MAS Organisations to Adapt your Composite Service[⋆]

Mario Henrique CRUZ TORRES[1], Victor NOËL[2], Tom HOLVOET[1], and Jean-Paul ARCANGELI[2]

[1] Department of Computer Science
Katholieke Universiteit Leuven
3001 Heverlee, Belgium
{mario.cruztorres,tom.holvoet}@cs.kuleuven.be

[2] Institut de Recherche en Informatique de Toulouse
Université de Toulouse
118, route de Narbonne, 31 062 Toulouse, France
{victor.noel,jean-paul.arcangeli}@irit.fr

Abstract. Adapting composite web-services is a concern of many researchers from the services community and a requirement of the industry. We propose the CASAS (Composable, Adaptive, Service, Agent System) framework that provides mechanisms to monitor and pro-actively adapt composite services. The framework integrates concepts of Service-Oriented Computing and Agent Organisations, offering monitoring and adaptation mechanisms to deal with adaptation in service compositions. CASAS is an improvement on related work in that it offers a high-level model that allows the definition and enforcement of global constraints for the service composition. We explain CASAS in detail and conclude by showing how one can use it to create an adaptable composite service written in the Business Process Execution Language (BPEL).

Keywords: web-services, adaptation, organisations, multi-agent systems, architecture

1 Introduction

The globalised world of business has created new demands for the architecture of distributed applications. These demands were shifted again with the creation of globally distributed supply chains [14]. The service-oriented computing paradigm provides concepts satisfying the demands in this distributed environment [13].

Nowadays, complex business processes are modelled as composite services using BPEL (Business Process Execution Language). However, BPEL is not suited

[⋆] This paper is published in the proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond (MONA+) collocated with the 8th IEEE European Conference on Web Services (ECOWS) held on December 1, 2010 in Ayia Napa, Cyprus.

to work in very dynamic environments, leading to research on how to adapt processes written in this language. We focus on the problem of adapting a composite service in order to deal with global constraints, such as the End-to-End Quality of Service (QoS), and the problem of preventing Service Level Agreement (SLA) violations.

We show how we can adapt an executing BPEL process in order to avoid SLA violations using the CASAS framework. We contribute to the state-of-the-art of composite services adaptation with our agent-based model.

The rest of the paper is organised as follows. Section 2 depicts a real-world problem used as our motivation scenario. In Section 3 we describe the concepts and abstractions used in our solution. We explain our solution's architecture in Section 4, then we describe our prototype in Section 5. To conclude the paper, we show the state-of-the-art related to web-service adaptation in Section 6. Finally, we present our conclusions and future research directions in Section 7.

2 Scenario

To illustrate the core ideas of this paper, we use a simple scenario from the Supply Chain Management domain based on interviews with the industrial partners of the DiCoMAS project[3].

A fourth-party logistics company (4PL) takes care of its clients' logistic procedures such as the transportation of materials between the client's factories. A 4PL has contracts with a number of carriers, called third-party logistics company (3PL), that do the actual transportation. The 4PL's goal is to save time and money for its clients, by optimising transportation and business processes.

Each time our example 4PL receives a transportation request, it creates a transportation plan using an Advanced Planning System (APS). The transportation plan is composed of a number of activities, each activity representing a transportation that should be made between two locations. The planning system splits the original transportation request in a number of sub-transports, because, normally, 3PLs are specialised in specific regions. Finally the transportation plan is written as a BPEL process and, after a first selection of 3PLs, deployed in a BPEL engine.

To execute this process, the 4PL's BPEL engine invokes the selected 3PL's web-services, informing them about the constraints, such as time constraints, monitoring constraints, etc.

How can we meet the quality requirements of executing a BPEL process within a specific time frame, even in the presence of partner failures, is the problem that we want to solve.

This problem is illustrated by our example 4PL, that needs to do the transportation within a limited time period, as specified in the plan, but sometimes

[3] DiCoMAS (Distributed Collaboration using Multi-Agent System Architectures) is a SBO (Strategisch BasisOnderzoek – Strategic Basic Research) project funded by IWT (Instituut voor de Aanmoediging van Innovatie door Wetenschap – Technologie in Vlaanderen - Belgium)

a partner that previously committed to do a transportation has problems and is not able to execute its part in the plan. For example, a small carrier, that has just one truck, is assigned a sub-transportation, but, suddenly, has a broken truck that needs to be repaired. In this situation, the 4PL needs to find another carrier capable of doing the transportation for that specific sub-transportation, preferably within the same quality constraints.

