Combining Organisational and Coordination Theory with Model Driven Approaches to Develop Dynamic, Flexible, Distributed Business Systems

Javier Vázquez-Salceda[1], Luigi Ceccaroni[2], Frank Dignum[3], Wamberto Vasconcelos[4], Julian Padget[5], Siobhan Clarke[6], Paul Sergeant[7], and Kees Nieuwenhuis[8]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain
[2] Tech Media Telecom Factory SL, Barcelona, Spain
[3] Universiteit Utrecht, The Netherlands
[4] University of Aberdeen, United Kingdom
[5] University of Bath, United Kingdom
[6] Trinity College Dublin, Ireland
[7] Calico Jack Ltd, United Kingdom
[8] Thales Nederland B.V., Netherlands

Abstract. Enterprise systems are increasingly behaving as nodes in a digital, dynamic ecosystem. On the one hand, this new situation requires flexible, spontaneous and opportunistic collaboration activitiesto be identified and established among business parties. On the other hand, it also requires engineering approaches able to integrate new functionalities and behaviours into running systems and active, distributed, interdependent processes. In this paper we present a multi-level architecture, combining organisational and coordination theories with model driven development, for the implementation, deployment and management of dynamic, flexible and robust service-oriented business applications.

1 Introduction

A new generation of networked business applications based on the notion of software services that can be dynamically deployed, composed and adapted will make it possible to create radically new types of applications. These applications will be able to communicate, flexibly reconfigure at runtime, adapt to their environment and dynamically combine sets of simple, specialised, independent services into more comprehensive business services. This will require profound changes in the way software systems are designed, deployed and managed, replacing existing "design in isolation" engineering with new approaches able to integrate new functionalities and behaviours into running systems and active, distributed, interdependent processes. Technical progress in the area of service-oriented architectures (SOAs) in recent years has been impressive, with new models, tools and standards emerging to cover a wide range of core and related functions. The main areas of progress include:

- interoperability (SOAP [15], WSDL [18] and OGSI [12]);
- discovery and management (UDDI [11] and WS-Management [3]);

– orchestration and choreography (WS-BPEL [5], XPDL [19], ebXML [10] and WS-CDL [17]); association of semantics with Web-services (OWL-S [16] and WSMO [20]).

Furthermore, advances have come from a variety of sources, including enterprise interoperability, Grid computing, software engineering, database and knowledge-base theory, artificial intelligence, object-oriented systems and logic. This rapid progress has, for the first time, raised the realistic possibility of deploying large numbers of services in intranets and extranets of companies and public organisations, as well as in the public Internet, in order to create communities of services that are always connected, frequently changing, open or semi-open, and form the baseline environment for software applications. However, this shift brings about not only potential benefits, but also serious challenges for how such systems and applications should be designed, managed and deployed. Existing approaches in some important areas (such as security, transactions and federation) tend to cover only technology issues such as, for example, how to secure a protocol or connect federated directories, without considering the paradigm change that occurs when large numbers of services are deployed and managed over time. In particular, existing approaches do not offer satisfactory answers to these questions:

– How to manage workflows in non-trivial environments, where not all services are owned by the same organisation? Since we cannot assume that all parties are either benevolent or that they will deliver results unless explicit obligations are defined and enforced, should workflows be agreed upon by all parties before they can be executed?
– How to align the configurations and settings needed by a service to operate with those of the operational environment?
– How service execution is affected by issues of trust, rights, obligations and prohibitions?
– What if critical applications simply cease to function if services provisioned from third parties disappear or malfunction?
– How to deal with knowledge representation, when connecting or binding together two or more actual entities or services using different ontologies?

These issues point to the need for a“social layer” as part of the service interaction context. From an engineering perspective, new approaches are needed that take an holistic view of service environments, and take into account not only the properties of individual applications, but also the objectives, structure and dynamics of the system as a whole. In recent years, research in fields as diverse as social science, management science, economics, biology, distributed systems and multi-agent systems, analysed, modelled and explained a wide range of social phenomena often seen in human and animal societies and tried to apply those results to computational systems. In particular, techniques have been developed, that:

– Make it possible to characterise and model the organisational structures commonly used by humans to organise themselves in societies with particular needs;
– Capture coordination patterns that are often used between humans to solve common problems (e.g., to sell goods or achieve specific goals);

– Characterise autonomous actors in an environment and model their potential, rational behaviour (in order to predict, for example, how individuals will act in the context of a given set of "rules").

