are views. The combination of two arrows and a curve represents a BX between the base tables and this Dejima (*get* from a base table to its corresponding view and *put* of the opposite direction).

(k) A vehicle can belong to multiple companies.
(l) A vehicle can pick up a passenger outside the system.

## III. Implementation of Alliance on Dejima

The Dejima architecture [3], [4] presents a flexible way to a group of peers for sharing a part of their data through views and controlling updates of the shared data, by writing a BX between each peer's view and local database. In this section, we propose an implementation of the reservation process of a ridesharing alliance using a trusted third party, abbreviated to TTP, and the Dejima architecture. Although we might be able to implement an alliance without a TTP, we leave such an implementation as a future work.

Figure 2 illustrates an implementation of the core part of a ridesharing alliance, i.e. the relationship between the alliance and companies, on the Dejima architecture. In this implementation, the alliance is represented by a mediator peer corresponding to a TTP, and each company is represented by a provider peer, because the alliance mediates the vehicle data provided by companies.

Each provider peer has base tables for managing local data:

- `vehicles(vid, loc, rid)`, which contains for each vehicle its identifier, current location, and the request identifier assigned to the vehicle, and
- `area_map(loc, area)`, which is used to obfuscate the precise locations of the vehicles by associating each location with a less precise area.

For simplicity, we regard a vehicle as available if `rid` is `NULL`. We omit vehicle peers in this figure. Instead, we assume that if its location is changed or its request is done, the corresponding data is updated on the base tables.

In this figure, Provider 1 publishes the information of available vehicles to the mediator through a view (called Dejima), `prov1_public(vid, area, rid)`. Note that, for privacy protection, only the area information (instead of the precise location information) of a vehicle is disclosed.

We assume here that Provider 2 has the same schema as Provider 1, although the Dejima architecture allows Provider 2 to have a different schema and business logic.

These Dejimas are synchronized with Dejimas in the mediator in order to have the same content, as represented by double lines in Figure 2. The mediator can integrate the information of all vehicles published by the providers into a single table `all_vehicles(cid, vid, area, rid)`, using a BX adding company identifiers (`cid`) to the simple union of the Dejimas in the mediator.

The explanation above corresponds to (0) in Figure 1, the integration of vehicle information. We below explain how the reservation process works in the implementation. The indices in the following list corresponds to those in Figure 1.

**Reservation process.**

(1) A passenger sends a request to the mediator. The mediator assigns an identifier `rid` to the request.
(2) The mediator calculates the estimated arrival time and fee by querying `all_vehicles`, and returns the passenger top-$k$ candidates according to some ranking function which can be specified by the passenger or mediator.
(3) The passenger chooses a candidate to reserve it.
(4) The mediator updates `all_vehicles` to assign the `rid` to the chosen candidate. The update is reflected to the corresponding Dejima in the mediator by the BX. For example, if the candidate is a vehicle of Provider 1, then the Dejima `prov1_public` is updated. Then, the update of the Dejima in the mediator is propagated to the Dejima of Provider 1.
(5) The provider decides whether it accepts or denies the update on the Dejima (i.e. the assigned request by the mediator) according to its update strategy. For example, a provider might not accept a request whose profit is too small. Several kinds of update strategy can be written in the BX between the base tables and the Dejima in a provider. An accepted update is reflected on the base tables, while a denied update is not reflected.
(6) The decision is propagated to the mediator through the Dejimas. That is, while an accepted update becomes valid on `all_vehicles`, a denied update is canceled on `all_vehicles`. The mediator then tells the result to the passenger. If the update is valid, then the reservation is established. The passenger sends information including the source and destination location to only the assigned vehicle. Otherwise, the passenger should choose another candidate. Therefore, the steps (3) to (6) are repeated until a reservation is established.

## IV. Properties Satisfied by Dejima

In this section, we discuss which properties are satisfied by the implementation on Dejima among those enumerated in Section II. Note that because the Property (c) could not be satisfied, it will be discussed in the next section.

The Property (a) is achieved by writing a BX between each pair of a Dejima and the set of corresponding base tables. Although we omit the details, the BX can be written according to simple rules which determine how an update in Dejima or base tables propagates to each other [3].

The Property (b) is achieved by the distributed transaction mechanism of the Dejima architecture [4].

It is difficult to satisfy the Properties (d) and (j) completely because the state-of-the-art privacy protection methods and access control methods are not perfect. However, addition of the alliance itself does not cause more information leak than a single company does. For (d), the location data of a passenger is known to the mediator, which is not regarded as an information leak if the mediator is a TTP. The reserved vehicle also knows the location, although this happens in the case of a ridesharing using a single company. We also note that this is not considered as an information leak in several studies about privacy-preserving ridesharing [9], [10]. For (j), Only the area information of vehicles is disclosed to the mediator, and the vehicle information of a company is not propagated to the other companies.

Furthermore, a passenger and a vehicle can obfuscate their locations by applying other location obfuscation techniques, e.g. the geo-indistinguishablity [11], to the implementation. This approach prevents the mediator from knowing the precise location of them.