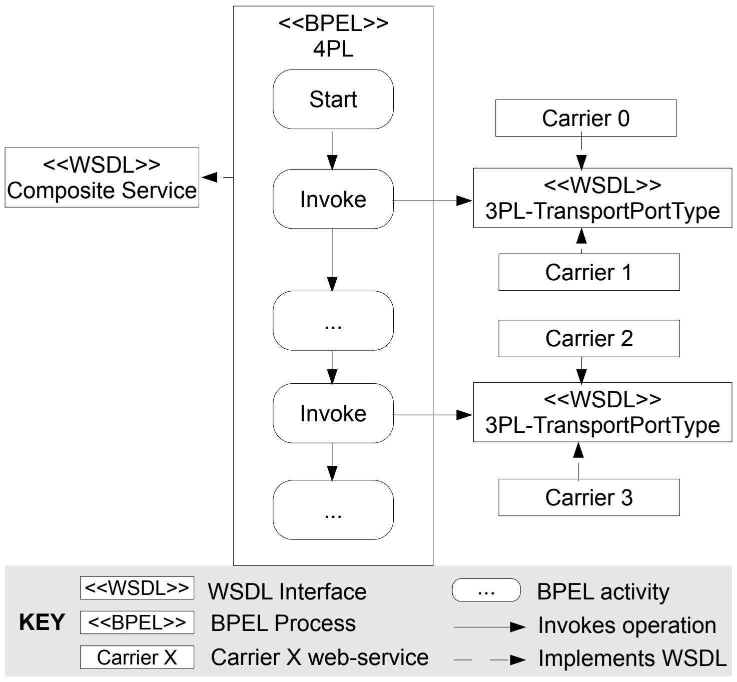

Fig. 1. Transportation plan deployed in a BPEL engine.

Figure 1 depicts a simplified transportation plan. The transportation plan is deployed on a BPEL engine and is seen as an internal service within the 4PL, simply called `Composite Service` here. Each `Invoke` activity in the transportation plan is executed by an external service, through the `3pl-TransportPortType` interface, which is implemented by partner companies web-services. Each 3PL must comply to the specified QoS, in this case the time to do the transportation.

In the next section, we show the concepts used in our solution to the problem of dynamic adaptation of composite services.

3 Another Level of Abstraction to Deal with Adaptations

The *de facto* standard for service composition is the BPEL language. BPEL can deal with primitive forms of adaptation, using `Dynamic Partner Links` and

`Endpoint References` [7]. However, it is hard, if not impossible, to model the adaptations and its constraints required by a composite service using only BPEL.

We add another level of abstraction to the composite service model to solve the adaptation problem. This layer, called the organisation layer, explicitly represents the interactions between all the services participating in the composition, the adaptation constraints and the expected behaviour of the composition.

3.1 The Macodo Organisation Model

The Multi-Agent Systems (MAS) research community has a body of knowledge on Organisational Models. In this community, there are two distinct visions regarding these models: a) organisation being a first class entity, with its properties, states, laws [16]; b) organisation mainly as a process, composed by a set of steps to be taken by different actors [5].

MAS Organisational Models cope with collaboration between autonomous entities, called agents, working and interacting together, cooperatively or not, to achieve an organisation goal [4]. In our work we rely on the Macodo Organisation Model (Middleware Architecture for COntext-driven Dynamic agent Organizations), which defines the organisation as a first class entity with its own dynamics and separated from the participating agents.

The Macodo Organisation Model copes with context-driven dynamic organisations. It allows us to model complex collaborations between different entities, the agents, and to specify the rules that will trigger actions to adapt these collaborations [16].

Figure 2 depicts the domain model of the Macodo Organisation Model. The main concepts of the model are: `Organisation`, which contains roles and role positions; `OrgContext` which keeps the context information needed by the organisation; `Role`, which constrain the behaviour expected from the agents; and `Context`, which is the contextual information needed by the organisation.

When joining an `Organisation`, an `Agent` takes a `RolePosition` in order to play a `Role`: this is represented by a `RoleContract`.

An `Organisation` encapsulates organisation rules used to adapt the collaboration. It actively inspects its context (`OrgContext`), *i.e.* the set of all the participating agent contexts, to enforce the organisation rules that need to deal with global constraints. A `Role` constrains the agent behaviour towards the organisation, requiring the agent to provide a set of capabilities and context. Using the `Roles` and agents' `Context`, the `Organisation` allows or denies the participation of the `Agents` in the collaboration.