The rest of this paper, which aims to describe how the ALIVE[1] architecture deals with the issues mentioned above, is organised as follows. Section 2 describes a detailed view of the architecture, while in section 3 some use cases are presented. The final section gives the conclusion and outlines directions for future research.

2 Proposed Architecture

The proposed ALIVE architecture combines *model driven development* (MDD) [13] with coordination and organisational mechanisms, providing support for *live* (that is, highly dynamic) and *open* systems of services. ALIVE's approach extends current trends in engineering by defining three levels in the design and management of distributed systems: the Service Level, the Coordination Level and the Organisation Level, illustrated in Fig. 1, and explained below.

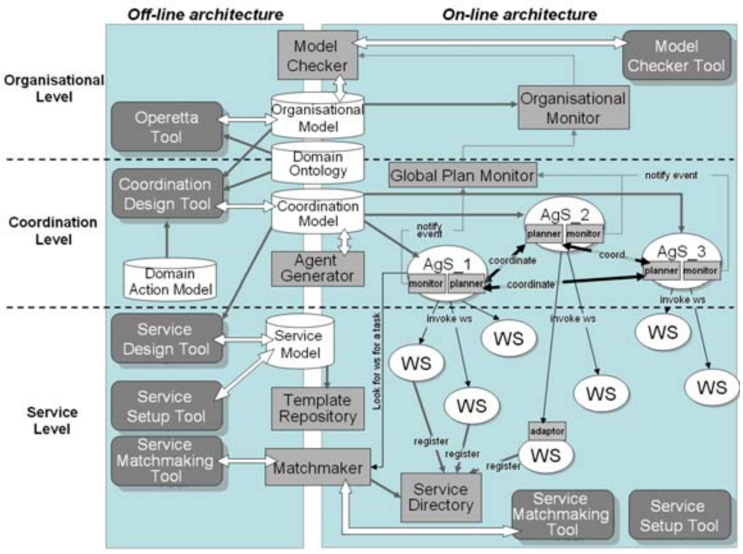

Fig. 1. ALIVE Multi-Level Architecture

The *Service Level* extends existing service models, in order to make components aware of their social context and of the rules of engagement with other components and services, by making use of semantic Web technologies. This "semantification" is particularly useful when highly dynamic and frequently changing services (the WSs in

[1] The ALIVE research project (`http://www.ist-alive.eu/`) is funded by the European Commission within the 7th Framework Programme for RTD (contract FP7 215890).

Fig. 1) are present in the system, as the meta-information in each service description (stored in a *service directory*) eases such processes as finding substitute services if a given service does not respond to invocations.

The *Coordination Level* provides the means to specify, at a high level, the patterns of interaction among services, using a variety of powerful coordination techniques from recent research in the area [4,9], based on agent technology. For these a special kind of services are introduced at this level: *agentified services*. These services are organisational-aware, that is, they are aware of the system objectives and manage task allocation and workflow generation and agreement. Also, at the coordination level agreed workflows can be adapted *while* the system is running – this is very useful when the system has to react to failures or exceptions (e.g., failing payment or booking systems).

The *Organisational Level* provides a social context for the Coordination and the Service levels, specifying the organisational rules that govern interaction and using recent developments in organisational dynamics [14] to allow the structural adaptation of systems over time. This is important when frequent changes of rules and restrictions are expected.

The ALIVE architecture can be seen as a service-oriented middleware supporting the combination, reorganisation and adaptation of services at both design- and runtime. These activities follow organisational patterns and adopt coordination techniques. Furthermore, the MDD paradigm integrated in the architecture offers significant assistance to system developers, as it provides semi-automated transformations between models of the three levels described above, as well as the capacity for multiple target platforms and representation languages. In the remaining part of this section, a more detailed description of the architecture is provided. Complementary theoretical aspects are described in [1] and related methodological aspects are described in [2].

