Frontiers in The Internet of Things

TYPE Original Research
PUBLISHED 15 December 2022
DOI 10.3389/friot.2022.1054963

EDITED BY
Chang-ai Sun,
University of Science and Technology Beijing, China

REVIEWED BY
Bo Cheng,
Beijing University of Posts and Telecommunications (BUPT), China
Xiaolong Xu,
Nanjing University of Information Science and Technology, China

*CORRESPONDENCE
Pascal Hirmer,
pascal.hirmer@ipvs.uni-stuttgart.de

SPECIALTY SECTION
This article was submitted to IoT Architectures, a section of the journal Frontiers in the Internet of Things

RECEIVED 27 September 2022
ACCEPTED 05 December 2022
PUBLISHED 15 December 2022

CITATION
Hirmer P, Breitenbücher U, Del Gaudio D, Képes K, Leymann F, Mitschang B, Mormul M and Przytarski D (2022), Situation-aware adaptation of choreographies—The DiStOPT approach. *Front. Internet. Things* 1:1054963. doi: 10.3389/friot.2022.1054963

Situation-aware adaptation of choreographies—The DiStOPT approach

Pascal Hirmer[1]*, Uwe Breitenbücher[2], Daniel Del Gaudio[1], Kálmán Képes[2], Frank Leymann[2], Bernhard Mitschang[1], Mathias Mormul[1] and Dennis Przytarski

[1]Institute of Parallel and Distributed Systems, University of Stuttgart, Stuttgart, Germany, [2]Institute of Architecture of Applications Systems, University of Stuttgart, Stuttgart, Germany

The rise of the IoT and Industry 4.0 has increased the complexity of collaborating business processes, i. e., choreographies, as more partners and assets are involved. However, maintaining and executing business choreographies are complex tasks. Moreover, enabling robust and reliable execution is important, as failures or delays cause high costs among partners. For example, manufacturing companies usually depend on different suppliers, and it is crucial to be up-to-date about possible delays in shipments as this leads to delays in the manufacturing of their products. In this case, a choreography needs to be designed and operated in a way that it can adapt to cope with such problems. This requires i) timely recognition and tamper-resistent logging of problems that occur at each involved partner, which are referred to as situations in the scope of this article, and ii) an approach for a timely adaptation of choreographies based on occurring situations. Therefore, in this article, we introduce *DiStOPT*, an approach to i) model and recognize situations in a distributed and timely manner, and ii) model and execute situation-aware choreographies based on the recognized situations. The contributions are evaluated in a manufacturing scenario and validated by a prototypical implementation.

KEYWORDS

situation recognition, choreographies, blockchain, situation templates, adaptation

1 Introduction

These days, business processes must often be integrated with other processes to coordinate complex interactions between each other, which results in choreographies. Currently, these choreographies need to cope with the emergence of new paradigms, such as the Internet of Things Machorro-Cano et al. (2020) and Industry 4.0 Sanchez et al. (2020), and therefore, require choreographies to be able to react to changes that occur in real environments, sometimes anticipated, and sometimes unanticipated and suddenly. For example, when an IoT device or a production machine in Industry 4.0 fails to execute a task (i.e., a failure situation), all dependent processes should be able to adapt themselves to this situation by using alternative activities and resources. In case of a failed embedded

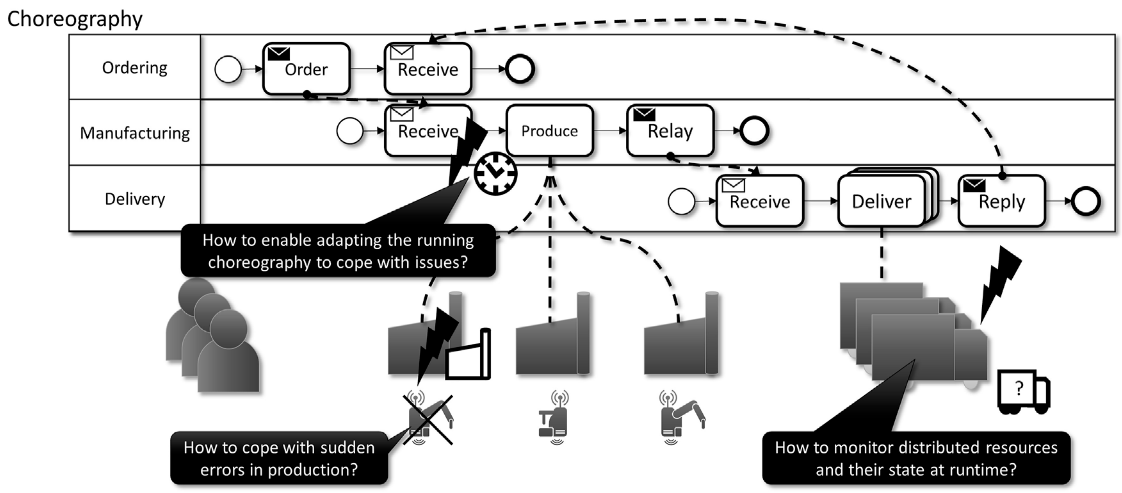
Choreography
Ordering
Manufacturing
Delivery
Order
Receive
Receive
Produce
Relay
Receive
Deliver
Reply
How to enable adapting the running choreography to cope with issues?
How to cope with sudden errors in production?
How to monitor distributed resources and their state at runtime?
?

FIGURE 1
Our motivating scenario describing an Industry 4.0 scenario (ordering, manufacturing, and delivering of products).

device, an IoT monitoring process must be able to switch over to another spare device. Failures in a production choreography may lead to failures not feasible for partner processes, e.g., waiting for a product to be shipped, therefore, adapted behavior should be enabled in a timely manner. Thus, as a result, the timely i) situation recognition and the ii) adaptation of choreographies is important in contextual and distributed systems.

In this paper, we present the *DiStOPT* approach, which extends existing work by distributed situation recognition, storage and choreography concepts to enable the development of distributed and situation-aware choreography-based applications. We evaluate our approach based on a scenario, described in Section 2, which serves as a running example throughout the paper. Afterwards, we describe fundamentals and previous work in Section 3. Our concepts are described in Section 4, Section 5 and Section 6, followed by a description of the system architecture in Section 7. In Section 8, we describe related work. In Section 9, we validate our approach based on a prototype. Finally, we give a summary in Section 10.

2 Motivation and problem statement

In the following, we describe an Industry 4.0 scenario (see Figure 1) consisting of ordering, manufacturing, and delivering of products which must interact with each other. In this scenario, monitoring data is used to derive the state of machines and possible machine failures that may influence the whole ordering process as it may introduce delays. In such dynamic scenarios, all involved machinery and infrastructure must be monitored and the current situations must be shared with other partners and processes so that they can react to changes.

In more detail, the scenario depicts three parties: ordering, manufacturing, and delivery. At the beginning, the ordering task specifies which products the manufacturer should produce within their facilities. After manufacturing, the products are sent *via* a delivery service to the location demanded by the order. Note that in this scenario, we have three simplified processes that together form a choreography, i.e., processes without a single centralized process controlling the others. In such a scenario, failures at one of the partners may lead to problems within the overall choreography. For example, if the manufacturer has failures with one of the machines that produce the ordered products (see lower left in Figure 1), it can lead to delays. However, detecting these failures early enough allows the manufacturer to adapt the running process and still deliver without significant delays. The same issue can arise with the delivery, as sudden failures in one delivery vehicle may interrupt the proper execution of the business processes. Again, timely detection of such issues can enable the adaptation of the running process by exchanging the delivery vehicle (see lower right of Figure 1). Overall, the whole choreography between the partners must be able to adapt itself to current situations. This is not an easy task, as automating in this case requires to monitor the current prevailing situations and context of a heterogeneous set of resources, e.g., people, production machines, vehicles and software processes. Modeling processes and choreographies that are situation-aware and context-aware is a complex task, especially when every detail of the monitored resources are regarded, e.g., current location, target location, and current condition. Reducing the complexity of modeling situations and context is necessary to enable situation-awareness in complex scenarios with fewer errors in the models, and therefore, also at runtime.

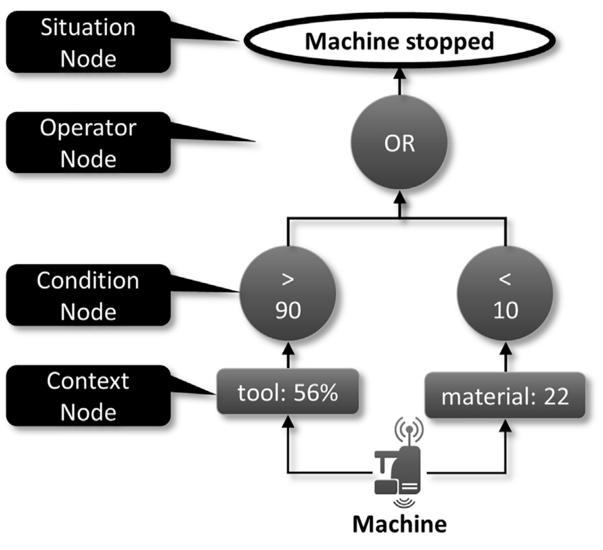

FIGURE 2
Example of a situation template Hirmer et al. (2015)

In summary, to enable the described scenario in which partners create a choreography of processes that is able to adapt itself to the current situation and context of resources, we need a system that is able to collect context data from distributed and heterogeneous resources, integrate their possible states within the processes and enable them to handle issues at runtime in case the current situation requires it. We tackle this with the *DiStOPT* approach by using i) *Situation Templates* and ii) *Situational Scopes* in combination with a distributed data collection and aggregation system, the *Situation Recognition*, to detect the current state of the environment, and by modeling the application logic in *Situation-Aware Choreographies*. By doing so, we are able to improve choreographies by reacting timely on occurring issues.