There are two possible ways for an organisation to adapt: a well balanced combination of the two is the key to an adaptive organisation.

The organisation can access the context information of the agents and check if the rules are being satisfied. That way, the adaptation is triggered by the organisation that keeps monitoring the collaboration between the agents. If one rule is not satisfied, it can change the state of a role and open a new role position, so that another agent can try to play that role in the organisation.

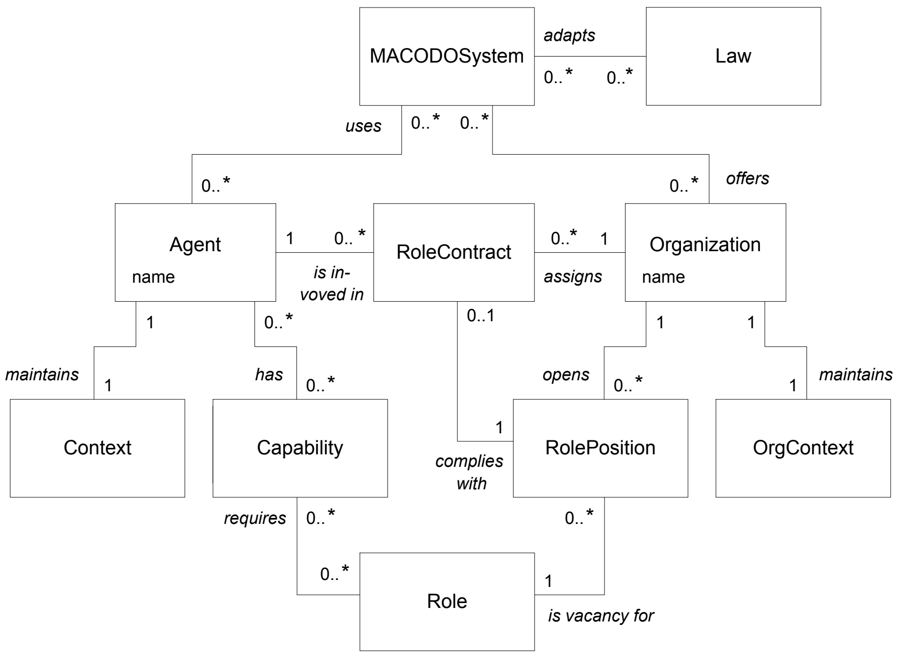

Fig. 2. Domain Model of the Macodo Context-Driven Organisational Model [16].

Another way for an organisation to adapt is through the agents behaviour. The agents can have a pro-active behaviour and monitor their own state. They can actively decide to leave or join an organisation. When an agent leaves an organisation, the organisation changes the state of the played role, opening a new role position, leading to its adaptation.

3.2 Mapping Macodo Organisations to Composite Services in BPEL

A BPEL process consists of: a) the BPEL code, which defines the execution flow; b) Web Service Description Language (WSDL) interfaces for the different consumed services; c) WSDL interface for the provided service (the composite service itself). A composite service specified in BPEL is made of a set of activities that are executed in a specific order. One special type of activity is declared using the `Invoke` construct, which invokes an operation in a partner link web-service.

In BPEL the communication with other web-services is done through the `PartnerLinks`, which define the relation between the BPEL process and partner web-services. Partner web-services are referenced by their `Port Type`, which is a set of abstract operations defined in WSDL.

We established a mapping between Web-services and Macodo Organisation concepts in order to create the organisation layer. This mapping is illustrated in Table 1.

Table 1. Mapping between Web-services and Macodo concepts

Web-services	**Macodo**
BPEL process	Organisation
PortType	Role
PartnerLink	Role Position
SLA	Capability
SLO	Agent Context

One BPEL process corresponds to one analogous `Macodo Organisation`. For each `Partner Link` specified in the BPEL process, we have a `Role Position` in the organisation. We have a `Role` for each `Port Type` in the BPEL process. The SLAs are specified in terms of required `Capabilities` and, finally, Service Level Objectives (SLOs) are specified as `Agent Context`.

We assume that the SLAs and SLOs are well understood and specified to the point that no human intervention is needed anymore. Otherwise it would be impossible to let the system adapt itself during runtime.

The CASAS framework uses this mapping and the BPEL process description to define the structure, such as the number of `Roles` and `Role Positions` of an organisation. Each BPEL process instance provides the correct flow of activities needed by the composite service and can be seen as the organisation functional behaviour. The agents operate the runtime binding to the real service providers and act as their representatives in the system.