2.1 Off-Line Support

The main objective of the off-line part of ALIVE's architecture is to give support not only to the design phase of distributed systems based on the ALIVE methodology (see [2]), but also to support the continuous maintenance of deployed systems during their lifetime.

The *OperettA Tool* is an organisational modelling tool. Its function is to create and manage the *organisational model* of a given distributed system. It is possible to connect OperettA to a domain ontology, use the entities of the ontology in the organisational model, and ensure the consistency of the entities across different models of the organisational level and a consistent connection between organisational model and coordination model. The functionality of this tool is extended by the Dynamic Organisational Modelling plug-in, for those applications which need extra methods and components to handle dynamic organisational change. One important aspect of the organisational models built by this tool is that they abstract away from the low-level details of the services that may be invoked. The designer is able to design the whole organisational level of a given distributed system through abstract concepts such as *objectives*, *roles*, *obligations*, *violations*, *sanctions* and high-level interaction diagrams that only identify critical states (called *landmarks*). The interaction with some end users (see section 3) has shown that this abstract specification is easy to understand to non-software specialists, easing the validation of designs by end users.

The *Model Checker* is used to verify some properties of organisational models. E.g., if a role of manager depends on a role of salesman to get information then an interaction scene is defined where this information exchange is modelled. This component not only is used at design time but can also be used on-line during the execution of the system. The Model Checker Tool thus complements the OperettA Tool, and is linked directly to it through a button in the OperettA interface.

The *Coordination Design Tool* is used by designers and system administrators to create and manage the coordination model of a given application. The tool assists the user in the definition of actors and tasks, the generation of the agents that will perform the actual coordination tasks and the inspection of predefined and generated plans. It also supports importing and inspecting the task descriptions defined in an application *domain action model*. A distinctive trait of the coordination models created by this tool (in comparison with other orchestration and choreography technologies [5,19,10,17]) is that the coordination models also abstract away from the low-level details of the services that may be invoked. The designer is able to design the whole coordination level of a distributed system by means of *actors*, *tasks*, *plans* and *plan coordination mechanisms*. The Agentified Services (see section 2.2) are the ones that connect, at execution time, the abstract tasks with the actual services that are invoked. Apart of the dynamism this solution brings at execution time, this also allows end users to inspect and better comprehend coordination models.

The *Service Design Tool* is used by designers and system administrators to generate or inspect service descriptions, edit service templates and register them in the *service directory*. It also connects with the *Service Matchmaking Tool* (a human interface to the matchmaker component), allowing designers and system administrators to search for services matching a given task description or implementing a given service template and registering it in the *service directory*.

Finally the *Service Setup Tool* allows designers and system administrators to check and modify the setup of the running environment, including the URIs of different resources, facilitating components, pre-defined services and service containers.

2.2 On-Line Support

The main objective of the on-line part of ALIVE's architecture is to give support to the deployment and execution of distributed systems. This part also covers on-line monitoring mechanisms to detect failures or deviations from expected functionality, and provides the mechanisms to correct, recover or adapt to them.

The *Service Directory* is one of the main components of the architecture. It is a repository for service interface descriptions, service process models and service profiles. It thus supports several query mechanisms for the discovery of specific services based on their syntactic and/or semantic descriptions. The service directory can be used by any component or service, and its main clients are the matchmaker components. Depending on the requirements and size of the distributed system, one or many service directories may be deployed. *Matchmakers* are an important component for service recognition. They are responsible for the discovery of appropriate services that fulfil the requirements of a given task. Depending on the level of abstraction of the task to be fulfilled, matchmakers may query a service directory directly or by the use of service templates. The *Service Templates* are intermediary descriptions linking higher-level goals or tasks (as specified in the Coordination Model) with specific service interactions. A template includes a parameterised process model for a class of services in terms of pre- and post-conditions. If the described process is of some complexity, such description may include an abstract workflow fragment indicating required sub-steps in the process. Parameters in a template are specified as abstract types linked to the variables or parameters in the process model. The parameters are dynamically bound at execution time into concrete ontology instances and/or concrete service process models when the template is selected by a matchmaker. Service template definitions are stored in a *Template Repository*.