In the following, we will describe the fundamentals of *Situation Templates*, *Situational Scopes*, as well as the work we build on called *SitOPT* Wieland et al. (2015).

3 Foundations and previous work

This section introduces previous work regarding *Situation Templates* and *Choreographies* as well as other foundations required for understanding the concepts presented in this paper.

3.1 Foundations of situation recognition

Zweigle et al. (2009) and Häussermann et al. (2010) introduced *Situation Templates* as a way to model situations based on context data, which, for instance, may originate from sensors producing signal outputs. *Situation Templates* are based on so-called *Situation Aggregation Trees* and follow a tree structure with a single root node, which represents the situation. The leaf nodes are so-called *Context Nodes* describing the context data involved to recognize a situation. For example, an increasing temperature measured by a sensor of a production machine could indicate that there is an issue. These context nodes are connected to *Condition Nodes* that are able to filter the context data. Next, the output of multiple condition nodes can be aggregated by *Operator Nodes* using operators, such as AND, OR, XOR, etc. Thus, context data is processed from the bottom to the top of the Situation Templates in order to determine the situation.

Figure 2 shows an example of a situation template monitoring a production machine. The situation “Machine stopped” occurs if one of the two modeled conditions is evaluated to *true*, i.e., either the machine has a tool decay over 90% or the available material is below 10. In a similar manner, situations can be modeled for different kinds of entities, ranging from virtual machines in the IT environment to production machines on the shop floor. In previous work, we developed different approaches to map *Situation Templates* onto executable representations, for example, using Complex Event Processing Hirmer et al. (2015, 2016); Franco da Silva et al. (2016). In this paper, we use the *Situation Templates* as a foundation to enable distributed situation recognition. In the following, we use the latest version of *Situation Templates* introduced in Hirmer et al. (2015).

3.2 Foundations of orchestration and choreography

To enable the composition of services, many approaches have been presented in research Nikoo et al. (2020). These can be categorized into orchestration and choreography (Figure 3), where the first coordinates multiple services from a single central entity, while the latter is decentralized and different partners coordinate themselves Peltz (2003).

Both categories can be modeled with so-called *Activities* that represent a single unit of work within an application, which in turn are then connected and ordered *via* edges between them. The difference regarding orchestration and choreography is that within an orchestration, a single partner is executing the activities that, e.g., use services or applications from another partner. In regard to a choreography, multiple partners interact with each other *via* message exchanges and coordinate their own activities, e.g., the first activity in a choreography might send an order from a customer to a seller while this is followed by an activity which sends a production order from the seller to a manufacturer. Additionally, note that a choreography does not have a single owner of the application, since it is a collaboration between multiple partners.

Relevant for the remainder of the paper are adaption methods of a composition in the style of a choreography Leite

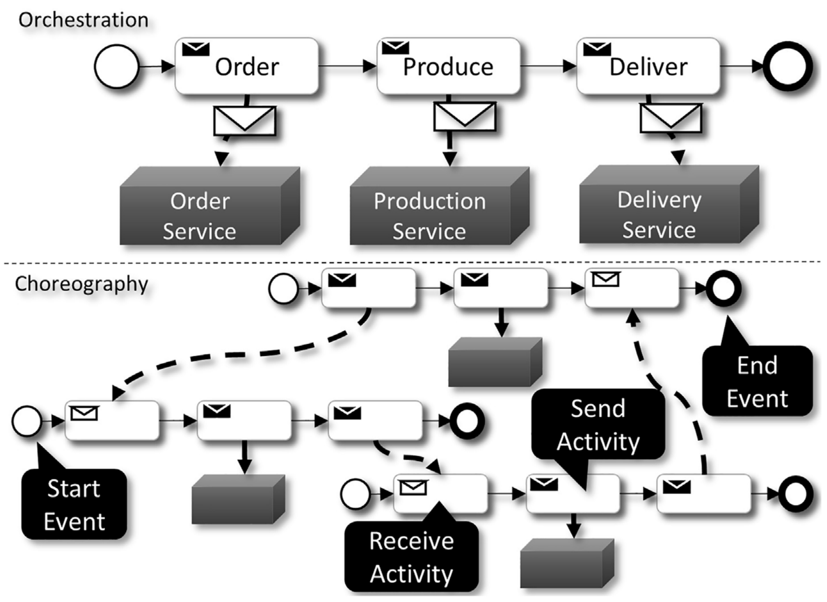

FIGURE 3
Example of an orchestration and choreography.

et al. (2013). One of the more popular approaches is model-based adaptation where the choreography model is changed to fit new (non-)functional requirements by exchanging single or complete subsets of activities and edges. Some of these approaches also use so-called *Variation Points* that specify that a planned adaptation should be performed at that place of choreography. These and other approaches can be applied at modeling, deployment and runtime to enable different kinds of adaptation strategies, such as selecting proper activities, services or whole partner processes.

3.3 Situation adaptive workflows

In the previous work *SitOPT* Wieland et al. (2015), we i) enabled the dynamic and autonomous adaptation of workflows to situations and ii) developed optimization methods for efficient situation recognition and provisioning.

SitRS Hirmer et al. (2015), a part of *SitOPT*, is a centralized, cloud-based middleware implementing concepts to process data from sensors in order to recognize situations, which are instances of *Situation Templates* (*cf.* Section 3.1). This was the basis for the modeling and situation recognition at runtime for the previous approaches used within centralized processes. Applications observe their situations and adapt behavior if necessary *via* the *SitOPT* system. The adaption is done on orchestrated processes, which additionally could use process fragments, a set of process activities, based on the current state of situations presented in Képes et al. (2016). Depending on the situations, the *Situation Handler*, a situation-aware service bus, is responsible for executing the corresponding, situation-dependent process fragment. Also, process activities can be grouped into a so-called *Situational Scope* so that the execution of a workflow depends on whether situations are present or absent while starting to execute the activities within the scope. Both concepts allowed processes to react to changing situations through the use of alternative activities.

In this paper, we extend the previous work by enabling a distribution of the situation recognition and, furthermore, adaptation of choreographies, which involve multiple partners in a distributed environment.

4 Overview of the *DiStOPT* approach

In this section, we give an overview of the *DiStOPT* approach. The details are then described in the next sections: i) the distributed situation recognition and ii) adaptation of choreographies based on situations.

The main concepts are depicted in Figure 4. The basic elements are [1] context data, [2] *Situations*, modeled with [3] *Situation Templates*, and [4] *Situational Scopes*. Context is the basic data which can be retrieved by monitoring the relevant resources, such as people,

1 https://www.espertech.com.
2 https://www.hyperledger.org/use/fabric.
3 https://reactjs.org.
4 https://redux.js.org.

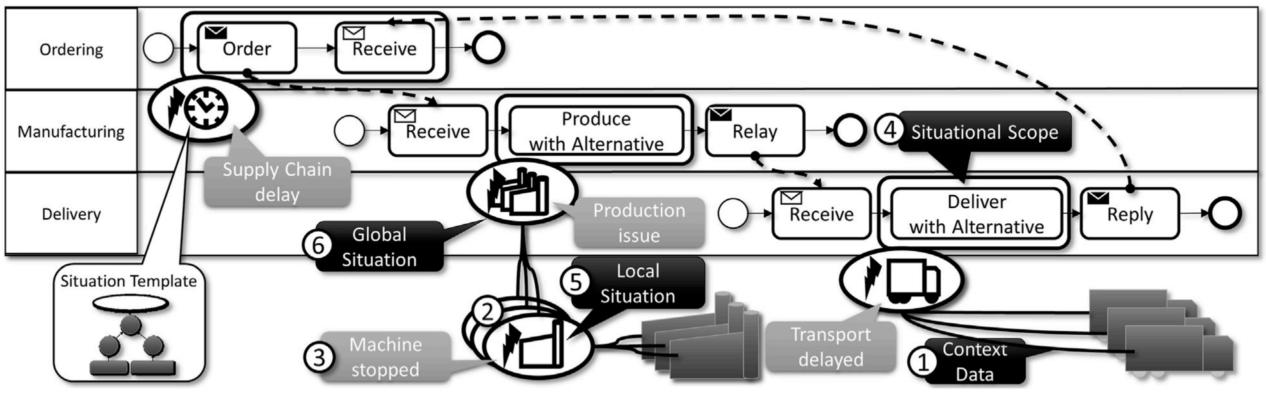

FIGURE 4
Overview of our *DiStOPT* approach based on an exemplary scenario.

manufacturing processes or products, see in [1] Figure 4. However, as this is plain information of a resource, it is often too fine-grained and must be combined with other sources of data to be effectively used. To enable this, we combine the fine-grained and low-level data into situations by using so-called *Situation Templates*, enabling to aggregate context data for various states within the application context, e.g., [3] in Figure 4. As many scenarios are not in a single location, e.g., a company owning multiple manufacturing sites, we introduce local and global *Situations* that distributes the information gathering. Local situations [5] can now be aggregated into global situations [6] and conceptually build a distributed system of data gathering and aggregation. On top of the distributed situation detection, we use so-called *Situational Scopes*, [4] which facilitate the modeling of situation-aware choreographies. A *Situational Scope* describes which activities can be executed when certain situations in the context of a process are present, e.g., deliver the ordered items with an alternative means in case of a transport delay of one of the delivery trucks. In *DiStOPT*, we apply these concepts to choreographies, that need a distributed and decentralized runtime system underneath, allowing the reading of situations, and therefore, situation-awareness of multiple cooperating and distributed processes.

