ABSTRACT
Web services are increasingly behaving as nodes in a digital, dynamic ecosystem. On the one hand, this new situation requires flexible, spontaneous and opportunistic collaboration activities to 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 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 [19], WSDL [22] and OGSI [15]);
- discovery and management (UDDI [14] and WS-Management [4]);
- orchestration and choreography (WS-BPEL [7], XPDL [23], ebXML [13] and WS-CDL [21]);
- association of semantics with Web-services (OWL-S [20] and WSMO [24]).

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 the following 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 is service execution 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 has 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 describes how we try to deal with the issues mentioned above using the ALIVE\(^1\) architecture. In the ALIVE architecture, the service layer is extended with a coordination and an organisation layer. Each of the layers deals with some specific issues mentioned above avoiding a monolithic solution.

This paper is organised as follows. Section 2 describes a detailed view of the architecture. The three subsequent sections provide more details for each of the three layers - namely organisation, coordination and service - and their connections. In section 6 some use cases are presented that illustrate the usefulness of our approach. The final section gives the conclusion and outlines directions for future research.

2. PROPOSED ARCHITECTURE

The proposed ALIVE architecture combines model driven development (MDD) [16] 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.

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 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. In current service oriented systems the coordination between services is specified using choreography and orchestration languages. However, these tools assume that the coordination patterns are designed offline and the whole workflow is predetermined. This works fine in applications where the system is designed by people beforehand and no adjustments to the workflow are needed during execution. However, problems arise when workflows have to be changed due to changing circumstances or unavailability of services. In ALIVE we use a variety of powerful coordination techniques from recent research in the area [5, 12], based on agent technology. For these techniques a special kind of services are introduced at this level: *agentified services*. These services are organisation-aware, that is, they are aware of the system objectives and manage task allocation and workflow generation and agreement. Because the agents are aware of the purpose of a workflow they have handles to change the workflow within the limits that it still fulfills the same purpose. 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).

On top of the coordination level we define a further level, the **Organisational Level**, which provides a social context for the Coordination and the Service levels. The organisation level specifies the organisational rules that govern interaction. It is assumed that the organisational level is the most stable description of the system in the long term. So, many facets are described only in an abstract way so as to allow for more than one implementation and/or changes of implementation (different services, workflows, interaction patterns) over time. The most striking example of this abstraction is that, on this level, the system is defined on the basis of goals and results and we abstract away from the specific actions used to accomplish these results and goals. In some sense, the organisation level defines the playground within which boundaries the coordination and service levels can move. It thus secures some overall stability of the system and guarantees global results that are required of the system. Although the organisational level is quite static we also use recent developments in organisational dynamics [18] to allow the structural adaptation of the global system over time. This is important when frequent changes of rules and restrictions are expected, but should be used as little as possible.

The ALIVE architecture can be seen as a service-oriented middleware supporting the combination, reorganisation and adaptation of services at both design- and run-time. 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 at the three levels described above, as well as the capacity for multiple target platforms and representation languages. 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 Setup 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.

In the next three sections, a more detailed description of the ar-

\(^1\)The ALIVE research project ([http://www.ist-alive.eu/](http://www.ist-alive.eu/)) is funded by the European Commission within the 7th Framework Programme for RTD (contract FP7 215890).
chitecture is provided. Complementary theoretical aspects are described in [1] and related methodological aspects are described in [2].

3. THE ORGANISATIONAL LEVEL

As can be seen from Fig. 1 we use a special tool (the Operetta Tool) to model the organisational level of the system. The tool is based on the framework for Agent Organizations (OperA) [3], which is a general framework to specify the organizational context of multi-agent systems. Within OperA an organisation is viewed as a set of entities (the stakeholders) and their interactions, which are regulated by mechanisms of social order. 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 interaction with some end users (see section 6) has shown that this abstract specification is easy to understand to non-software specialists, easing the validation of designs by end users. 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.

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 organisational model designed on this level is specified in terms of four structures:

- The social structure specifies objectives of the society, its roles and what kind of model governs coordination.
- The interaction structure describes interaction moments, as scene scripts, representing a society task that requires the coordinated action of several roles, and gives a partial ordering of scene scripts, which specify the intended interactions between roles.
- The normative structure expresses organisational norms and regulations related to roles.
- The communicative structure specifies the ontologies for description of domain concepts and communication illocutions.

We will briefly explain what types of things are modeled by each of these structures.