Based on this mapping, separately from the BPEL process definition, the organisation provides a way to define adaptation rules that deal with the runtime adaptation that can occur during the process execution. For that, the agents provide context information to the organisation. But also, agents contain monitoring mechanisms and have pro-active behaviour to trigger adaptations (*e.g.* an agent can monitor its SLOs and predict that an SLA will be broken).

Figure 3 shows all the entities that collaborate in our system to create an adaptive composite service. It shows a BPEL process, the adaptation rules, the `Macodo Organisation`, and the `Agents` that can participate in the organisation. The left most `Agent` is taking the `position2 RolePosition`, the other `RolePosition`, called `position1`, is not taken by any `Agent`. An `Agent` taking a `RolePosition` means that the agent is playing a specific `Role` in one `Macodo Organisation`. When the BPEL engine invokes an operation in a `PartnerLink`, the CASAS framework intercepts and redirects that invocation to the right `Agent`, which will, in turn invoke the operation on the actual web-service.

In the following section we introduce the CASAS framework, then we detail a prototype that gives example of adaptation rules and agent behaviours.

4 Casas Architecture

To handle the service-oriented concerns, CASAS rests on the Apache ServiceMix Enterprise Service Bus (ESB). In particular, this ESB provides the Apache Ode

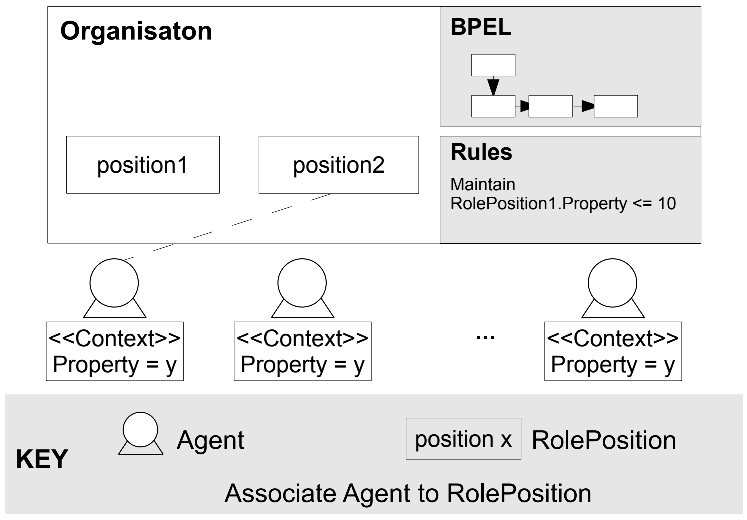

Fig. 3. Conceptual solution integrating Macodo organisations, BPEL, and agents.

BPEL engine to execute the workflows and the Apache CXF component to access external web services. A Normalised Message Router (NMR) is also present and is used by CASAS to intercept messages exchanged in the ESB and redirect them to the correct recipients.

In Fig. 4, depicting the high-level architecture of CASAS, we see 3 main elements:

a) The Macodo System, which is responsible for managing the organisations (one per workflow instance). Each created organisation contains the rules that have to be enforced (the adaptation rules), a set of role positions to maintain (the partner links) and the functional behaviour for the organisation (the workflow). The Macodo System enforces the rules in the organisation by opening and closing role positions when needed and also allows agents with a position in the organisation to play their role.
b) The set of agents in the system, which are responsible for joining organisations if they can take open role positions. They play their role when asked by the organisation and maintain a context that can be consulted by the organisation when required. Agents are parametrised by the web-service they represent, a social behaviour to decide when to join or leave an organisation and a monitoring behaviour to update their current context as well as trigger adaptations.
c) The CASAS system: each time a new workflow instance is launched, it creates a new organisation. It also creates the agents that will represent the different available partner web-services. The CASAS system uses the NMR API to set up the connection between: (i) each organisation and a particular instance of the workflow; (ii) each agent and represented web-service. The CASAS systems does this by listening and modifying exchanged messages.

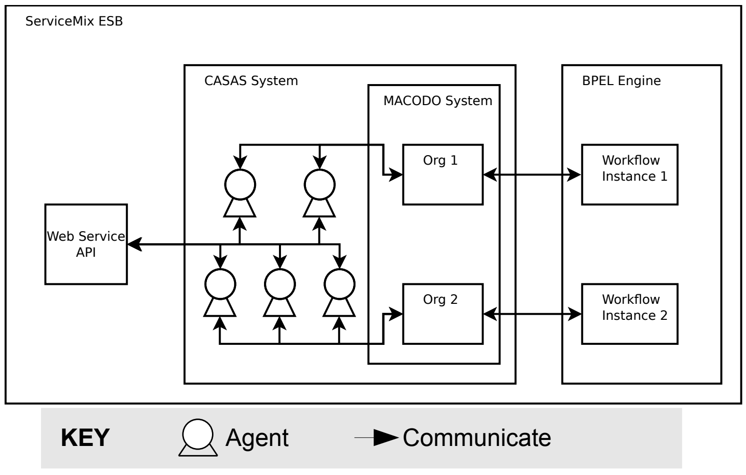

Fig. 4. CASAS Framework Architecture.