5 Distributed situation recognition

The first contribution of our work refers to i) a distributed, decentralized recognition of occurring situations in order to timely adapt the affected choreographies and ii) the tamper-resistant long-term historical storage for recognized situations using the blockchain technology. As described in Section 3.1, situations are the result of a processed *Situation Template*, which uses context data from different distributed sources as input to calculate if the situation occurs. In other words, a *Situation Template* is instantiated using context data to determine situations. In our scenario, for example, context data sources could be production machines or delivery trucks and a situation could be, for example, “Machine Stopped” or “Transport Delayed”.

5.1 Extended situation templates

To enable a distributed situation recognition, first, an extension of the previously introduced *Situation Templates* is required (cf. Section 3). This includes the ability i) to model context data from different distributed objects in a single *Situation Template* and ii) to use situations calculated by other *Situation Templates* as input, which enables modeling situation hierarchies.

To address the increasing modeling complexity, which comes with the distribution, we introduce a layer-based modeling approach. The goal is to use already modeled *Situation Templates* as input for further *Situation Templates*. Hence, *Situation Templates* can also serve as context data sources for a next layer of Situation Templates.

With these hierarchical *Situation Templates*, it is possible to model situations of a closed space (e.g., a production line) within a single *Situation Template* and then aggregate multiple of these *Situation Templates* into a larger one to describe situations of a larger scope (e.g., a whole factory). This hierarchical approach allows to keep the modeling complexity low, since situations are modeled for a smaller scope, so-called *Local Situations*, and are only then aggregated in an additional step. In this step, local situations are used as context data sources of a larger scope

5 https://spring.io.
6 https://grafana.com.

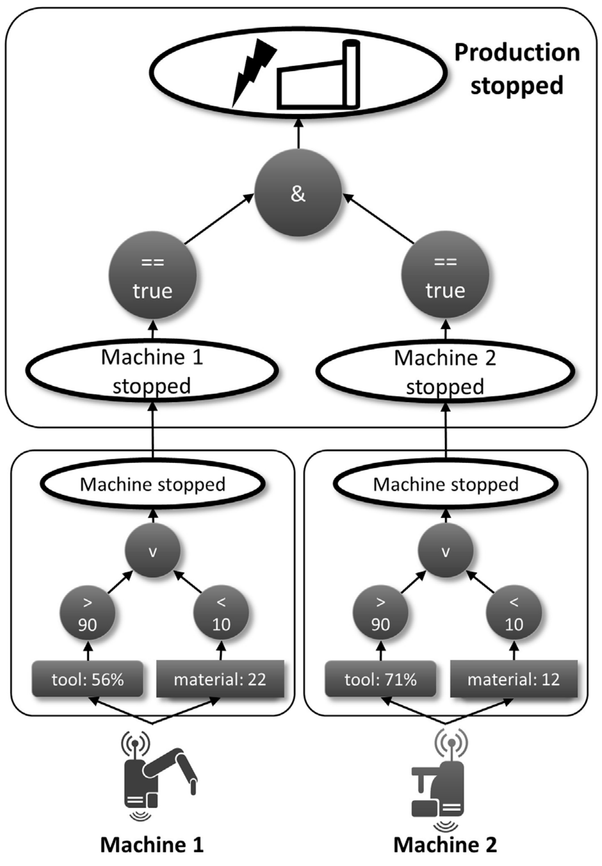

FIGURE 5
Aggregation of situations *via* multiple Situation Templates.

Situation Template, which models a so-called *Global Situation*. In the following, we use the term *Local Situation* for a situation which is valid within a specific scope and the term *Global Situation* for a globally valid situation, possibly consisting of many local situations.

A more detailed example showing such hierarchies of local and global situations is depicted in Figure 5. In this example, instead of modeling one complex *Situation Template* involving all context, condition, operator, and situation nodes for the whole scenario, it makes sense to split it into two *Local Situations*, one for each machine and one *Global Situation* "Production Stopped" to aggregate the local situations. This ensures reusability, modularity, and a clear separation of concerns. As depicted in this example, in the situation "Machine Stopped", it is checked for both production machines (Figure 5, bottom) if the tool decay is higher than 90% or if the amount of materials is smaller than 10. If this is the case, these situations evaluate to true. In the global situation "Production Stopped", it is then only checked if both local situations are evaluated to true and, if this is the case, the global situation also evaluates to true, which could lead to a delay in the production and, thus, to changes in the overlying choreography. Modeling situations in this way, not only the modeling effort can be reduced but also the efficiency of the template evaluation, since the situations can be calculated near to the context data sources instead of in a remote location, e.g., in a cloud environment. This leads to a significant reduction in network latency and load Mormul et al. (2020).

The concepts of local and global situations also offer the advantage of reusability. Once a situation is modeled (e.g., a failure of a production machine), it can be used as context data. As can be seen in Figure 5, the two Situation Templates "Machine Stopped" are reuses of the Situation Template given in Figure 2 and are applied to Machine 1 and Machine 2 as local situations to build the global situation "Production Stopped".

5.2 Situation recognition

After the situations have been modeled, their recognition should be executed in a distributed, decentralized manner. For example, to save costs, it usually makes sense to recognize situations locally that are specific to a certain area, e.g., a room, a production machine, a factory hall, and as close to the involved objects as possible. If edge cloud environments are available on the shop floor, they can be used to compute the local situations. Otherwise, the computation of the situation needs to be shipped to a farther location. Overlaying global *Situation Templates* that involve multiple shop floors and their local situations pre-computed in the edge cloud could then, for example, be evaluated in a backend cloud environment. Hence, it is only necessary to transfer whether the local situations evaluate to true or false to the backend cloud and not all the basic data that is necessary to evaluate them, e.g., the sensor data of a production machine. The aggregation of data potentially decreases the amount of network traffic.

The evaluation of situations is conducted in our work using Complex Event Processing (CEP). Franco da Silva et al. (2016) developed an approach to transform *Situation Templates* into *Complex Event Processing* queries that can be evaluated in a corresponding CEP engine. CEP engines, such as Esper, are lightweight and can be operated in edge and backend clouds, thus, serving the purpose of the distribution strategies presented in this paper. To enable the communication between the different situation recognition events distributed among edge and backend clouds, we developed a lightweight messaging engine Del Gaudio and Hirmer (2020a), which serves the purpose to exchange messages in decentralized architectures, in our case, the exchange of situations. Since also the deployment and configuration of CEP engines is a complex task, we automated this using the OASIS standard TOSCA. This is another important part to conquer the complexity of distributed situation recognition. For more details of our approach, we refer to Franco da Silva et al. (2018).

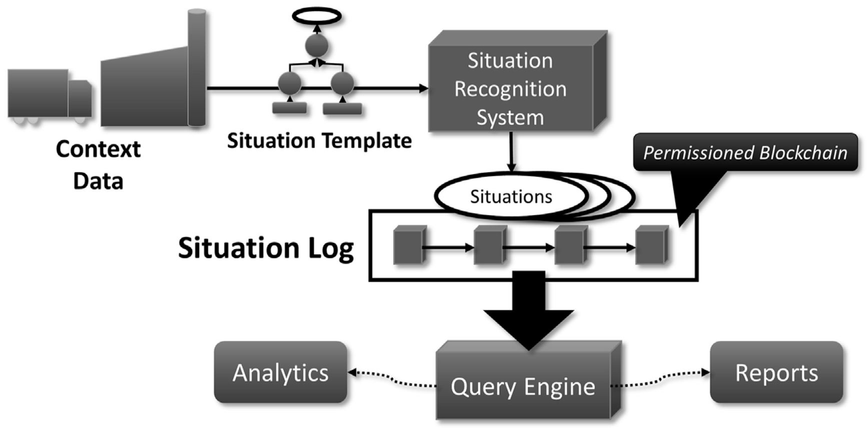

FIGURE 6
Situation Log storing recognized local and global situations in a permissioned blockchain.

5.3 Situation log

In our scenario, in case of failures, it is important to determine which parts failed and why. Especially in safety-critical applications, this information is essential, e.g., railway signalling, nuclear power plants, or vaccine production or transports, as recognized situations may become very important and critical if accidents of any kind occur. In these cases, audits of the recognized situations and succeeding actions are needed. Hence, tampering with situations and actions has to be avoided, i.e., it must be ensured that situations and their involved context data cannot be changed after their recognition. Therefore, it is necessary that recognized situations are stored in a tamper-resistant way across all partners.

To enable this, we utilize the blockchain technology to maintain a distributed and verifiable data store for storing these situations for historical or investigational purposes. This is depicted in Figure 6. A permissioned blockchain, serving as a situation log, stores all occurred situations. A query engine delivers read-only access to these situations by all partners for historical analytics and reporting purposes.

