Mutual Dynamic Adaptation of Models and Service Enactment in ALIVE*

Athanasios Staikopoulos[1], Sébastien Saudrais[1], Siobhán Clarke[1], Julian Padget[2], Owen Cliffe[2] and Marina De Vos[2]

[1] Trinity College Dublin, Computer Science, Ireland
{Athanasios.Staikopoulos, Sebastien.Saudrais, Siobhan.Clarke}@cs.tcd.ie
[2] University of Bath, Computer Science, UK
{jap, occ, mdv}@cs.bath.ac.uk

Abstract. In complex service-oriented systems, a number of layers of abstraction may be considered, in particular the models of the organisations involved, how interactions are coordinated and the services which are used and made available, are all relevant to the construction of complex service-oriented systems. As each of these layers is built upon another there is a clear need to provide a maintenance mechanism, capable of maintaining consistency across the concepts used in each layer. In addition, over time designs may change because of the introduction of new requirements and the availability and capabilities of services may change due to implementation modifications or service failures, leading to the need to consider a two-way adaptation, namely between the system design and its run-time. The contribution of this paper is the description of our (novel) mutual adaptation mechanism and, using an industry scenario based on the proposed ALIVE framework, its illustration in use of the kinds of adaptation.

Keywords: Model-driven architecture, web services, workflows, monitoring, adaptation.

1 Introduction

Today's software systems are becoming increasingly large and complicated. They are built upon many different technologies where a variety of abstraction layers are utilized, making it difficult for software engineering methodologies to support properly the various stages of their life-cycle, including design, implementation of artefacts and actual execution. Consequently, there is a clear need to develop maintenance and monitoring mechanisms allowing the dynamic adaptation, reconfiguration and self-management of such systems. It becomes increasingly clear that such mechanisms can provide a fundamental framework, where other more elaborate mechanisms can be established moving systems towards the vision of

* This work has been carried out in the framework of the FP7 project ALIVE IST-215890, which is funded by the European Community. The author(s) would like to acknowledge the contributions of his (their) colleagues from ALIVE Consortium (http://www.ist-alive.eu)

autonomic computing [1], where under certain circumstances a system may (re-) configure itself and adapt automatically to changing environments.

The work described in this paper is carried out in the context of the EU-funded ALIVE project [2, 3]. The premise behind the project is that current service-oriented architectures (SOAs) are typically incremental developments of existing Web service frameworks, making them fragile and inappropriate for long-term deployment in changing environments. Our proposed solution is to utilize the rich body of experience found in human organisations through the formalization of organisational theory and the coordination mechanisms that underpin the interactions between the entities. This provides us with a range of strategies that have been tried-and-tested in (human) social and economic contexts and that, with the provision of sufficient appropriate information about the state of the environment and the enactment of a workflow, can be applied to the dynamic adaptation of SOAs. A key element of our solution is the use of model-driven architectural descriptions of the SOA design – representing the organisational and coordination artefacts mentioned earlier – that admit formal adaptation and are thus able to capture and reflect changes in the deployed system.

In this paper we propose a bidirectional adaptation approach for maintaining design models with their run-time execution. The models visualising the service organisations and coordination as specified in ALIVE are used in a model-driven approach, while service enactment is a result of a model transformation process.

In SOA functional components are exposed as services, each of which is associated with an externalised description of the service's interface and functionality. These services are composed and linked in a loosely-coupled pattern in such a way that individual services may be replaced and re-used without modification. Current approaches to SOAs build on existing Web service (WS) technologies, such as SOAP, WSDL and BPEL to describe and execute service interactions. Given a set of services, process descriptions in the form of workflows may be constructed and executed using existing workflow interpreters, which take a given language such as BPEL and invoke services in accordance with the specified flow of control.