From a service-oriented point of view, the CASAS system is responsible for dynamically providing the best partner services to the workflow instances. This selection is done in compliance to the SLA of each partner web-service. The BPEL engine doesn't know about the changes that can happen to the partner web-services, since it communicates with endpoints provided by the CASAS system. The CASAS system makes the connection between the BPEL engine and the real partner web-services, acting as a type of evolved proxy.

The MAS, explained in the next section, is transparent to the BPEL process and to the partner web-services, being used just internally by the CASAS system. From an agent-oriented point of view, the MAS is composed by the agents and their environment, *i.e.* the organisation that regulates their interactions.

4.1 Multi-Agent System

We used the SpEArAF (Species to Engineer Architectures for Agent Frameworks) development process to design the architecture and implement the MAS used in our prototype. The interested reader can find more details about SpEArAF in the present proceedings [11].

SpEArAF completes methodologies that mainly focus on the design of functionality, by promoting the engineering of application-specific frameworks for the development of multi-agent applications. By defining dedicated frameworks, the idea is to provide specific types of agents that fit functional requirements. Developers can rely on the framework when designing and implementing a MAS, they don't need to deal with operational concerns and can focus on the agent's functional behaviours. In the prototype presented in this paper, our objective is to make a framework for species of agents that are part of a Macodo organisation, described previously, and that interact with web-services through an ESB.

Frameworks are realised by assembling software components (possibly by reuse) in architectures for agents. Then, when programming the MAS, *hotspots* in the frameworks can be instantiated (possibly with sub-architectures) by the framework user to specify the behaviour of the agents using a set of agent-oriented and application-specific programming primitives defined by the framework. In practice, the architectures are defined using the MAKE AGENTS YOURSELF[4] tool that supports SPEARAF and the frameworks are implemented with JAVA.

Agent Architecture Figure 5 depicts the architecture of one agent and its bindings to the Macodo System. As many agents as needed can be created at runtime and will have this architecture and bindings. An agent interacts with the system through a set of interfaces: he can receive *system events* about opened and closed role positions, based on that information he can *start a contract* with an organisation (*i.e.* take a position) and, when it has a contract for a role position, the system can send him requests to *play* his role and consult his *context.*

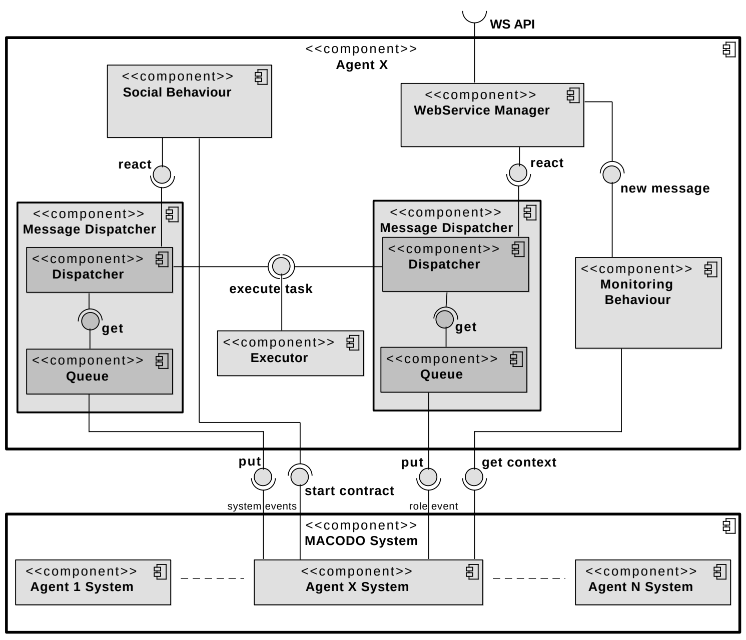
WS API
<<component>> Agent X
<<component>> Social Behaviour
<<component>> WebService Manager
react
react
new message
<<component>> Message Dispatcher
<<component>> Dispatcher
get
<<component>> Queue
execute task
<<component>> Executor
<<component>> Message Dispatcher
<<component>> Dispatcher
get
<<component>> Queue
<<component>> Monitoring Behaviour
put
start contract
put
get context
system events
role event
<<component>> MACODO System
<<component>> Agent 1 System
<<component>> Agent X System
<<component>> Agent N System

Fig. 5. Agent Architecture.