Every recognized situation is reported to the other partners and jointly written into a permissioned blockchain, i.e., an immutable, tamper-resistant situation log. This is done by a consensus mechanism in the blockchain. Depending on the permissioned blockchain system, this consensus mechanism is modular and customizable in a sense that the content of a transaction is checked (i.e., the business semantics) before it can be approved, and a consensus policy is implemented where all or some partners must approve a transaction before it is accepted (i.e., written into the blockchain). To ensure that the recognized situations come from trustworthy sources in a sense that the low-level context data have not been manipulated, signatures can be used, which verify the data's origin and its authenticity. For IoT devices, attribute-based signatures offer the required flexibility, including the use of these attributes to verify properties during data collection (e.g., data accuracy). Gritti et al. Gritti et al. (2019) introduce such a lightweight method. Previous work Przytarski et al. (2021) shows how this method can be combined with blockchain technology.

A permissioned blockchain includes an access control layer so that only authorized users (e.g., the partners) have read and write access to the blockchain. Thus, the data in permissioned blockchains are inaccessible to the public but shared by the partners. Therefore, it is practically impossible to manipulate the stored situations from outside and in. This ensures that all partners have a consistent, tamper-resistant, trustworthy and historical view of the recognized situations.

Unlike in public blockchains where a consensus mechanism must work on a global scale, a permissioned blockchain may utilize a much simpler and more efficient consensus mechanism with better characteristics in terms of speed and resource consumption. Therefore, the consensus model is not predefined, so that this can be freely decided as each consensus model considers different challenges with respect to forking, performance, and security. As the number of participants in a permissioned blockchain network is predefined, it is reasonable to use a consensus mechanism that guarantees finality with some sort of fault tolerance Cachin and Vukolic (2017).

Some permissioned blockchain systems utilize a database system that provide a query engine to query the blockchain for analytical or reporting purposes. The query engine computes the result of a given query by using the local database instance that stores, sometimes just parts of, the replicated blockchain. To also support filtering with complex conditions, we are currently working on a storage engine for blockchain technology that utilizes the triple data model (`entity-attribute-value`)

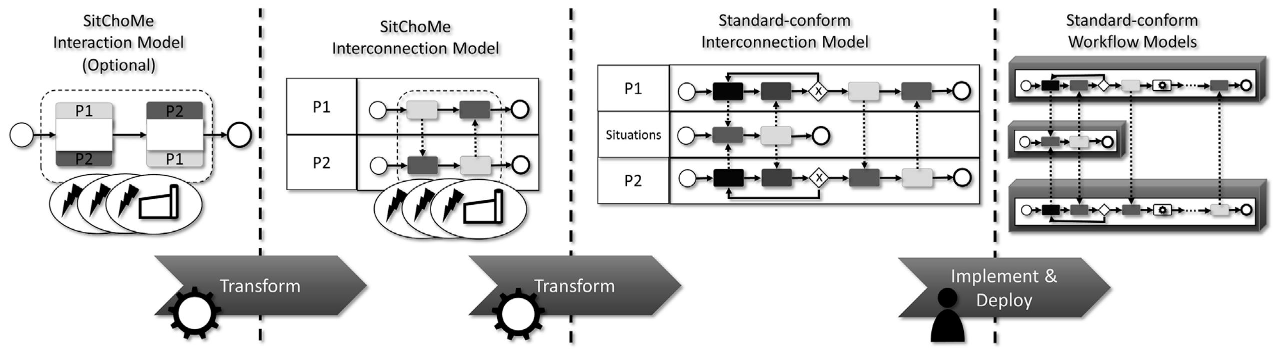
SitChoMe Interaction Model (Optional)
P1
P2
SitChoMe Interconnection Model
P1
P2
Standard-conform Interconnection Model
P1
Situations
P2
Standard-conform Workflow Models
Transform
Transform
Implement & Deploy

FIGURE 7
Overview of the SitChoMe modeling method.

Przytarski (2019) and provides a powerful query engine with a familiar query language. This enables access to present and historical situations even without extensive knowledge of the blockchain system.

6 Situation-aware choreographies

In this section, we describe our contributions regarding the modeling, deployment, and execution of so-called *Situation-Aware Choreographies*. These choreographies coordinate the interaction between multiple business partners and specify situation-aware behavior to react to changing situations within so-called *Situational Scopes* that specify whether a set of activities can be executed or not under a set of recognized situations.

6.1 Modeling of situation-aware choreographies

The modeling of *Situation-Aware Choreographies* is achieved with the so-called *SitChoMe* method depicted in Figure 7. The method builds on so-called *SitChoMe Interaction* or *Interconnection Models*, both specifying the interactions between multiple partners and their behavior according to specified situations.

Modelers can start by modeling a *SitChoMe Interaction Model*, which contains only activities that model the interaction between partners, such as, message exchanges. In addition, a *SitChoMe Interaction Model* specifies sets of activities within so-called *Situational Scopes* (see dashed box around the activities in Figure 7) within a choreography which should only be executed when a specific set of situations is in a desired state at runtime. For example, an alternative ordering of parts from a specific partner should only be done if the forecasted delivery time is not acceptable anymore, or, a choreography specifies to execute an order activity only when a part is already available at the partner. These activities are only executed when the specified situations are active within the current applications' context. In addition, the *SitChoMe* method allows modelers to model the desired *Situation-Aware Choreographies* as a so-called *SitChoMe Interconnection Model*, that is used to model not only interaction activities between partners but local and technical activities, as well.

Our method allows to transform a *SitChoMe Interaction Model* into a *SitChoMe Interconnection Model* in an automated manner, therefore, partners can model the global and public situation-aware interaction model together and afterwards add additional partner specific activities within an interconnection model. The main difference between these models is that partners can have their specific process activities modeled in a more detailed manner (see separate pools in Figure 7).

After the *SitChoMe Interconnection Model* is generated and optionally refined, e.g., by adding partner-specific activities, the *Situational Scopes* within the model are replaced with activities that technically implement the needed situation-aware logic. For example, if a set of activities should only be executed if a set of situations is valid at runtime, a set of activities is generated that evaluates the state of the specified situations before execution. These activities are interacting with a new process (see middle pool in Figure 7 at the interconnection model part) with the goal of retrieving the current state of a situation. The encapsulation of this logic into its own process allows partners to decide whether they share the same process to retrieve the state of situations or each implementing their own, e. g., when the situation is only needed by a single partner.

After the generation of the standard-conform (i.e., BPMN 2.0) interconnection model, the partners implement their part of the overall choreography independently according to the generated SitChoMe model. Therefore, each partner has information what message must be sent at what time inside the process, and additionally, each one can implement their logic

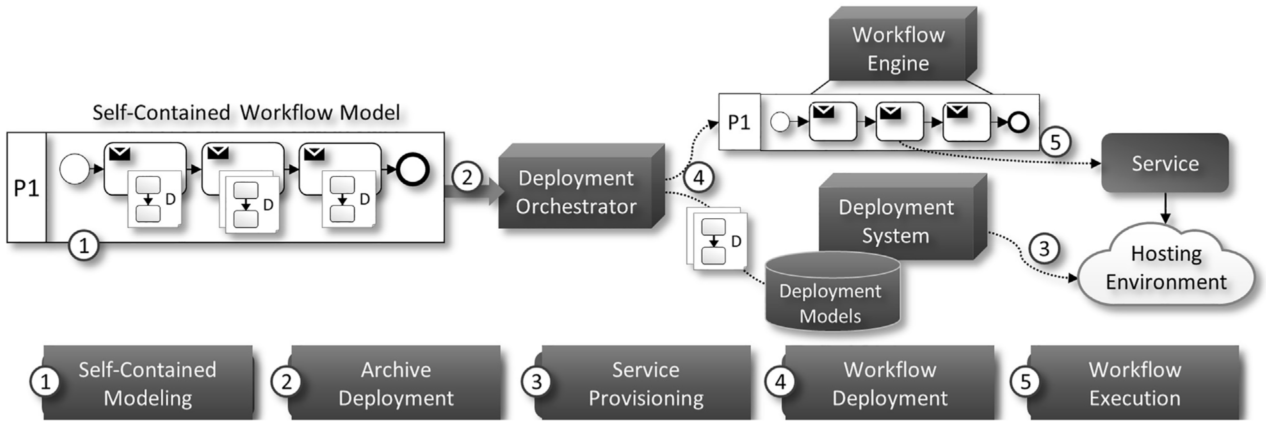

FIGURE 8
Overview of the deployment approach based on our work in Weder et al. (2020).

according to the specified situations. When the partners implemented their parts, each process of the choreography is deployed in their own systems and bound against the partners they communicate with (see Section 6.2 for details).

6.2 Deployment of choreography partner processes

The individual deployment of situation-aware processes that are part of a situation-aware choreography is a challenge, as it is necessary to configure the process to be able to communicate with the services called by its activities and with the situation recognition systems. Based on Weder et al. (2020), users are able to tackle the described problems with *Self-Contained Workflows* and the method depicted in Figure 8.

The concept of our deployment method is the annotation of activities within a process model with so-called deployment models of services which are used by the activities (see [1] in Figure 8). For example, if a local process of a partner needs additional services, they can attach the deployment model that specifies all components which are needed to deploy the service. On the other hand, if an activity uses an already available service, e.g., a service which is used to retrieve the state of situations, only its endpoint information is attached at deployment time. After the workflow model is annotated with the necessary service information, it is packaged into a single archive and is given to the deployment orchestrator (see [2] in Figure 8). The deployment orchestrator is responsible for deploying the services (see [3] in Figure 8) and binding of the workflow to these services for execution at runtime (see [4] and [5] in Figure 8).