Agentified services are organisation-aware services able to coordinate with others according to a given organisation and coordination model. They are able to:

- Incorporate the description of an organisation role, including its objectives, rights, obligations and prohibitions;
- Build, at run-time, local plans to fulfill the role's objectives;
- Coordinate its activities with other agentified services, thus building a partial global plan [8].

Agentified services can interact with normal Web services by means of standard SOAP and REST interfaces. Furthermore Agentified services communicate coordination-related issues to other agentified services using protocol-based conversations expressed in a coordination language (based on GPGP) implemented over SOAP. The exchanged plans are abstract workflows possibly with tasks referring to abstract services rather than to concrete ones (e.g., "map service" instead of "Google Maps"). When a plan is agreed upon, an agentified service will look (via the matchmaker component) for services that can fulfil the abstract tasks, binding them together.

External Services (i.e., existing, third-party services that have not been designed following the ALIVE framework) can be invoked at execution time according to their service description. Usually, external services are not consumed directly; instead, this is done via service adaptors. *Service adaptors* allow external services to be utilised for suitable organisational tasks. Typical examples of adaptation are type translation services to adapt a service interface to the entities and data types used by a given organisation.

One or several *Monitoring Components* are able to aggregate and analyse events related to the execution of services, the fulfilment of coordination plans and the achievements of role and/or organisational objectives. During the on-line execution, events are generated by the components (viz., the agentified services), whenever deviations, exceptions or failures are detected that cannot be handled by the agentified service itself or the existing coordination plan in place. In such situations the current organisational model is evaluated and then either (a) the objectives affected by the detected issue may be re-evaluated (their priority may be lowered or they may even be dropped completely), or (b) more significant changes in the organisation model may be required (for instance, changing the rights of a role). In case (a) the agent's coordination modules will create a new plan based on the updated organisational objectives. In case (b) the updated model is sent to the Agent Generator component to (re)generate the agentified services that populate the system. Depending on the set-up preferences of the administrator, the monitoring component may be a separate component used by several agentified services or may be a federation of several components inside the agentified services themselves.

Finally, there are graphical tools to support system administrators in the management of a distributed system. The *Monitor Tool* allows the administrator to inspect the status of a system's execution, getting information from the different components in the on-line architecture (especially from the monitoring components). In this way the administrator can keep track of the events generated at execution time and inspect how the system handles them. The *Service Set-up Tool* can be used by system administrators to check and modify the setup of the running environment. Finally the *Service Matchmaking Tool* (also used off-line) allows administrators to search manually for services that match a given task description (by using the matchmaker component) when the system is not able to cope with a particular issue or the administrator wants to change manually an automatically selected service for another that is considered more suitable for reasons not modelled within the ALIVE framework.

3 Application Scenarios

The ALIVE multi-level architecture is especially useful for scenarios where changes can occur at either abstract or concrete levels, and where services are expected to be continuously changing, with new services entering the system and existing services leaving it at run-time. For example, when there is a significant change at a high level (e.g., a change in the organisational structure), the service orchestration at lower levels can be automatically reorganised. Another example is the automatic adaptation of higher levels when lower ones suffer significant changes (e.g., the continuous failure in some low-level services). The proposed architecture is currently being applied to three real-life systems:

– *Dynamic orchestration of distributed services on interactive community displays* – The concept of *interactive community displays* (ICDs) is to build a virtual space within which people in urban communities can interact and share knowledge, experiences, and mutual interests. ICDs integrate urban information (in real time) and create public spaces in the Internet for people living in or visiting a city. ICDs are being developed all over the world in the context of digital cities [6]. An important question concerns why urban information spaces attract people given this era of globalisation. The Internet has triggered global businesses, and at the same time enables us to create rich information spaces for everyday life. While the Internet makes research and businesses global, life is inherently local. Business requires homogeneity to allow global competition, while life is heterogeneous, reflecting the different cultural backgrounds [7]. TMT Factory, one of the partners of the ALIVE project, is creating a personalised recommendation tool which provides city services to residents and tourists through ICDs (see Fig. 2). These ICDs, installed in urban environments, display tourist information and other services, dynamically personalised according to user preferences, geographical location and local laws. The ALIVE architecture provides support for the organisational modelling of the