In the agent component itself we can see: a) a set of operational components (the frozen-spots of the framework) that implements the agent dynam-

[4] http://www.irit.fr/MAY

ics (`Message Dispatchers` to serially react to events) and the domain-specific mechanisms (`WebService Manager`) he uses; b) a set of behavioural components (the hotspots of the framework) that must be implemented to give a behaviour to the agent: handling organisation (`Social Behaviour`) events and playing the role (`Monitoring Behaviour`)

In terms of agent dynamics, agents are handling two types of events in parallel and reacting directly to them. Events from the system are handled serially by the `Social Behaviour` component, which decides if the agent should take a position, eventually leaving another one. Play events are handled serially by the `WebService Manager` component that forwards the invocation to the real service represented by the agent. The `Context` of the agent is managed by the `Monitoring Behaviour` component. This component extracts information from the exchanged messages with the real service, which are, in turn, provided by the `WebService Manager` component.

The presented application-specific framework provides an agent architecture that explicitly exposes, through its components, the concepts available at the design level: social behaviour and monitoring behaviour. These behaviours can be implemented by the application developer (framework user) using the application-specific primitives without worrying about implementation of operational concerns. For example, in our case to implement the `Social Behaviour`, developers can use the `start contract` primitives using information provided in `system events`.

This architecture facilitates the development of the whole application by enabling clear separation of concerns and explicitly manipulating concepts from the organisation model, starting from the design down to the implementation.

5 Prototype

The prototype was written in Java, according to the described architecture. The software requires the BPEL process definition, the adaptation rules, the partner web-services addresses, the agents bounded to these services, and the agent behaviours.

5.1 Adaptation Rules

Adaptation rules are defined per role position and are parametrised by a function responsible for checking that the QoS indicators of the agents (its context) satisfies the defined SLA for the partner (a threshold). The organisation dynamics is hard-coded: it continuously checks the context threshold, it accepts, refuses and revises `RoleContracts` based on this context information.

5.2 Behaviours of the Agents

As said before, there are two parts of the agent behaviour: (i) social behaviour for joining and leaving the organisation; (ii) monitoring behaviour for managing the context. We implemented them as follow in a reactive way.

An agent that receives an event for an open position will try to start a contract if he has the capability of playing this role, *i.e.* he has a matching `PortType` and required SLA. If the agent decides to take a role position but the position is already taken, or if the organisation refuses to give the position to him, or if the position is closed while playing it, then the agent waits for new open positions.

The monitoring behaviour is strongly dependent to the type of context required by the organisation. This context is determined depending on the adaptation rules presented before. We used the delivery time of the 3PL as the context in the studied scenario. The 3PL provides this information through its web-service operations.

5.3 Runtime

When there is a request to start a workflow, CASAS instantiates the agents, with the configured behaviour, and the organisation, with its adaptation rules. Role positions are opened and agents join them. The organisation accepts the first agent that has the needed capabilities and provides the required context to take the role position.

The agents always update their context based on the delivery time reported by the 3PLs. If one agent does not comply with the organisation rules anymore, the organisation closes its role position and opens a new one. The organisation uses the agent's context and its own context to enforce a global constraint, which in our scenario is the total time to execute the transportation.

5.4 Future Extensions

In more complex scenarios, the context could be learnt by the agent based on the exchanged data, or acquired and revised by interacting with other agents. Agents could pro-actively and collectively decide not to join an organisation based on their context and let another agent take an open position. The realisation of more evolved personal and collective behaviours is out of scope of this paper but is an important concern in the agent community and in particular in the agent-oriented software engineering sub-field.

Another extension is that the organisation could relax its constraints when there is no agents able to take a role position. This could also be seen as an adaptation rule.

6 Related Work

Service adaptation is a wide research domain. In this section, we discuss a number of approaches and tools that allow the adaptation of composite services, more specifically, the adaptation of composite services based on BPEL.

The most common form of service adaptation is through late-binding, also called vertical adaptation [12]. This type of adaptation doesn't change the structure of the composite service, in terms of the order of executed activities for instance. Instead, it changes the service implementations.