In context of the *DiStOPT* approach, the deployment allows each partner of a choreography to implement their own local processes with the help of already available or yet to be deployed services which can then be attached to the activities of the processes, thus, reducing the development and deployment effort.

As we have scenarios which are highly distributed and there is no central control, we based the deployment of our choreographies on previous work on distributed deployment Képes et al. (2019) and extended it for decentralized control Wild et al. (2020). Both approaches together achieve the deployment of the situation-aware partner workflows and situation recognition components. It builds on top of a message bus in each deployment orchestrator that is used to communicate with other message busses *via* messaging. For example, one orchestrator can send a message to another to deploy a component in a specific environment unavailable to the other. For decentralized deployment, i.e., there is no single orchestrator managing all local deployment on the partners' side, we use an approach based on *Deployment Choreographies*. Multiple partners participate in a deployment choreography, each with their own automatically generated local deployment processes, that deploy only the components in their respective environments. In addition, each partner can start the deployment, by starting the local deployment process and with the help of the message bus notify the other partners to start their deployment processes, and in case of choreographies, make everything available.

6.3 Execution of choreography partner processes

The execution environment for the created and deployed situation-aware partner workflows which together form a situation-aware choreography consists of a *Workflow Engine*, *Situation-Aware Message Bus* and the *Situation Recognition*

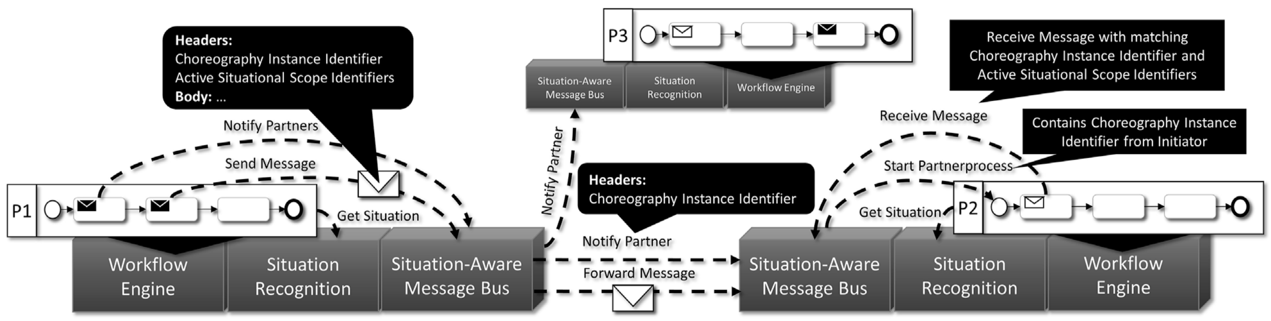
Headers:
Choreography Instance Identifier
Active Situational Scope Identifiers
Body: ...
Notify Partners
Send Message
P1
Get Situation
Workflow Engine
Situation Recognition
Situation-Aware Message Bus
P3
Situation-Aware Message Bus
Situation Recognition
Workflow Engine
Notify Partner
Headers:
Choreography Instance Identifier
Notify Partner
Forward Message
Receive Message with matching Choreography Instance Identifier and Active Situational Scope Identifiers
Receive Message
Contains Choreography Instance Identifier from Initiator
Start Partnerprocess
Get Situation
P2
Situation-Aware Message Bus
Situation Recognition
Workflow Engine

FIGURE 9
Overview of the Execution Approach based on Képes et al. (2021)

components for each partner (see Figure 9). These components support retrieval of the current state of situations, and, therefore, can be used to start choreographies when selected situations are available. As our modeling method uses situations to execute specific activities, some messages must be queued by the message bus in case a message can only be given to a partner workflow when a certain situation is available.

The first interaction between the workflows and the execution environment is the initiation of the overall choreography by instantiating each local partner process. This has to be managed by the system, as multiple partners in a choreography may start a choreography instance, i.e., the choreography could be modeled in such a way that each partner may start the overall choreography. Therefore, the other partners have to be informed that a new instance has started, and each partner must start an instance of its part of the choreography in its local environment (see edges with *Notify Partners* and *Notify Partner* in Figure 9).

To implement situation-aware behavior, the situation-aware workflows, generated initially by our method and then implemented in the last step of the method, retrieve the current state of situations (*Get Situation* in Figure 9) and, based on the received state, wait, abort or continue with execution, following the specification of the *Situational Scope* it originates from. When messages need to be shared, the Message Bus is used to pass these to the respective partners (send and receive messages in Figure 9). To keep execution consistent between the partners, e.g., that a partner is further ahead in the execution as the other, not only situations are used to synchronize each other but message queuing is used as well. If a partner expects to receive a message which was already sent by another partner, the message bus must be able to queue the message and pass it to the workflow instance when appropriate. Appropriate in this case means that messages must fit the running instance of the choreography by checking the *Choreography Instance Identifier*, and it must fit the current state of situations, i.e., the active *Situational Scopes* by evaluating the active *Situational Scope Identifiers* of the message. In case it fits, the message can be passed to the process. Otherwise, the state between partners differs at that point in time, which can happen due to inconsistencies between the situation's state between partners. However, the state of the situations eventually gets consistent over time and the right message can be received from the message bus *via* the identifiers in the message as one of the partners will adapt by actively switching to the modeled alternative defined in the *Situational Scope*.

7 Implementation architecture of the *DiStOPT* system

The system architecture of the *DiStOPT* system consists of a set of distributed components which enable gathering and aggregation of context data, the evaluation of situations at runtime, and their use within a *Situation-aware Choreograpy*.

7.1 Architecture

The system is divided into modeling time and runtime (see Figure 10).

Situation Templates (cf. Section 5.1) are modeled with the *Situation Modeler* (see upper left in Figure 10). The *Situation Modeler* is browser-based and can be operated either stand-alone on a local machine or as a cloud service. Modeled *Situation Templates* are transformed into executable processing models as described in Del Gaudio and Hirmer (2020a) and Section 5.2. In this step, *Complex Event Processing* (CEP) queries are generated from *Situation Templates*. The processing model consists of abstract operations and directed links between them, based on the Pipes-and-Filters pattern. After transformation, the *Situation Template* is passed to the *Situation Manager*, which deploys all necessary software on the involved devices. This includes, for

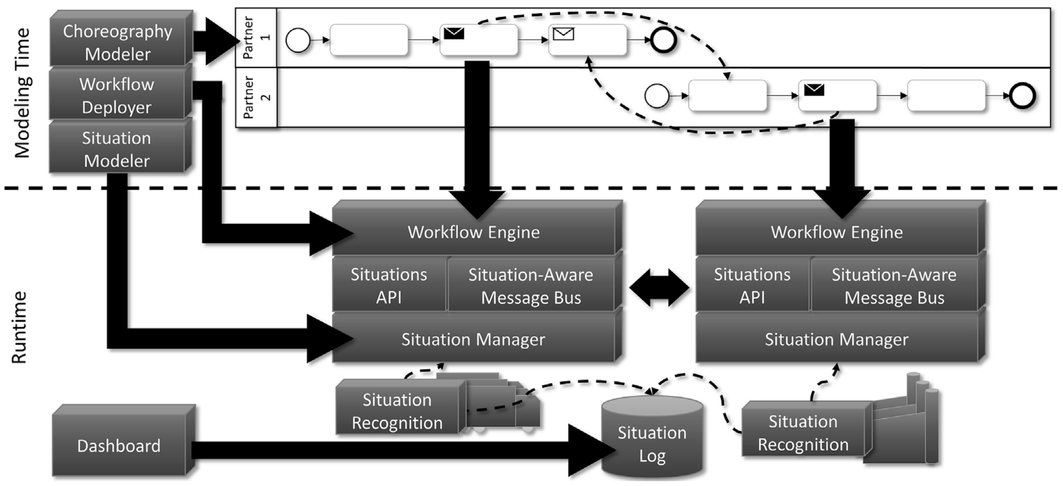

FIGURE 10
Implementation architecture of the *DiStOPT* system.

example, the CEP engine to evaluate the CEP queries. Running the situation recognition locally enables early filtering and, thus, less communication overhead. The CEP queries are shipped to the CEP engine for evaluation. Whenever a situation is recognized, the information is stored on the situation log(cf. Section 5.3) and the *Situation Manager* is notified. Situations stored in the situation log can be visualized and inspected in a dashboard, which can be individualized for each partner.

The modeling of a *Situation-aware Choreography Model* (cf. Section 6.1) is done within the *Choreography Modeler* (see upper left in Figure 10), which allows modeling choreographies as *SitChoMe Interaction* and *Interconnection Models* with *Situational Scopes*. After modeling, the choreography is transformed within the same component in a *SitChoMe Interconnection Model*, if needed, which replaces the *Situational Scopes* into language-compliant logic that is used to interact with the runtime system and to control the execution of the activities within the modeled *Situational Scopes*. These models are then implemented with internal and technical activities that use, e.g., services needed for each partner to achieve the overall business goal. The final processes are then deployed in the runtime environment (cf. Section 6.2) with the *Workflow Deployer* (see upper left in Figure 10), which not just deploys the process on a *Workflow Engine* (see middle in Figure 10), it also binds and optionally deploys the needed services for the process to run properly. Next, the process is executed (cf. Section 6.3). At runtime, the deployed processes communicate with the Situations API *via* the *Situation-Aware Message Bus* (see middle in Figure 10). Both components use the underlying *Situation Recognition* component to retrieve the current state of situations. The *Situation Recognition* component aggregates the low-level context data, executes the Situation Template and then determines the current state (cf. Section 5.2). Data retrieval can be achieved in a pull and push-based fashion, i.e., the data retrieved is either send to the *Situation Manager* or it is fetched from the resources *via* adapters running on them directly.