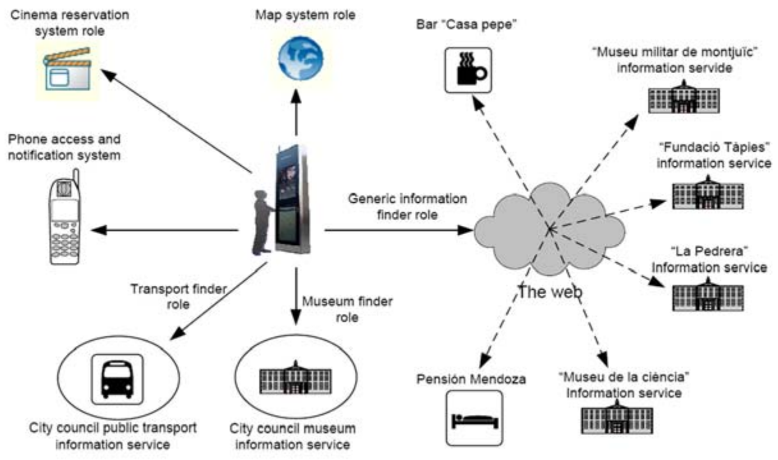
Cinema reservation system role
Map system role
Bar "Casa pepe"
"Museu militar de montjuïc" information servide
Phone access and notification system
"Fundació Tàpies" information service
Generic information finder role
"La Pedrera" Information service
Transport finder role
Museum finder role
The web
Pensión Mendoza
"Museu de la ciència" Information service
City council public transport information service
City council museum information service

Fig. 2. Example of the ICD interaction with different organisational services, modelled by decomposing the system functionalities in terms of clearly-defined roles

different entities that offer services to users and dynamically adapts service composition in case of service failure.

– *Dynamic crisis management*. Simulations of crisis scenarios have the potential to improve the organizational structures and the quality of response of the parties involved. As a crisis evolves, organizational structures must be systematically updated to reflect the changes in the numbers of personnel and the seriousness of the crisis. Simulation allows the effectiveness of these organizational changes to be evaluated. In the real world, simulations are enacted using active personnel. However, such simulations are expensive, both in terms of the cost of execution and the cost of the time required for the emergency service personnel involved. THALES B. V., another partner of the ALIVE project, is creating an emergency simulation tool for the Dutch emergency forces. This simulator effectively explores diverse crisis management scenarios when a natural disaster has escalating city- and nation-wide consequences, allowing policy makers to evaluate current and potential emergency policies and procedures. In particular The ALIVE architecture provides rich ways to describe all stakeholders in different emergency scenarios, their roles, responsibilities and dependencies and suggest the coordination strategies that can emerge according to a given set of policies.
– *Communication in entertainment domains*. Calico Jack Ltd., another partner, is developing an Entertainment Communication Router (ECR), a system that exploits the richness of online entertainment systems by coupling them with social networking resources. The ECR is designed to address the fragmentation of communication resulting from an increasing diversity of channels. This fragmentation makes it difficult to optimise the delivery of messages within contraints imposed by communication norms and a dynamic context of preferences, presence and availability. The system manages the roles, identities and social relations that users have in different fora (e.g. Facebook) to support rich forms of communication both between players

and with their wider social environment. In this case the ALIVE architecture provides support for high-level context definition and management. This includes the description of the information routing rules that reflect the complex, dynamic and sometimes conflicting contraints that emerge from the users' context.

4 Conclusions

The ALIVE framework aims to support the design and development of distributed systems suitable for highly dynamic environments, based on model-driven engineering, and three interconnected levels: service, coordination and organisation.