3.1 Social structure

The social structure of an organization describes the objectives of the society, its roles and what kind of model governs coordination. The central concept in the social structure description is the role. Roles are abstractions providing a means of generically addressing stereotypical (expected) behaviours (i.e. whoever takes up a role is expected to behave in a particular way) [3]. Roles identify activities necessary to achieve organisational objectives and enable abstraction from the specific actors and/or services that will eventually perform them. That is, roles specify the expectations of the organisation with respect to the actor’s activity in the organisation. Roles are described in terms of objectives (what an actor of the role is expected to achieve) and norms (how an actor expected to
of activity. We distinguish regulative and constitutive norms of organisation, related to the roles they play, or to a particular area of the interaction structures. In our framework norms define the scene objectives. Usually all role objectives should be related to scene results.

3.3 Normative structure

The normative structure is the part of the Organisational Model that represents the collection of norms and rights related to the social and interaction structures. In our framework norms define the obligations, permissions and prohibitions of the actors in the organisation, related to the roles they play, or to a particular area of activity. We distinguish regulative and constitutive norms [6]. Regulative norms regulate agents behaviour enacting roles. They are expressed by means of deontic declarative expressions. Constitutive norms regulate the creation of institutional facts. The constitutive norms will be part of the ontological specification of the system in the form of "counts-as" statements. E.g. sending credit card number, security number and name counts as payment. So, the constitutive norms indicate which concrete actions and facts are related to the organisational concepts. The regulative norms are spread over role norms, scene norms, and transition norms:

- Role norms specify the rules of behaviour for actors performing that role, irrespective of the interaction scene.
- Scene norms describe the expected behaviour of actors in a scene.
- Transition norms: Impose additional limitations to actors attempting to follow a transition between two scenes. These are typically used if a result in one scene limits the possibilities of accessible scenes afterwards (e.g., buying something in the auction scene creates an obligation to pass the payment scene afterwards).

Role rights (also role-rights or role capabilities) indicate the capabilities that actors of the role receive when enacting the role. These are capabilities that an agent usually does not possess but which are inherent to the role. E.g. as a customer of a bank I can use the Internet banking facilities of that bank.

3.4 Communicative structure

In the communicative structure the communication primitives are described. I.e. both the message types that can be used as well as an ontology that defines the concepts used in the messages and the rest of the specification. Practically, 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 the different structures of the organisational level and a consistent connection between organisational model and coordination model.

The message types that can be used at the organisation level indicate what type of protocols can (or should) be used to implement interactions. If only "delegate" or "order" messages and "accept" and "reject" messages are specified the protocols will be very simple and direct, but also limited in functionality. If "requests" and "answer" are allowed the door is opened to negotiations over tasks, which allows for far more flexibility.

4. THE COORDINATION LEVEL

The organisational model of the system is meant to provide a kind of background model for the rest of the system that indicates the boundaries and norms to which the system should adhere. The idea is thus that this organisational model will hardly ever change in the lifetime of the system. In contrast, the coordination and service level parts of the system are supposed to adapt to the current situation. Therefore they have an off-line, design component and an on-line component that is used while the system is running.

The Coordination Design Tool is used by designers and system administrators to create and manage the coordination model of a given application. The coordination level connects the high level, abstract objectives of the organisation to workflows that invoke services that can actually achieve those objectives. To achieve this connection the level consists of two main components, the agents and the workflows (or plans). 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. We use agents at this level because we do not only want to create static workflows that will achieve the organisational objectives, but we want to be able to create these workflows on the fly from existing plan libraries and also be able to adjust them when circumstances require so. In order to do this we need agents that can fulfill organisational roles and thus "know" about the organisational objectives for that role and the boundaries within which that role has to act. Using knowledge about the purpose of a workflow, planning knowledge and rights and duties attached to the role the agent can reason and negotiate with other agents which is the best course of action to reach the objectives of the organisation. We will now briefly describe the agent component and the
workflow component of the coordination level. After that we will discuss the monitoring tool and its role in adjusting the system to changing circumstances.

4.1 Agents for coordination

As can be seen from Fig. 1 the agents at the coordination level have three main tasks: planning, enacting and monitoring. Part of the responsibilities of the agents come from the role description at the organisational level and ontological knowledge. The agents are supposed to play a particular role in the system and thus the objectives of the role, the interactions it should participate in and its rights and duties are used as the basis to define the goals (from the objectives and landmarks), plans and beliefs about the world. Although the organisational level describes what the agent should achieve (playing a particular role) it does not indicate how this should be done. So, the planning component is designed at the co-ordination level. Of course, agents should be endowed with enough capabilities to reach their objectives.