8 Related work

This section describes related work in the areas of distributed situation recognition in Section 8.1 and context-aware choreography adaptation in Section 8.2.

8.1 Related work on distributed situation recognition

In related work, approaches exist for distributed situation recognition using ontologies, e.g., Fang et al. Fang et al. (2008). These approaches do not achieve the latency required in real-time critical scenarios, such as Industry 4.0 Sanchez et al. (2020), due to time-consuming reasoning. The goal of our approach is to achieve distributed situation recognition times in the range of milliseconds, as we showed in Franco da Silva et al. (2016). Many approaches using ontologies are in the range of seconds to minutes, even without distribution Wang et al. (2004); Dargie et al. (2013). Using machine learning leads to similar limitations regarding latency Attard et al. (2013). In the area of distributed Complex Event Processing (CEP), Schilling et al. Schilling et al. (2010) aim at integrating different CEP systems using a common meta language. This could be beneficial for our distribution because we would not be limited to one type of execution engine. However, in the work of Schilling et al. Schilling et al.

(2010), the queries have to be hand-written and distributed. This is difficult for domain experts, e.g., in Industry 4.0, who do not have extensive computer science knowledge. In our approach, we provide an abstraction by *Situation Templates* that can be modeled using a graphical user interface, as for example, the *Situation Modeler* from Section 5 and Section 7. Other approaches in distributed CEP, e.g., by Schultz-Moller et al. Schultz-Møller et al. (2009), follow the concept of automatic query rewriting. Here, CEP queries are split up using automated rewriting and are distributed on different operators based on a cost model, which is mostly based on CPU usage in the different nodes. Since there are many aspects, such as data protection or security, that play a role in distributing the CEP queries, this only can be known by an expert user. Furthermore, approaches exist that enable a massive distribution of sensors, e.g., by Laerhoven and Gellersen Van Laerhoven and Gellersen (2004) in cloths, to detect activities of the person wearing the cloth. This is similar to detecting the situation in the edge, but there is no concept presented in Van Laerhoven and Gellersen (2004) to integrate the activities with other activities from different edges or create a global situation involving different locations.

Overall, related work in the area of distributed situation recognition focuses mostly on very specific concepts and technologies, such as CEP. In contrast, we base our approach on the more abstract and generic *Situation Templates*, which we extend to achieve the desired distribution of the situation recognition.

8.2 Related work on context-aware choreography

In related work, it is identified that the increased usage of paradigms such as Industry 4.0 or IoT need further coordination Machorro-Cano et al. (2020); Belkeziz and Jarir (2020) and should be context-aware Perera et al. (2014). As already mentioned in the fundamentals section (see Section 3.1 and Section 3.2) to create context-aware applications we need to be able to adapt applications to the current context. According to a systematic literature review by Leite et al. Leite et al. (2013) adaptation of a choreography can be categorized according to the properties time, i.e., design time or runtime, and the amount of needed human intervention, i.e., automated or manual. Model-based methods are related to Model-Driven Development (MDD) and emphasize the adaptation view from a higher-level. Alternatively, adaptation is needed when certain thresholds are not met, hence, measurement-based methods are valid as they trigger the re-composition of a choreography. According to Leite et al. a multi-agent system (MAS) is based on individual agents that together try to achieve goals or execute tasks, however, each agent has its own autonomy to solve its local problem. The formal-methods-based approaches build on formal models such as finite state machines or process calculus, such as, π-calculus to describe the behavior of a choreography application. As another category Leite et al. present the semantic-reasoning-based approaches which use ontologies to describe the services between partners and try to improve the interoperability between these by reasoning over the communication. The last category are proxy-layer approaches which use the Proxy pattern, a software layer, which intercepts messages and handles adaptation behind it. For example, by selecting the recipient or adjusting the message itself.

Our approach is based on design time models using situation-awareness at runtime. The proposed solution builds on the context modeled to support the modeling and execution of adaptable choreography applications with the use of the presented situational scopes.

Autili et al. Autili et al. (2017); Filippone et al. (2020); Filippone et al. (2022) present a method for context-aware choreographies. The general idea is to model choreography models with so-called variation points which are later implemented *via* fitting services (orchestrated *via* coordination delegates) according to the needs of the choreography and the context of it. In contrast, our approach is based on modeling alternative logic which must be modeled to fit the situations and context.

Bucchiarone et al. Bucchiarone et al. (2017) present a context-aware and adaptable orchestration method which uses different workflow fragments and goals. By using the available fragments and goals their system is able to generate the needed workflow achieving the overall business goal. The context is used to constrain which fragments can be used and how the goal can be achieved. Although, the proposed concept allows for the context-aware adaptation of application logic, it is only supporting it from a centralized and orchestrated perspective not considering multiple partners and therefore choreographies.

Chand et al. (2011) achieve context-aware choreography and their adaption *via* a context-aware service bus. The bus routes messages between different services while monitoring the current context and adapts the message routing accordingly. Our approach uses a service bus as well to coordinate different partners, however, *DiStOPT* uses a distributed system which enables each partner to exploit local and global Situation Templates and to communicate in a decentralized manner.

Cao et al. (2015) propose a framework for context-aware service orchestration. The framework orchestrates services based on policies and context, which in turn leads to exchange of services used at runtime, and therefore, a simple form of adaptation. Although the proposed framework enables adaption according to context, it only allows to adapt the selection of the services for each task in a choreography. This disregards context to be a main part of the choreography itself, e.g., using alternative paths within a choreography when context changes occur, and therefore using different activities on a logical level, allowing fine-grained situation-awareness.

Cheng et al. (2017) introduce an approach for *"Situation-Aware Dynamic Service Coordination in an IoT Environment"*. In their paper, the authors introduce an event-driven, service-oriented IoT service coordination platform architecture as well

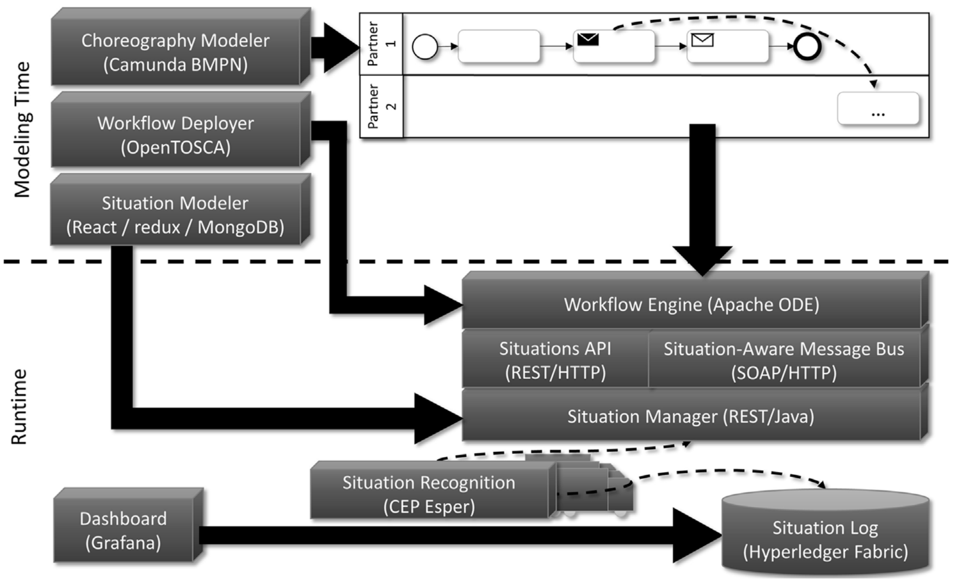
Modeling Time
Runtime
Choreography Modeler (Camunda BMPN)
Workflow Deployer (OpenTOSCA)
Situation Modeler (React / redux / MongoDB)
Partner 1
Partner 2
...
Workflow Engine (Apache ODE)
Situations API (REST/HTTP)
Situation-Aware Message Bus (SOAP/HTTP)
Situation Manager (REST/Java)
Situation Recognition (CEP Esper)
Dashboard (Grafana)
Situation Log (Hyperledger Fabric)

FIGURE 11
Architecture of the Prototype and used Technologies.

as a situational event pattern to enable automaton-based situational event detection. In contrast to their work, we use Situation Templates to enable situation modeling for non-technical users. Furthermore, our focus lies not only on single processes but rather on the situation-aware adaptation of choreographies that are involving multiple partner processes.

9 Validation of the DistOPT prototype

To validate our approach, we implemented a prototype based on our motivating scenario (cf. Section 2), the implementation architecture as shown in Figure 10, and based on previous work of Del Gaudio and Hirmer (2020a,b); Képes et al. (2020, 2019); Wild et al. (2020), which integrates the Situation Recognition and Situation-Aware Choreography parts of *DiStOPT*.