Model Driven Engineering (MDE) refers to the systematic use of models as primary artefacts for the specification and implementation of software systems. The Model Driven Development (MDD) methodology is based on the automatic creation of implementation artefacts from abstracted models via a predefined model transformation process. So far, model-driven approaches are primarily focused on the design, implementation and deployment stages of software development. However, MDD can similarly support the maintenance, requirements and testing phases. In those cases, MDD can be applied in the opposite direction, for the purpose of building or recovering high-level models from existing implementation artefacts to support round-trip engineering. Thus, it is possible to bridge the gap and provide consistency among design models and actual executions.

The remainder of the paper is organised as follows: Section 2 provides an overview of the research context. Section 3 presents our mutual adaptation approach for models and enactments. Section 4 highlights our approach with an example drawn from an ALIVE use-case scenario. Section 5, provides various discussion points and compares our approach with related work. Finally, section 6 outlines our conclusions and summarises the fundamental characteristics of our approach.

2 Dynamic Model Adaptation

Dynamic model adaptation refers to applying automated modifications on models often representing executing systems at run-time. Model-driven development often produces design artefacts that are lost during the execution and yet may be needed if the architect wants to change the actual execution when something goes wrong. The use of run-time models permits the complete or partial reuse of the current design models and their adaptation to the actual execution of systems. In particular [4] gives examples where run-time models can be useful in adaptation of systems. These examples are relevant to our two-way dynamic model adaptation mechanism.

The first case where run-time models are useful is the observation of the execution. The execution utilizes real code to perform the functions prescribed by the models. The use of a run-time model, based on the observation of the execution, allows for the creation of an abstract view of the execution, which in turn may be used by an adaptation module. The set of events which are observed in this process have to be generated from the design models.

The second case is the automatic adaptation of the system depending on the execution's observation. Patterns of adaption are usually defined by the architect during the design phase by taking account of some critical execution events. When a predefined set of events is triggered, the adaptation is performed on the run-time model and then changes are applied in the generated execution.

Finally, the third case is redesigning the actual execution using the run-time models. The architect, by looking at the run-time models, may decide to modify or add new functionalities to the system. These modifications are then transferred to the execution by production of run-time changes.

3 An Approach for Mutual Dynamic Adaptation

In this paper, we propose an approach for the dynamic adaptation of models and executables based on model transformations and the monitoring of the service enactment. The adaptation of models and executables is performed dynamically; both automatically and at run-time. Moreover, their dynamic adaptation is not based on the direct execution of models, so they are not compiled by model compilers and they do not run on specialised virtual machines - where executable models are monitored, but rather the adaptation is based on monitoring the enactment of native code that is the product of a model driven transformation process. Next, a monitoring mechanism monitors changes on service enactment and on design models by listening to specific significant events. Depending on the events generated the corresponding handling module is triggered to maintain/adapt the design models and generate the new enactment that will be loaded and executed from tools. The connectivity of external tools and the monitoring mechanism is maintained by the instrumentation framework. The approach is mutual, meaning that adaptations can be performed both a) from run-time execution to design models and b) from design models to run-time execution.

Another important characteristic that distinguishes our approach from others is that in our case model adaptations are applied both on structural models defining the organisation of Multi Agent Systems (MAS) [5] and behavioural models defining their coordination. Furthermore, adaptations are applied on agent/service allocation and deployment, which are subject to various criteria such as availability of resources and generation of unexpected faults.

More specifically, our approach is influenced by the three levels identified in the ALIVE project, namely; the organisation, coordination and services. Each of the levels plays an important role in MAS. For example, organisation provides the structure, relation and rules of agents, coordination specifies the allowable patterns of interaction and services provide the rules of engagement in terms of services. This multi-layer conceptual separation of concerns provides a number of architectural advances, based on the fundamental concepts of decoupling and modularisation.