The actions of the agents can still be more abstract than the actual services that are used on the service level, but they also can directly use available services. Therefore we see agents as agentified services that are organisation-aware and 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 [11].

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.

The matchmaker agent is a special agent that forms a gateway for the agents to the service level matchmaker component. Using a matchmaker agent avoids each agent having to incorporate the protocols to communicate with the service level matchmaker. They can now communicate with the matchmaker agent using requests for abstract services using the same language and protocols they use as for communicating with the other agentified services. The matchmaker agent will translate the requests for abstract services into service level queries and give answers back to the agents.

Finally, the agents at the coordination level also monitor the execution of the services (workflows) they invoke. One or several Monitoring Components will then 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.

4.2 Workflows

As said before, the coordination level also supports the definition, composition and import of actions (descriptions) to be used by the agents. A distinctive trait of the coordination models created by this tool (in comparison with other orchestration and choreography technologies [7, 23, 13, 21]) is that the coordination models also abstract away from the low-level details of the services that may be invoked. Actions are specified using an identifier (name) and their pre- and postconditions. This specification allows the use of (simple) planning systems to create workflows from actions to achieve certain goals. The designer is thus able to design the whole coordination level of a distributed system by means of actors, tasks, plans and plan coordination mechanisms. The Agentified Services 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.

5. SERVICE LEVEL

The service layer connects existing, real-world services to the coordination and organisational layers of an ALIVE organisation by mapping concepts, functionality and the rules of engagement which are already in existence to those relating to the organization being synthesized. The service layer is concerned with the description of services, components for selecting appropriate services for a given task and the execution and monitoring of services associated with a given organisation. We will describe the service level in three parts: the services, the matchmaker and the execution and monitoring of the services.

5.1 Services

Of course, the central entities on this level are the services. 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. In order to discover or invoke a particular service we use externalised descriptions of the services properties and its semantics. These descriptions provide the necessary information about service interfaces and service operations and may be functional (relating to the actual operation and semantics of the invocation of service operations) or non-functional (describing properties of a service, operation or interface which do not relate to the operational properties of that service). Functional descriptions are given for each operation of a given service and are typically broken down into Inputs, Outputs, Preconditions and Effects. Inputs are data which must be passed into an operation, descriptions may include a syntactic model of this data (e.g. XML schema fragments) as well as semantic descriptions of the meaning of this data (e.g. OWL Classes for particular data types). Outputs are data which will be passed out of a service operation upon its successful completion. As with inputs these may include syntactic descriptions of the data being passed or semantic descriptions of its meaning. Preconditions are descriptions of the state of affairs of
the world which must be satisfied at the point of invocation in order for a particular service operation to execute successfully. Preconditions may describe restrictions on the types of inputs accepted by the service operation or external properties (such as constraints on the state of the service itself) which must be held to be true. Finally, Effects are descriptions of the new state of affairs of the world which will be satisfied upon the successful invocation of a given service operation. As with preconditions these may relate to the outputs of the service or to internal state of the service (or the world). Effects may be conditional on the inputs to the service such that invoking the service with one type of input might yield one effect while invoking it with another would yield a different effect.

These descriptions may be incorporated into the services themselves, or may be provided externally and may already be defined for existing services or may need to be defined separately in order to incorporate existing services into an ALIVE organisation. Typically Service descriptions are broken down into "syntactic" descriptions which describe the interfaces, operations, operation type parameters and interaction patterns of particular services, and "semantic" descriptions which extend these underlying descriptions with higher-level information describing the semantics of operations and their parameters in such a way that services may be invoked by (semi-)automatically by intelligent agents or components.

Service operation descriptions are collected into a single service interface description which defines the operations available for a single service interface. In addition to defining the service operations, service interface descriptions also define how those operations are bound to particular protocols, (e.g. SOAP) and in some cases include the service endpoint address of the service interface. Service interface descriptions are represented using the Web Services Description Language (WSDL). Each service will have exactly one service interface description. In top of the service description the service process model gives a high-level (semantically grounded) interpretation of a given call of a service interface description, with each parameter (input or output) corresponding to an ontological type in OWL (as opposed to a simple XML type in WSDL).

In its turn the service process model is part of the service profile. The service profile describes the properties of a service which may be used during service discovery in the context of semantic web services. In the context of OWL-S a service profile includes the simple process model for a given service operation (top-level inputs, outputs preconditions and effects (IOPE)), a taxonomic description of the service which allows services to be selected by category, and other non-functional properties. Once a given service is selected only the process model and service grounding are required to invoke the service. Each service which may be used within the ALIVE framework will have an associated service profile. For services which do not have a pre-existing service description it contains only the process model which can be inferred from an existing service interface description.