For the prototype, we used the following technologies: The Situation Recognition part is based on technologies such as *Esper*[1], *Hyperledger Fabric*[2], *React*[3], *Redux*[4], *Spring*[5], *Grafana*[6], and *MongoDB*[7]. For the choreography modeling and execution, we used Camunda *BPMN*[8], *Apache ODE*[9] and *OpenTOSCA*[10]. The architecture of the prototype is shown in Figure 11.

7 https://www.mongodb.com.

8 https://camunda.com.

9 https://ode.apache.org.

10 https://www.opentosca.org.

Situation Templates for our scenario can be modeled in a user-friendly web interface, the *Situation Modeler*. Using this tool, we created the Situation Templates "Transport Delayed" as well as "Production Stopped" (cf. Section 5). To implement the *Situation Modeler*, we use the libraries React for building interactive UI elements and Redux for managing the UI state. We implemented a REST API with Spring to store and retrieve situation models into and from a MongoDB database. Also, we implemented the *Situation Modeler* with multi-tenancy, so that multiple partners can model without getting insight into each other's internals.

Our test environment consist of different IoT devices, including Raspberry Pis (version 4) and Arduinos. These IoT devices simulate the production machines and delivery trucks of our motivating scenario and produce context data as input for the modeled Situation Templates. The context data is generated automatically using scripts. Next, we transform the modeled Situation Templates into CEP queries and evaluate these queries based on the context data using the CEP engine Esper. Situations occur randomly based on the generated data and will then lead to adaptations in the choreographies. For the situation log, the tamper-resistant data store for recognized situations, we use the blockchain technology Hyperledger Fabric from IBM. The situations "Transport Delayed" and "Production Stopped" are stored in the *Situation* Log once they occur. The situations can then be displayed in the Grafana dashboard.

On the side of *Situation-Aware Choreographies*, we built our prototype based on Camunda BPMN, Apache ODE and OpenTOSCA. While the first is a modeling and runtime environment for BPMN OMG (2011) workflows, Apache ODE is an engine for BPEL OASIS (2007) workflows. OpenTOSCA is a TOSCA-based OASIS (2013) ecosystem for the modeling, deployment and management of Cloud, IoT and Quantum applications. To model the *SitChoMe Interaction and Interconnection Models* for our scenario and to transform these into standard-compliant models, we implemented an extension for Camunda to model the desired choreography with situational scopes and transform them afterwards. As we used Camunda, the modeling within our approach is based on BPMN and can be implemented and executed on the Camunda engine. However, the implementation can also be achieved using the workflow language BPEL, which can be used within the OpenTOSCA ecosystem. The ecosystem implements the presented self-contained workflow deployment approach, enabling partners to package their workflows into a single package and to deploy whenever needed. The execution system is based on a Situation-Aware Service Bus, which is a main component within the ecosystem. The bus allows different partners to communicate with each other *via* HTTP messages with JSON or SOAP. For communication within a choreography, each service bus can communicate with another *via* HTTP and, in case of IoT scenarios needing a small footprint for messaging, also *via* MQTT.

9.1 Discussion

The previously described prototypical implementation serves as validation of our approach. For the implementation, our goal was to focus on efficiency and lightweightness in order to apply it to scenarios with limited resources, which is the case, for example, in the Internet of Things or in Industry 4.0. Using the CEP engine Esper enables recognition of situations even on resource-limited devices in a timely manner, as we also evaluated in previous work Franco da Silva et al. (2016). Furthermore, by providing graphical modeling tools for situations and choreographies, we ease using our approach and broaden the potential user group also to non-technical domain experts. Especially the use of Situation Templates, an established means for situation modeling Häussermann et al. (2010); Zweigle et al. (2009), helps in modeling situations more efficiently. For choreography modeling and execution, we build on existing standards, such as BPMN and BPEL, which allows for a more stable and future-proof system. Furthermore, for deployment purposes, we use the TOSCA standard, which allows creating reusable deployment models, which can also be visualized in a graphical manner to become human-readable.

Overall, our approach and its according implementation aims at achieving usability by domain experts as well as providing a lightweight system that can be applied even to resource-limited environments. Using the described standards and technologies, we were able to realize this in our prototypical implementation. In the next section, we explain how we applied our system in practical research projects.

9.2 Application in real-world scenarios

For validation purposes, we were able to apply our approach in real-world scenarios in the scope of the research projects "Optimization and adaptation of situation-aware applications based on workflow fragments (SitOPT)" and "Industrial Communication for Factories (IC4F)," respectively.

In SitOPT, the goal was to adapt production processes in factories based on occurring situations. For example, if a production machine were to fail, another machine could take over until the issue is resolved. This requires the adaptation of overlying processes. To do so, we integrated the situation recognition and workflow adaptation approach of this paper. As described in Franco da Silva et al. (2016), we were able to recognize situations in near-real time. This is crucial since a timely recognition of issues in production environments can limit downtimes by timely adapting the processes and, thus, decrease costs.

In IC4F, we applied the approach of situation recognition to an autonomous transport scenario in a smart factory. In this scenario, self-driving forklifts were in place that transport goods between different shelves or load them onto trucks. In case of issues with one of the forklifts, the product distribution and delivery process needed to be adapted accordingly, e.g., so that one forklift can replace another one based and adjust its delivery priorities. Here, we were able to integrate and validate our approach in a real scenario.

10 Summary and future work

In this article, we presented the *DiStOPT* approach and its developed concepts used within scenarios like Industry 4.0. One of the goals of the approach is to enable partners to collaborate within a distributed situation-aware application that is able to adapt itself according to current situations at a partner site.

DiStOPT consists of i) an extension of Situation Templates to model local and global situations, ii) a distributed situation recognition system, and iii) the SitChoMe method that allows modeling, deployment and execution of situation-aware choreographies.

Our approach builds on distributed recognition of situations and their aggregation in a tree-based model, enabling partners to observe the state of multiple distributed and heterogeneous resources/entities. To prevent partners from manipulating the state of situations, the logs of situations are stored in a tamper-resistant blockchain for historical and analytical purposes. On top of the situation recognition system, the *DiStOPT* approach provides the modeling and execution of situation-aware choreography applications. These allow to model different partner interactions and activities, and additionally, offer situation-awareness *via Situational Scopes*. To validate our approach, we developed the DiStOPT prototype using open-source technologies and applied it to scenarios taken from industry cooperations.

In future work, we plan to evaluate and extend our concepts to automotive scenarios, for example, in software-defined environments in cars and for driving.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

The contributions are divided into the ones related to choreographies, which is the expertise of the members of the Institute of Architecture of Application Systems, namely UB, KK, and FL and the ones related to situation recognition, the expertise of the members of the Institute of Parallel and Distributed Systems, namely PH, DD, BM, DP, and MM. Furthermore, the contributions in regard to blockchain technology are provided by DP, contributions in regard to situation modeling by PH and MM, contributions in regard to situation recognition by DD, and contributions in regard to modeling and adapting choreographies by UB and KK. Professors FL and BM serve as supervisors of the work.

Funding

This work is funded by the DFG project DiStOPT (252975529).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Attard, J., Scerri, S., Rivera, I., and Handschuh, S. (2013). "Ontology-based situation recognition for context-aware systems," in Proceedings of the 9th International Conference on Semantic Systems, Graz, Austria, September, 2013 (New York, NY, USA: Association for Computing Machinery), 113–120. I-SEMANTICS '13. doi:10.1145/2506182.2506197

Autili, M., Inverardi, P., Perucci, A., and Tivoli, M. (2017). "Synthesis of distributed and adaptable coordinators to enable choreography evolution," in Software Engineering for Self-Adaptive Systems III. Assurances. Editors R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese (Cham: Springer International Publishing), 282.

Belkeziz, R., and Jarir, Z. (2020). An overview of the iot coordination challenge. *Int. J. Serv. Sci. Manag. Eng. Technol. (IJSSMET)* 11, 99–115. doi:10.4018/ijssmet.2020010107

Bucchiarone, A., Marconi, A., Pistore, M., and Raik, H. (2017). A context-aware framework for dynamic composition of process fragments in the internet of services. *J. Internet Serv. Appl.* 8, 6. doi:10.1186/s13174-017-0057-0

Cachin, C., and Vukolic, M. (2017). "Blockchain consensus protocols in the Wild (keynote talk),". Editor A. W. Richa (Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik of Leibniz International Proceedings in Informatics LIPIcs), 1, 16. 1–1. doi:10.4230/LIPIcs.DISC.2017.131st International Symposium on Distributed Computing (DISC 2017)

Cao, Z., Zhang, X., Zhang, W., Xie, X., Shi, J., and Xu, H. (2015). "A context-aware adaptive web service composition framework," in 2015 IEEE International Conference on Computational Intelligence Communication Technology, Ghaziabad, India, February, 2015, 62–66. doi:10.1109/CICT.2015.68

Chand, J., Sengupta, S., Kanjilal, A., and Bhattacharya, S. (2011). CA-ESB: Context aware enterprise service bus. *Int. J. Comput. Appl.* 30, 1–8. doi:10.5120/3626-5062

Cheng, B., Wang, M., Zhao, S., Zhai, Z., Zhu, D., and Junliang, C. (2017). Situation-aware dynamic service coordination in an iot environment. *IEEE/ACM Trans. Netw.* 1, 2082–2095. doi:10.1109/TNET.2017.2705239

Dargie, W., EldoraMendez, J., Möbius, C., Rybina, K., Thost, V., et al. (2013). "Situation recognition for service management systems using owl 2 reasoners," in 2013 IEEE International Conference on Pervasive Computing and Communications Workshops, San Diego, CA, July, 2013 (PERCOM Workshops), 31–36. doi:10.1109/PerComW.2013.6529452

Del Gaudio, D., and Hirmer, P. (2020a). A lightweight messaging engine for decentralized data processing in the internet of things. *SICS Softw. -Inensiv. Cyber-Phys. Syst.* 35, 39–48. doi:10.1007/s00450-019-00410-z

Del Gaudio, D., and Hirmer, P. (2020b). Seamless integration of devices in industry 4.0 environments. *Internet Things* 12, 100321. doi:10.1016/j.iot.2020.100321

Fang, Q., Zhao, Y., Yang, G., and Zheng, W. (2008). "Scalable distributed ontology reasoning using dht-based partitioning," in *The semantic web*. Editors J. Domingue and C. Anutariya (Berlin, Heidelberg: Springer Berlin Heidelberg), 91.