The Property (e) should be satisfied without violating the Property (b). Companies participating multiple alliances may form a cycle that causes infinite cyclic propagation. For example, we assume that company $c_1$ and $c_2$ participate in alliances $A_1$ and $A_2$. Then, these peers form a cycle $c_1 \rightarrow A_1 \rightarrow c_2 \rightarrow A_2 \rightarrow c_1$. Even in such a case, the data independence check and cyclic propagation detection mechanisms of the Dejima architecture [3], [4] allows us to achieve these properties.

The Properties (f) and (g) are achieved by the BX, similarly to the Property (a). Each provider can define the relationship between its local schema and the global schema of the alliance.

The Properties (h) and (i) are also achieved by the BX, similarly to the Property (a), because each provider can specify its policy for publishing information in its base tables and its strategy for updating base tables when the corresponding Dejima is updated.

Although we omitted vehicle stakeholders in the implementation above, the Properties (k) and (l) can be achieved by adding peers representing vehicles to the implementation. Then, a pair of Dejima is created between a company and a vehicle. Writing the BX for the Dejimas properly, the Property (k) can be achieved in the same way as the Property (e), and the Property (l) can be achieved in a similar way to the Properties (h) and (i).

## V. Discussion about Unsatisfied Properties

In this section, we discuss the properties that cannot be satisfied by utilizing the Dejima architecture. Unfortunately, for the Property (c), the current Dejima architecture does not have a mechanism to guarantee that no peer can "cheat" the protocol specified for implementing the alliance. For example, a malicious company might tell its customers' information to others; the mediator might not be a trusted one. Other kinds of mechanisms including smart contract [12], [13] might be employed for achieving the property.

The implementation in Section III provides a limited protection against possible information leaks for the Properties (d) and (j). On the other hand, there are a few studies [9], [10], [14] about privacy-preserving ridesharing. Aïvodji et al. [14] proposed a privacy preservation method for computing meeting points of a passenger and a vehicle, utilizing a homomorphic encryption. This method enables a passenger and his/her "reserved" vehicle to hide their locations from each other. Tong et al. [10] proposed a method for keeping jointly differential privacy in ridesharing. It is not clear whether the current Dejima architecture is able to employ techniques proposed in these studies.

## VI. Conclusion

We have clarified properties desired for controlling and sharing data in a service alliance. We have also discussed how the Dejima architecture is useful for achieving several important properties. While we have focused on a ridesharing alliance in this paper, we believe that our discussion would be useful for implementing other kinds of service alliances because several essences are common in them.

## References

[1] D. Licorish, "Ride-sharing in Canada: It's Complicated," https://www.lowestrates.ca/blog/ride-sharing-canada-its-complicated, 2016.

[2] M. Tom, "A visual guide to the twisted web created by the Uber/Didi merger," https://pitchbook.com/news/articles/a-visual-guide-to-the-twisted-web-created-by-the-uberdidi-merger, 2016.

[3] Y. Ishihara, H. Kato, K. Nakano, M. Onizuka, and Y. Sasaki, "Toward bx-based architecture for controlling and sharing distributed data," in *Second Workshop on Software Foundations for Data Interoperability (to appear)*, 2019.

[4] Y. Asano, D.-F. Herr, Y. Ishihara, H. Kato, K. Nakano, M. Onizuka, and Y. Sasaki, "Flexible framework for data integration and update propagation: system aspect," in *Second Workshop on Software Foundations for Data Interoperability (to appear)*, 2019.

[5] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov, "Piazza: Data management infrastructure for Semantic Web applications," in *WWW*, 2003, pp. 556–567.

[6] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov, "The Piazza peer data management system," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 787–798, 2004.

[7] Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir, "ORCHESTRA: Rapid, collaborative sharing of dynamic data," in *CIDR*, 2005, pp. 107–118.

[8] G. Karvounarakis, T. J. Green, Z. G. Ives, and V. l Tannen, "Collaborative data sharing via update exchange and provenance," *ACM Transactions on Database Systems*, vol. 38, no. 3, pp. 19:1–19:42, 2013.

[9] P. Goel, L. Kulik, and K. Ramamohanarao, "Privacy-aware dynamic ride sharing," *ACM Trans. Spatial Algorithms Syst.*, vol. 2, no. 1, pp. 4:1–4:41, Mar. 2016.

[10] W. Tong, J. Hua, and S. Zhong, "A jointly differentially private scheduling protocol for ridesharing services," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2444–2456, Oct 2017.

[11] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: differential privacy for location-based systems," in *2013 ACM SIGSAC Conference on Computer and Communications Security*, 2013, pp. 901–914.

[12] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997. [Online]. Available: http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/548

[13] J. Ray, "A Next-Generation Smart Contract and Decentralized Application Platform," https://github.com/ethereum/wiki/wiki/White-Paper, 2019.

[14] U. M. Aïvodji, S. Gambs, M.-J. Huguet, and M.-O. Killijian, "Meeting points in ridesharing: A privacy-preserving approach," *Transportation Research Part C: Emerging Technologies*, vol. 72, pp. 239–253, Nov. 2016.