When a given task is to be executed a process model will be selected using the matchmaking process and a service grounding will be used to substitute appropriate input parameters into the actual service invocation and to extract the relevant output parameters from any return values.

As indicated before, services can also be External Services. That is, existing, third-party services that have not been designed following the ALIVE methodology. These external services can also be invoked at execution time according to their service description. Usually, however, external services are not consumed directly; instead, this is done via service adaptors. Service adapters 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.

5.2 Matchmaking

In order for an ALIVE organisation to be constructed and enacted, appropriate services must be selected for each functional unit of work within the organisation. The process of discovering such services is generally referred to as service discovery and matchmaking and may be conducted automatically based on published service descriptions or with human assistance using the Service Matchmaking Tool (a human interface to the matcher component), allowing designers and system administrators to search for services matching a given task description or satisfying a given service template and registering it in the service directory.

The Matchmaker is 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, a matchmaker may query a service directory directly or by the use of service templates, which 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.

The Service Directory is a repository for service interface descriptions, service process models and service profiles. It supports several query mechanisms for the discovery of specific services based on their syntactic and/or semantic descriptions. Service directories are used as part of the service discovery process in the service layer and during the construction of abstract workflows in the coordination layer. When a workflow is being constructed the coordination layer may query service directories to determine if a given service description can be satisfied. When a specific service is being bound to a give service process model in the execution phase a service execution framework will search and query one or more service directories in order to list potential matches for a given service description which will in turn be filtered and ordered by one or more service matchmakers. Depending on the requirements and size of the distributed system, one or many service directories may be deployed.

The matchmaking process operates in two phases: (i) identification of a candidate set of services, based on IOPE, using a subsumes type matcher [10], then (ii) computing a preference order based on criteria supplied by the client, which is in effect a partial-order over the subsumes requirement, through which we can construct either exact or plugin matches since these are both subsets of subsumes.

We have chosen to adopt the currently leading technology of OWL-S as the basis for service description and matching. An OWL-S matcher is a service registry that supports service querying based on the service semantics, captured in corresponding OWL-S descriptions. It can be thought of as the analog of UDDI registries for web services, with the addition of OWL annotations to support more precise service specifications through the use of ontologies.

Here we provide a broad description of how the matchmaker handles a matching query. At the very least, the query must de-
scribe the signature of the desired service, that is the classes of the input and output parameters. Essentially such a signature expresses two facts: (a) That the input parameter classes represent what the client promises to be able to supply to a matched service in order to invoke it. Thus, with the input parameter classes the client claims that it can fulfill any of these classes, that is it has, for any of these classes, an (indirect) instance of it, and (b) That the output parameter classes represent what output is required (and can be handled by) the client. In other words, the user wants the service to produce, for any of these classes, some value that belongs to it.

The matchmaker then computes and returns the maximal set of services such that: (a) Their input parameters can be fulfilled by the client, assuming only its advertised capabilities. That is, if the client would be able to supply arguments to the exact service signature it requested, then it can provably invoke the matched service. Note that a service may have more relaxed constraints on the inputs it requires, for example it may declare inputs that are superclasses of what the client can supply, thus they can by definition be fulfilled by a value that would fulfill the subclass, or it may declare fewer inputs, and (b) Their outputs (assuming the service is not abnormally terminated or unavailable) are guaranteed to supply a value for any output classes the client requested. Note that the service outputs may be more specific (that is, subclasses) than the client requested, or it may produce more outputs, which could, of course, be ignored. The above matching criteria coincide with OWLS-MX subsumes matching level [10].

After identifying the set of candidate services, the matchmaker may further order them according to the client’s preferences, so that it finds the “best” results, according to the client’s perspective, on top and might, for example, be interested in “just the 5 best matches”, discarding the rest. The matching services may be augmented with further, arbitrary information to aid finding the best of them. For example, out-of-band information such as reliability or performance statistics may be associated with services. The matchmaker offers extremely flexible support for capturing and honouring client preferences. All that is needed is to define a partial order function for the matched services, according to which the matchmaker orders the results, starting from the best elements to the worst. This is much like sorting a list according to a total order, only more powerful. For example, the client may express preferences such as “My top preference is for services that specify input classes closest to the ones I requested; then I care equally about better scoring services by textual descriptions against my description, and finally services with good reliability”. The specification of preferences may be arbitrarily complicated. To cope with this, the matchmaker offers a small partial order algebra with composable operators, capable of expressing arbitrary preferences in a concise way, and which can be evaluated in order to find the best results according to it.