Filippone, G., Autili, M., and Tivoli, M. (2022). Synthesis of context-aware business-to-business processes for location-based services through choreographies. *J. Softw. Evolu. Process* 34, e2416. doi:10.1002/smr.2416

Filippone, G., Autili, M., and Tivoli, M. (2020). "Towards the synthesis of context-aware choreographies," in 2020 IEEE International Symposium on Software Reliability Engineering Workshops, Coimbra, Portugal, October, 2020 (ISSREW). doi:10.1109/ISSREW51248.2020.00072

Franco da Silva, A. C., Hirmer, P., Breitenbücher, U., Kopp, O., and Mitschang, B. (2018). Customization and provisioning of complex event processing using tosca. *Comput. Sci. Res. Dev.* 33, 317–327. doi:10.1007/s00450-017-0386-z

Franco da Silva, A. C., Hirmer, P., Wieland, M., and Mitschang, B. (2016). SitRS XT – towards near real time situation recognition. *J. Inf. Data Manag.* 7, 4. doi:10.5753/jidm.2016.1573

Gritti, C., Önen, M., and Molva, R. (2019). Privacy-preserving delegable authentication in the internet of things. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, April 2019. 861

Häussermann, K., Hubig, C., Levi, P., Leymann, F., Siemoneit, O., Wieland, M., et al. (2010). "Understanding and designing situation-aware mobile and ubiquitous computing systems," in Proceedings of the International Conference on Computer, Electrical, and Systems Science, and Engineering 2010, Amsterdam, Netherlands, March, 2010. *ICCESSE 2010*.

Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., and Leymann, F. (2015). "SitRS - a situation recognition service based on modeling and executing situation templates,"in Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing, Heraklion, Greece, June, 2015. Editors J. Barzen, R. Khalaf, F. Leymann, and B. Mitschang (IBM Research Report), RC25564

Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., Sáez, S. G., et al. (2016). Situation recognition and handling based on executing situation templates and situation-aware workflows. *Computing* 1, 163–181. doi:10.1007/s00607-016-0522-9

Képes, K., Breitenbücher, U., Gómez Sáez, S., Guth, J., Leymann, F., and Wieland, M. (2016). "Situation-aware execution and dynamic adaptation of traditional workflow models," in *Service-oriented and cloud computing*. Editors M. Aiello, E. B. Johnsen, S. Dustdar, and I. Georgievski (Cham: Springer International Publishing), 69.

Képes, K., Breitenbücher, U., Leymann, F., Saatkamp, K., and Weder, B. (2019). "Deployment of distributed applications across public and private networks," in Proceedings of the 23rd International Enterprise Distributed Object Computing Conference (EDOC 2019)IEEE), 236–242. doi:10.1109/EDOC.2019.00036

Képes, K., Leymann, F., Weder, B., and Wild, K. (2021). "Sidd: The situation-aware distributed deployment system," in *Service-oriented computing – ICSOC 2020 workshops*. Editors H. Hacid, F. Outay, H.-y. Paik, A. Alloum, M. Petrocchi, M. R. Bouadjenek, et al. Dubai, December, 2021 (Springer International Publishing), 72.

Képes, K., Leymann, F., and Zimmermann, M. (2020). "Situation-aware updates for cyber-physical systems," in *Service-oriented computing*. Editor S. Dustdar (Cham: Springer International Publishing), 12.

Leite, L. A. F., Ansaldi Oliva, G., Nogueira, G. M., Gerosa, M. A., Kon, F., and Milojicic, D. S. (2013). A systematic literature review of service choreography adaptation. *Serv. Oriented Comput. Appl.* 7, 199–216. doi:10.1007/s11761-012-0125-z

Machorro-Cano, I., Alor-Hernández, G., Olmedo-Aguirre, J. O., Rodríguez-Mazahua, L., and Segura-Ozuna, M. G. (2020). *IoT services orchestration and choreography in the healthcare domain*. Burlington, VT: Springer International Publishing, 429. –454. doi:10.1007/978-3-030-26488-8_19

Mormul, M., Hirmer, P., Stach, C., and Mitschang, B. (2020). "Dear: Distributed evaluation of alerting rules," in Proceedings of the IEEE 13th International Conference on Cloud Computing, Beijing, China, October, 2020. Editors L. Khan, G. Huang, and Beijing (IEEE), CLOUD '20), 158–165. doi:10.1109/CLOUD49709.2020.00034

Nikoo, M. S., Babur, O., and van den Brand, M. (2020). "A survey on service composition languages," in Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (New York, NY, USA: Association for Computing Machinery). MODELS '20. doi:10.1145/3417990.3421402

OASIS (2013). *Topology and orchestration specification for cloud applications (TOSCA) version 1.0*. Burlington, VT: Organization for the Advancement of Structured Information Standards OASIS.

OASIS (2007). *Web services business process execution language (WS-BPEL) version 2.0*. . Burlington, VT: Burlington, VT: Organization for the Advancement of Structured Information Standards OASIS.

OMG (2011). *Business process model and notation (BPMN) version 2.0*. Object Management Group OMG.

Peltz, C. (2003). Web services orchestration and choreography. *Computer* 36, 46–52. doi:10.1109/MC.2003.1236471

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Commun. Surv. Tutorials* 16, 414–454. doi:10.1109/SURV.2013.042313.00197

Przytarski, D., Stach, C., Gritti, C., and Mitschang, B. (2021). A blueprint for a trustworthy health data platform encompassing IoT and blockchain technologies. *EPiC Ser. Comput.* 76, 56

Przytarski, D. (2019). "Using triples as the data model for blockchain systems," in *BlockSW/CKG@ISWC'19*.

Sanchez, M., Exposito, E., and Aguilar, J. (2020). Industry 4.0: Survey from a system integration perspective. *Int. J. Comput. Integr. Manuf.* 0, 1017–1041. doi:10.1080/0951192X.2020.1775295

Schilling, B., Koldehofe, B., Pletat, U., and Rothermel, K. (2010). Distributed heterogeneous event processing: Enhancing scalability and interoperability of cep in an industrial context." in Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, Cambridge, United Kingdom, July, 2010. New York, NY, USA: Association for Computing Machinery, 150–159. doi:10.1145/1827418.1827453

Schultz-Møller, N. P., Migliavacca, M., and Pietzuch, P. (2009). "Distributed complex event processing with query rewriting," in Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, Nashville, TN, July, 2009 (New York, NY, USA: Association for Computing Machinery). doi:10.1145/1619258.1619264

Van Laerhoven, K., and Gellersen, H. (2004). Spine versus porcupine: A study in distributed wearable activity recognition. *Eighth Int. Symposium Wearable Comput.* 1, 142–149. doi:10.1109/ISWC.2004.40

Wang, X. H., Zhang, D. Q., Gu, T., and Pung, H. K. (2004). "Ontology based context modeling and reasoning using owl," in *IEEE annual conference on pervasive computing and communications workshops* (Proceedings of the Second), 18–22. doi:10.1109/PERCOMW.2004.1276898

Weder, B., Breitenbücher, U., Képes, K., Leymann, F., and Zimmermann, M. (2020). "Deployable self-contained workflow models," in Proceedings of the 8th European Conference on Service-Oriented and Cloud Computing (ESOCC), Heraklion, Greece, March, 2020 (Springer), 85–96. doi:10.1007/978-3-030-44769-4_7

Wieland, M., Schwarz, H., Breitenbücher, U., and Leymann, F. (2015). "Towards situation-aware adaptive workflows," in Proceedings of the 13th Intl. Conference on Pervasive Computing and Communications Workshops: 11th Workshop on Context and Activity Modeling and Recognition, St. Louis, MO, March, 2015.

Wild, K., Breitenbücher, U., Képes, K., Leymann, F., and Weder, B. (2020). "Decentralized cross-organizational application deployment automation: An approach for generating deployment choreographies based on declarative deployment models," in Proceedings of the 32nd Conference on Advanced Information Systems Engineering (CAiSE 2020), Grenoble, France, June, 2020 (Springer International Publishing of Lecture Notes in Computer Science). doi:10.1007/978-3-030-49435-3

Zweigle, O., Häussermann, K., Käppeler, U.-P., and Levi, P. (2009). "Supervised learning algorithm for automatic adaption of situation templates using uncertain data," in Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human. doi:10.1145/1655925.1655960