In order to reflect this architectural alignment within the ALIVE project the adaptation process has to cross both directions (bottom-to-top and top-to-bottom) in the multi level hierarchy. Thus, changes in the service level may require adaptations of the coordination model and in turn changes in the coordination model may require changes of the organisation structure. Very similarly, this adaptation dependency is implied in the opposite direction from organisation to coordination and services. In that way, the ALIVE architecture remains highly adaptive across its inner and cross levels. At implementation level, the dependency of inner adaptations is maintained by linking the Organisation, Coordination and Service handlers, whereas cross dependency via transformations.

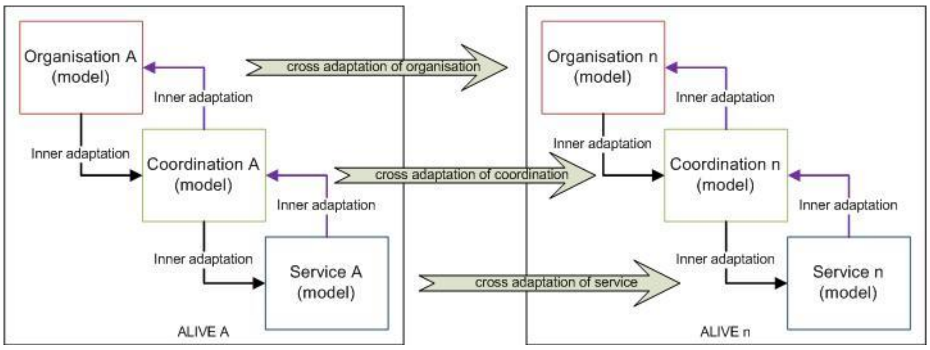
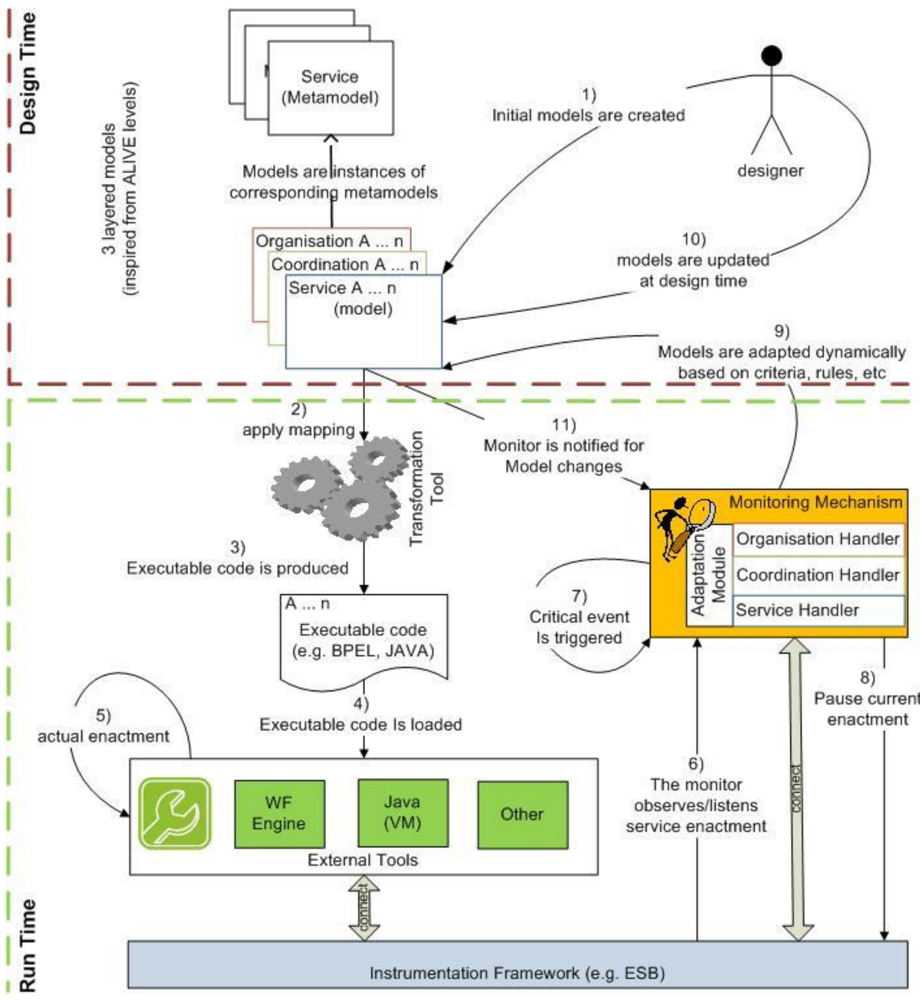