The objective is to create a PartialOrder[Match], which is then applied to the last function. We now summarise the key operators (given some type E):

1) create: [E] (dominates: (o1: E, o2: E) → boolean) → PartialOrder[E], which takes a binary function capable of comparing two objects and determining whether the first dominates the second

2) map: [E, T] (po: PartialOrder[T], mapper: (T → E)) → PartialOrder[E], whose purpose is to allow the construction of, say, a partial order over services by means of a metric function that maps a service to some measure. Thus, given some function Service → Integer, for some measure, we can construct a PartialOrder[Match] from a PartialOrder[Integer].

3) flatten: [E] (pos: Iterable[PartialOrder[E]], metaoader: PartialOrder [PartialOrder[E]]) → PartialOrder[E], whose purpose is to construct a single partial order of services from several partial orders, where the latter each captures the orderings of services according to a single client preference criterion while the result reflects their combination by client preference ordering, such as in the (natural language) example in the preceding paragraph.

4) topElements: [E](elements: Iterable[E], pos: PartialOrder[E]) → List[List[E]], whose purpose is to construct blocks of (mutually incomparable) results, where each succeeding block is dominated by some element of a preceding block.

Thus, we are able, with a high degree of generality able to identify services that do what is required and furthermore to order them according to arbitrarily complex (as far as this is expressible in OWL) criteria.

5.3 Service execution and monitoring

Each service has one or more service end points, which allow communication with service consumers. In the case that a provider of a service is internal to the organisation, each of these end points will be associated with a service execution environment and a corresponding agent which manages service invocations for that provider. For a given service interface, there may be more than one service provider. A service endpoint is a deployed service interface, which may be invoked by a service consumer, either within or outside the organisation. Each service end point has (at least) a Service location (URI), Service interface description (e.g. WSDL description) and optionally a semantic description (process model) of the services functionality and semantic description of the services non-functional properties (service profile). Service end points will be discovered (through the service discovery and matchmaking process) in order to achieve specific goals within the organisation. These endpoints will be invoked by agents within the organisation as part of the process of those agents achieving specific organisational goals.

A service execution framework may offer and/or consume web services. Each service execution framework may be responsible for exposing multiple services. Internally deployed services are expected to follow specific work flows, in cases where decisions must be made within these workflows the workflow execution environment may delegite this decision making to a software agent. Service execution frameworks are used within the service layer to deploy the services which will be offered by the organisation and manage the direct execution of the workflows associated with those services. Service execution frameworks receive plans from the coordination level as abstract workflows which will be deployed within the framework. As workflows are executed the service execution framework will pass service execution events to the service monitoring framework which account for significant events in the service execution process. When an abstract workflow is deployed within an execution framework, the framework will use one or more service matchmakers in order to determine appropriate services for specific abstract workflow tasks, in addition where matches require service adaptors those adaptors will be deployed within the execution framework.

The execution of the services is monitored by the service monitoring framework which provides a means for tools at both the service level and at higher levels to aggregate and analyse events relating to the deployment and execution of services. Each execution framework will be associated with a given monitoring framework which will receive messages relating to the underlying service execution. The monitoring framework will store these messages and
allow historic interactions to be queried.

6. 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 ALIVE architecture is currently being applied to three real-life applications:

- **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 [8]. 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 [9]. 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. 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 different entities that offer services to users and dynamically adapts service composition in case of service failure.

- **Communication in entertainment domains**. Calico Jack Ltd., another partner, is developing an Entertainment Communications Router, a system that exploits the richness of online entertainment systems by coupling them with online social networking resources. It manages the roles, identities and social relations that users have in different fora (e.g., Facebook, Second Life) to support rich forms of communication between players driven by their social relations. In this case the ALIVE architecture provides support for high-level context definition and management, including the description of the information routing rules according to the users’ social relations.

- **Dynamic crisis management**. 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. 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.

Due to space limitations we cannot describe all three scenarios in depth. We will just discuss some of the advantages that the ALIVE framework brings for the crisis management scenario. More information about the implementation of the scenario can be found in [17].