The work of Anja Strunk and others [15] tackles the adaptation problem in controlled environments, where the alternative services are known at design time. The designer of the composite service indicates, at design time, which are the alternative services that are part of the composition.

The infrastructure is composed by three components: a BPEL engine, a monitoring component and a rebinding component. The BPEL engine is responsible for executing the BPEL process. The monitoring component watches events and triggers the rebinding component when SLA violations are detected. The rebinding component replaces failing services with equivalent ones, which was previously specified at design time. The main difference between this approach and ours is that we provide a high-level model which can be used to express the adaptation rules. Another difference is the architecture, our approach uses an ESB to decouple the adaptations from a specific BPEL engine.

Anis Charfi *et al.* [1] proposes a plug-in architecture for self-adaptive web service compositions. In their approach, the orchestration engine is extended with adaptation plug-ins. Each plug-in has a well-defined objective and is developed by domain experts.

The plug-ins are written in the form of aspects, the engine is based on BPEL4AOP, defining pointcuts and advices. The plug-ins can be loaded at runtime and engine weaves the aspects specified in the plug-ins into the running BPEL process. This solution provides a mechanism to adapt any BPEL activity even before is has happened, because of the use of aspects. Our approach does not assume any particular BPEL engine, because we do not need to modify it to insert monitors or actuators. Instead, we use the ESB infrastructure to intercept messages sent by the BPEL engine, decoupling our solution from a particular BPEL engine implementation.

Another approach to adapt BPEL processes is through the use of an ESB connected to a BPEL engine and to a service adaptation engine. Massimiliano Colombo *et al.* [3] applied the ideas of Autonomic Computing, such as self-configuration, self-healing, and context-awareness to create a platform called SCENE. This platform is used to tackle the problem of dynamically reconfiguring composite services.

SCENE provides a rule language that is interpreted on a Drools engine. The rules are checked at runtime and are used to realise the correct bindings between the BPEL engine and the concrete services. The rules are specified in terms of events that are generated by activities specified in the BPEL definition.

To adapt a running BPEL process, the platform intercepts all events and reroute them to the Rule engine. The rule engine checks what rules match the event and process it, possibly leading to the rebinding of a service.

The SCENE platform architecture implements the Message Router Enterprise Integration Pattern [8], with the rules defining the routing logic. SCENE is a very good solution and our architecture resembles it. The main difference is that with our Organisation model it is possible to write rules that take global constraints into account, using the `OrgContext` for instance. An issue with the SCENE approach is that it is not possible to rebind services before problems arise because the platform always react to events that already happened. In our solution it is possible to specify pro-active agent behaviours, allowing the rebinding before problems happen.

Annapaola Marconi *et al.* [10] approach provides adaptation through the concept of Adaptable Pervasive Flow (APF). APFs are specified using standard control elements, such as sequence, choice, and parallel operators. The flows are modelled in a way that they are logically attached to physical entities, and can be used to model workflows that are related to specific objects.

In this work, the researchers created a new language called Adaptable Pervasive Flow Language (APFL), which is an extension to BPEL. The new constructs take the context into account in order to adapt the execution of the business process, providing, for instance, more alternative flows than the standard BPEL constructs. This work provides an interesting way to solve adaptation problems and the main difference with our approach is in our Organisation model and with pro-active adaptation to avoid future problems. Another difference is that with our approach it is possible to use standard BPEL engines.

Recently, Philipp Leitner *et al.* [9] proposed a framework called PREvent, which is a system that integrates monitoring, prediction, and adaptation of service compositions. The main goal of the PREvent framework is to adapt service compositions in order to prevent SLA violations.

The framework mainly consists of three components: Composition Monitor, SLO Predictor, and the Composition Adaptor. The Composition Monitor is responsible for monitoring the runtime data. Prediction of violations are handled by the SLO Predictor, which uses learning techniques to identify the services that can cause SLA violations in the future. Finally the Composition Adaptor component is responsible for identifying and applying adaptation actions. With our approach, we can define rules that deal with the organisation context, global constraints, instead of focusing on individual web-services. Besides that, the organisation model provides concepts that can directly deal with adaptation concerns, such as the organisation rules, role positions, etc.

From the field of MAS, tackling the adaptation of a composition of partners has been studied in [2] in the context of manufacturing control. In this work, in the same way that sequences of services must be provided by different partners in a composite service, containers have to be manufactured by different machines. The objective of the system is to be as efficient as possible while handling the dynamicity of the system (new containers arriving, machine failures, priorities…). Machines, operators and containers are embodied as agents. Here, the composition as well as the adaptation is expressed only from the point of view of the agents by means of local interactions and results in the self-organisation of the whole system.