Fig. 1. Maintaining Multi-Levels of Model Adaptation

3.1 Adaptation steps and process

Conceptually, within MDE each of the ALIVE levels is formalised and represented with a corresponding metamodel. The models which are diagramming instances of the ALIVE metamodel are created by designers using specialised graphical tools. After models have been constructed, model transformations are defined to create executable process specifications in languages such as BPEL. Specialised tools (engines) can then load the executables and initiate the enactment of the modelled ALIVE scenario.

Process executions are instrumented with a monitoring framework, which listens for significant events during the execution of a given process. When a significant event occurs the monitor is notified and the control is transferred on the corresponding handler. The handlers are interlinked to reflect the architectural dependencies among levels, and maintain the process of inner adaptation. Connectivity among external tools (engines) and the monitoring mechanism is maintained by a middleware instrumentation framework.



Design Time
Run Time
3 layered models (inspired from ALIVE levels)
Service (Metamodel)
Models are instances of corresponding metamodels
Organisation A ... n
Coordination A ... n
Service A ... n (model)
1) Initial models are created
designer
10) models are updated at design time
9) Models are adapted dynamically based on criteria, rules, etc
2) apply mapping
Transformation Tool
11) Monitor is notified for Model changes
3) Executable code is produced
A ... n
Executable code (e.g. BPEL, JAVA)
4) Executable code Is loaded
5) actual enactment
WF Engine
Java (VM)
Other
External Tools
connect
Monitoring Mechanism
Adaptation Module
Organisation Handler
Coordination Handler
Service Handler
7) Critical event Is triggered
8) Pause current enactment
6) The monitor observes/listens service enactment
connect
Instrumentation Framework (e.g. ESB)

Fig. 2. Our Mutual, Multi-Layered Adaptation Approach.

The process steps can be distinguished into three phases as follows:

Initialisation phase: The initialisation phase corresponds to the design time and the generation of the executable code. The first step is the creation of the organisation, coordination and service models by the architect (1) using design tools. The models which are instances of the ALIVE metamodel depict a particular use case scenario such as Thales. At design-time the designer can also specify automatic execution adaptations that will be executed by the adaptation module. The models are next sent to predefined model transformations (2) to automatically create executable code (3), such as BPEL and WSDL. Then, execution tools load the executable code and initiate enactment (4). During the execution (5), a monitor mechanism observes execution and listens for specific significant events (6) controlled by conditions, rules etc.

Model adaptations due to events/failures in service enactment: During the execution of the application, adaptations may occur depending on the significant events. Initial plans may not be possible to be performed due to limited availability of resources, failures and other external reasons. These (critical) events are captured by the monitoring mechanism and passed on the corresponding (organisation, coordination, service) model handler for an adaptation action (7) whereas the current service enactment is suspended (8). As a result, the corresponding model handler dynamically updates/adapts existing models to new ones (9). Depending on the rules, adaptations may be propagated internally between the successive inner levels of ALIVE. Once the new models are produced, the generation process produces new executions by using steps (2-3-4) and the service enactment restarts (5).

Adaptation of service enactment due to design alterations: Alternatively, adaptations can occur as a result of a manual modification of the models by the architect while service enactment (10). The monitor mechanism is notified for the model changes (11) and the current enactment is suspended (8). Once more new executable code is generated by steps (2-3-4) and an updated enactment restarts (5).