The crucial distinction of the ALIVE approach from existing ones is that it provides an organisational context (such as, for instance, objectives, structures and regulations) that can be used to select, compose and invoke services dynamically. ALIVE also provides a notion of organisational awareness to some components (such as the agentified services at the Coordination Level or the matchmaker component at the Service Level) that can direct system execution in order to achieve higher-level organisational objectives. One of the effects is that exceptions can be managed not only at the lower level (as in other service-oriented architectures) but at higher levels, looking for alternative ways to fulfil a task or even a full abstract workflow by agreeing upon a new plan of action. Furthermore, organisational and coordination models are defined at a level of abstraction that allows non-expert end-users to support better the design and the maintenance of the system.

The first version of the ALIVE tool suite is now under development and will become available through the project's Sourceforge site[2].

Acknowledgements

This research has been carried out within the FP7-215890 ALIVE project, funded by the European Commission. Javier Vázquez-Salceda is partially funded by the Ramón y Cajal program of the Spanish Ministry of Education and Science.

References

1. ALIVE project: ALIVE Theoretical Framework, http://www.ist-alive.eu/index.php?option=com_docman&task=doc_download&gid=3&Itemid=49 (retrieved 20090527)
2. ALIVE project: ALIVE Methodology Document, http://www.ist-alive.eu/index.php?option=com_docman&task=doc_download&gid=2&Itemid=49 (retrieved 20090527)
3. DMTF: Web Services Management, http://www.dmtf.org/standards/wsman/ (retrieved 20090527)
4. Ghijsen, M., Jansweijer, W., Wielinga, B.B.: Towards a Framework for Agent Coordination and Reorganization, AgentCoRe. In: Sichman, J.S., Padget, J., Ossowski, S., Noriega, P. (eds.) COIN 2007. LNCS (LNAI), vol. 4870, pp. 1–14. Springer, Heidelberg (2008)

[2] http://sourceforge.net/projects/ict-alive/

5. IBM: Business process execution language for web services version 1.1 (July 2003), `http://www.ibm.com/developerworks/library/specification/ws-bpel/` (retrieved 20090527)
6. Ishida, T. (ed.): Understanding Digital Cities: Cross-Cultural Perspectives. MIT Press, Cambridge (2000)
7. Ishida, T.: Digital City Kyoto. Communications of the ACM 45(7), 76–81 (2002)
8. Lesser, V.: Evolution of the GPGP/TAEMS domain-independent coordination framework. In: AAMAS 2002: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1–2. ACM, New York (2002)
9. Matskin, M., et al.: Enabling Web Services Composition with Software Agents. In: Proc. of the Conference on Internet and Multimedia Systems, and Applications. Honolulu (2005)
10. OASIS: The Framework for eBusiness (April 2006), `http://www.ebxml.org/` (retrieved 20090527)
11. OASIS: Universal Description, Discovery and Integration (UDDI) version 3.0.2. OASIS (2002), `http://www.uddi.org/pubs/uddi_v3.htm` (retrieved 20090527)
12. OGSI Alliance: OGSI Specifications, `http://www.osgi.org/Specifications/HomePage` (retrieved 20090527)
13. OMG: Model Driven Architecture, `http://www.omg.org/mda/` (retrieved 20090527)
14. van der Vecht, B., Dignum, F., Jules, J., Meyer, C., Dignum, V.: Organizations and Autonomous Agents: Bottom-up Dynamics of Coordination Mechanisms. In: 5th Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems. Estoril (2008)
15. W3C: Simple Object Access Protocol (SOAP) 1.1. W3C Note, `http://www.w3.org/TR/2000/NOTE-SOAP-20000508` (retrieved 20090527)
16. W3C: OWL-S - Semantic Markup for Web Services (2004), `http://www.w3.org/Submission/OWL-S/` (retrieved 20090527)
17. W3C: Web Service Choreography Description Language (WS-CDL) (2005), `http://www.w3.org/TR/ws-cdl-10/` (retrieved 20090527)
18. W3C: Web Service Description Language, WSDL (2001), `http://www.w3.org/TR/wsdl` (retrieved 20090527)
19. WfMC. XML Process Definition Language (XPDL). Document Number WFMC-TC-1025: Version 1.14 Document Status - Final (2005)
20. WSMO working group: Web Service Modeling Ontology. ESSI cluster, `http://www.wsmo.org/` (retrieved 20090527)