The main difference between the mentioned works and ours is that ours provide a high level model, the organisation model, of the composite service, its constraints and partner services. Using our model it is possible to reason about the adaptations needed by the entire composition and not just by specific services.

7 Conclusion and Future Work

The main contribution of this work is to provide a high-level model which encompasses the composite service, partner services and the adaptation rules needed by the composition. This high-level model is provided by our framework abstractions, such as `Organisations`, `Roles`, `Role Positions`, `Capabilities` and `Agent Context`, opening new possibilities for the adaptation of composite services, specifically in terms of satisfying global constraints for the composition, which can be specified in terms of the organisation context (`OrgContext`).

Another contribution of our work is that it allows to pro-actively adapt the service compositions. Instead of simply reacting to events and adapting the composition afterwards, agents can have pro-active behaviour and ask to leave the composition, before problems happen.

Service adaptation is a concern from the research community due to the dynamic nature of many problem domains and due to still not having a standard solution for adaptation. We borrowed ideas from the MAS research community, which has experience dealing with problems that demand adaptation. We presented the CASAS framework which uses the concepts from the Macodo Organisation Model and from Composite web-services, showing the mapping between two research domains concepts in order to deal with the adaptation problem.

As future work, we will quantitatively evaluate our approach, focusing on providing and enforcing global constraints on the service composition, such as End-to-End QoS, using for this real world data provided by our project partners.

Furthermore, we plan to investigate other MAS techniques that can enrich our framework, as the Adaptive Multi-Agent Systems (AMAS) theory [6], which focus on self-organising cooperative systems and can provide useful concepts to the problem of adapting composite services.

References

1. Charfi, A., Dinkelaker, T., Mezini, M.: A plug-in architecture for self-adaptive web service compositions. IEEE International Conference on Web Services 0, 35–42 (2009)
2. Clair, G., Kaddoum, E., Gleizes, M.P., Picard, G.: Self-Regulation in Self-Organising Multi-Agent Systems for Adaptive and Intelligent Manufacturing Control. In: IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO). pp. 107–116 (2008)
3. Colombo, M., Di Nitto, E., Mauri, M.: SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In: Dan,

A., Lamersdorf, W. (eds.) ICSOC 2006, LNCS, vol. 4294, pp. 191–202. Springer Berlin / Heidelberg (2006)

4. DeLoach, S.: OMACS: A framework for adaptive, complex systems, pp. 76–98. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models (2009)
5. Dignum, V.: The Role of Organization in Agent Systems, chap. I, pp. 1–17. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models (2009)
6. Gleizes, M.P., Camps, V., GeorgÃ©, J.P., Capera, D.: Engineering systems which generate emergent functionalities. In: Weyns, D., Brueckner, S.A., Demazeau, Y. (eds.) EEMMAS 2007. LNCS (LNAI), vol. 5049, pp. 58–75. Springer (2008)
7. Gudgin, M., Hadley, M., Rogers, T.: Web services addressing 1.0 - core. World Wide Web Consortium, Recommendation REC-ws-addr-core-20060509 (May 2006)
8. Hohpe, G., Woolf, B.: Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional (October 2003)
9. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: Monitoring, prediction and prevention of sla violations in composite services. In: 2010 IEEE International Conference on Web Services (2010)
10. Marconi, A., Pistore, M., Sirbu, A., Eberle, H., Leymann, F., Unger, T.: Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation. In: Service-Oriented Computing. LNCS, vol. 5900, pp. 445–454. Springer Berlin / Heidelberg (2009)
11. Noël, V., Arcangeli, J.P., Gleizes, M.P.: Between Design and Implementation of Multi-Agent Systems: A Component-Based Two-Step Process. In: EUMAS'10 (2010)
12. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson, Prentice Hall (2008)
13. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. Computer 40(11), 38–45 (2007)
14. Pereira, J.V.: The new supply chain's frontier: Information management. International Journal of Information Management 29(5), 372 – 379 (2009)
15. Strunk, A., Braun, I., Reichert, S., Schill, A.: Supporting Rebinding in BPEL. In: IEEE International Conference on Web Services. pp. 864 –871 (july 2009)
16. Weyns, D., Haesevoets, R., Helleboogh, A., Holvoet, T.: The MACODO Organization Model for Context-Driven Dynamic Agent Organizations. ACM Trans. Auton. Adapt. Syst. 6(4) (2010)