6.1 Crisis management using the ALIVE approach

In the crisis management scenario, we assume that a major storm is emerging above the North Sea after a long period of dry weather. Initially, the storm reaches the Port of Rotterdam, Europoort. One of Shell’s refineries is at risk: large quantities of petroleum products and associated dangerous chemicals may be dispersed at any moment. As the disaster progresses, several dikes are breached, flooding northern Rotterdam. Due to recent weather conditions several inland dikes outside of Rotterdam have dried out completely. Sudden inundation of water causes them to weaken and breach as well. This causes other suburbs of Rotterdam to flood as well.

The Netherlands has an extensive crisis management structure to respond to such disasters. This structure is based on the severity of the disaster, and allows local, regional and national authorities to take action where necessary. Most incidents are relatively minor, and do not require much cross-organizational interactions. However, when a more complex crisis does arise, this structure defines how first responders within a safety region, that is, the police, fire and medical services, municipal governments and other services, are to cooperate. Procedures for coordinated regional incident handling in the Netherlands, GRIP (Gecoördineerde Regionale IncidentenbestrijdingsProcedure) have been standardized by the Dutch Ministry of Internal Affairs. GRIP levels provide a mechanism to describe the severity of a problem. As a crisis grows and more responders become involved, more organization between these responders is required. At a regional and national level, the LOCC acts as a dedicated crisis management coordinator with links to all of the emergency response services/organizations.

The organizational model in ALIVE specifies the necessary components of the crisis management domain at a level of abstraction that allows the organization to remain stable over the different GRIP levels, but still providing enough meaningful information to help the simulation design. The social structure (i.e., roles and dependencies) of the crisis management organization is depicted in figure 2.

Each role in the organization has its own objectives and norms. Informally, the objectives of the roles shown in figure 2 are as follows. The Coordinator is responsible for solving the crisis, the Government wants to be informed at all times of the situation status, the Medics aim at evacuating and/or treating casualties, the Police must maintain order, the Firemen to save/rescue people and assets, the Army should provide logistic support and the Information Providers (which denotes a group of several roles with the same objectives) are responsible to provide reliable information.

The number of agents playing a role can change from one GRIP to another. For example, at GRIP-1, the Police role is typically enacted by a single agent (i.e., there is only one Police Officer involved in the crisis handling). At GRIP-2, however, the police role might be enacted by several units. These police units, while their overall interaction with the coordinator remains the same, now have an internal structure (a hierarchy in this case) and Police-specific coordination and control mechanisms (e.g., the order objective can
be divided in sub-objectives, for example, regulate traffic, patrol sector B5, etc., which are assigned to different units). These internal structures are unimportant for the crisis management organization, and can safely be abstracted from. Roles which can have such internal structure (at a higher GRIP) are denoted with an M; in essence, the role is enacted by a multi-agent system (or, sometimes even, by a multi-agent organization).

Note that some of the roles will not be enacted at the lower GRIP levels (e.g., the roles of Government or Army will not be relevant in GRIPs below GRIP-2 and GRIP-3, respectively).

A simple interaction structure is depicted in figure 3 defining the order in which the interactions between the roles in the organization should take place. Interactions are grouped into scenes, where a scene reflects a meaningful (on its own standing) subset of organizational interaction related to the achievement of a (set of) (sub)objectives. Each scene in the interaction structure can be further detailed as a landmark pattern. Figure 3(C) shows the landmark pattern for the **Handle Incident** scene. Landmarks denote the important states that should be reached in the achievement of the scene, and the landmark pattern imposes an ordering over these landmarks to denote the order in which these important states should be reached. The landmark pattern of the **Handle Incident** scene has 2 landmarks: **evaluate_severity**, and **evaluate_people** leading to a ending state where location is safe.

This organization structures are used at Coordination Level to determine concrete workflows and interaction protocols. An example of such protocol is depicted in figure 4.

As mentioned above, the landmark pattern provided for the scene **Handle Incident** at OL only specifies two important objectives (landmark states) to be achieved by role enacting agents during scene execution. Therefore, procedural information needs to be added at CL, as described in the workflow in figure 4. Different types of emergency situations will call for different interpretations of the OL descriptions.

The resulting workflow patterns contain abstract actions that are executed by services at the service level. As the services for this simulation are all internal, this part is pretty straightforward and left out for reasons of space.

7. **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 fulfill 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

**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. We are grateful to Dimitris Andreou for detailed discussions on the design and function of the matchmaking process and of course to all other members of the ALIVE consortium.

8. **REFERENCES**


Figure 3: Organizational Interaction Pattern.

Figure 4: Possible workflow model at CL to deal with the scene Handle Incident of Figure 3.