4 Applying the Approach with an ALIVE Scenario

At this point, we present how the two-way dynamic adaptation of models and service enactment is maintained with a motivation example. The example describes a crisis management scenario from THALES [6, 7] used in the context of ALIVE project [2]. More specifically, the scenario describes how the Dutch Ministry of Internal Affairs manages an emergency depending on the severity of an incident, by defining five GRIP levels of emergency handling. Each level specifies the tasks, roles, authorities and responsibilities of the members involved in the handling of an incident. For purposes of simplicity, in this paper we consider an emergency scenario scaled from GRIP 0 to 1. GRIP 0 describes how to handle a routine accident where no major coordination is required, whereas GRID 1 describes how many different authorities coordinate at an operational level.

4.1 Initialisation phase

Initially, at design time the organisation, coordination and service concepts of the THALES scenario are modelled at GRIP 0 level by the designer. In this example a combination of UML 2.0 diagrams are used to depict effectively these concepts with Class/Collaboration, Interaction and Component models respectively.

Organisation: At GRIP-0, the organisation consists of few structures. Most importantly, the *CrisisManagement* class has a *GripLevel* attribute to maintain the current state of the incident. *CrisisManagement* is related to at most one (see optional cardinality [0..1]) *Ambulance*, *Fire_Fighting_Team* and *PoliceOfficer* classes. The *Handle_Incident* collaboration depicts how a *PoliceOfficer* playing the role of *securePlace*, an *Ambulance* by *provideTreatment* and a *Fire_Fighting_Team by extinguishFire* collaborate with one another to handle an incident.

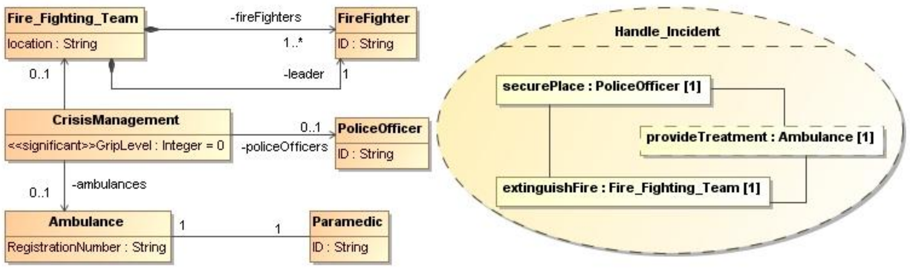


Fig. 3. Organisation models at GRIP 0

Coordination: At GRIP-0, the coordination (describes the possible interactions among members) for handling an incident is specified in a network-like relation. All parties have equal responsibility in resolving the situation and communicate via *inform* methods and exchange *incident* information.

Service: At GRIP-0, the services/agents are limited to those of a *FireService*, *PoliceService* and *AmbulanceService*. The services have to implement the interaction structures specified at coordination level and expose the relevant operations and interfaces.

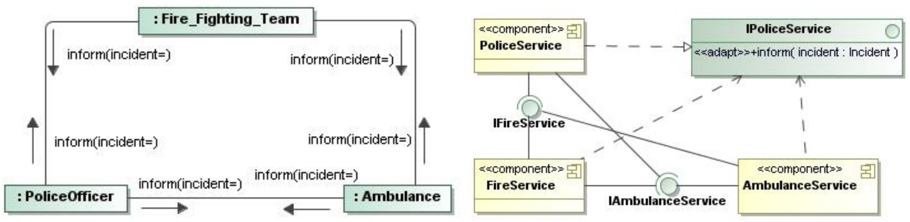


Fig. 4. Coordination (left) and Service (right) models at GRIP 0

Later the coordination patterns and interfaces will be transformed to corresponding Web service implementations for BPEL and WSDL via predefined model transformations. At this point we do not present the details of the transformation process, however there are many approaches in this regard see [8, 9]. Next the generated artefacts are loaded for execution to an execution engine such as Apache's Orchestration Director Engine (ODE) [10].

The significant events need to be marked with stereotypes and tag values on design models, so appropriate handlers can be created. For example, in fig.3 we have marked the property *GripLevel* of *CrisisManagement* as significant, so an appropriate handler can be created to monitor the state changes during enactment. Similarly, exceptions on interface operations can be marked as adapted, indicating that a handler needs to be generated and the path of enactment needs to be changed.

Finally, specific adaptation rules are defined by the designer and attached to models. These rules define the adaptation patterns to be followed in case of a significant event. The rules may be specified in a QVT-like language or refer to other implementations of ontological or rule-based languages. The handlers are capable to interpret these rules and perform the adaptations.

4.2 Model adaptations due to events/failures in service enactment

During the execution of the workflow, significant events may be triggered and processed by the monitoring mechanism. The events may propagate a series of inner adaptations from their corresponding handlers to design models as seen in chapter 3.

Thus, during the execution of the *PoliceService* by an agent, an error may occur due to some unavailable resources. In this case the models have to be adapted at run-time with new enactment plans which first need to be constructed. The adaptation process is directed by the adaptation pattern associated with the significant event and retrieved from the model. The pattern may be specified in model-driven native specification (QVT based) or other (rule-based) language. In the first case the adaptation is performed as an ordinary transformation, where in the latter it is performed by a dedicated tool.

4.3 Adaptation of service enactment due to design alterations

The most obvious adaptation case is when a service execution needs to be updated due to design alterations. In this case, the initial design models of organisation, coordination and service has been adapted with new structures/roles, coordination patterns and service functionalities. In our scenario, this is because the designer due to some external circumstances has re-evaluated the severity of the incident from *Grip-Level* 0 to 1. In the opposite direction now, the changes in models would propagate events which may cause a sequence of inner adaptations. Finally, from the adapted models an updated service enactment will be generated.

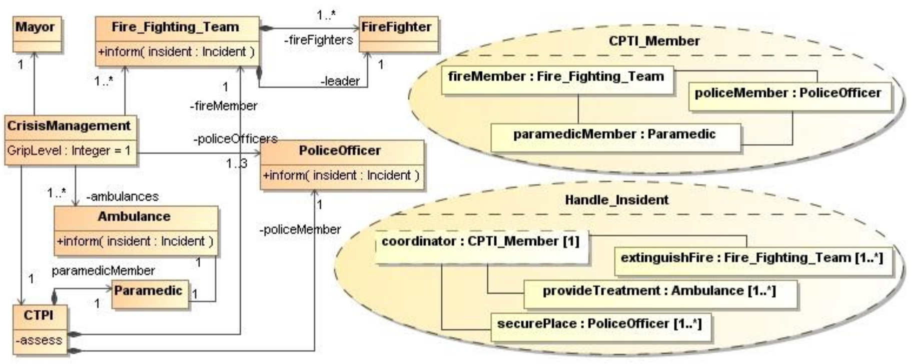

Mayor
Fire_Fighting_Team
+inform(insident : Incident)
FireFighter
-fireFighters
-leader
1..*
1
CrisisManagement
GripLevel : Integer = 1
-fireMember
-policeOfficers
1..3
PoliceOfficer
+inform(insident : Incident)
-ambulances
Ambulance
+inform(insident : Incident)
-policeMember
paramedicMember
Paramedic
CTPI
-assess
CPTI_Member
fireMember : Fire_Fighting_Team
policeMember : PoliceOfficer
paramedicMember : Paramedic
Handle_Insident
coordinator : CPTI_Member [1]
extinguishFire : Fire_Fighting_Team [1..*]
provideTreatment : Ambulance [1..*]
securePlace : PoliceOfficer [1..*]

Fig. 5. After design-adaptation Organisation models at GRIP 1

Organisation: In GRIP-1, a local coordination team (CTPI) is now set up to supervise the operations and a *Mayor* entity is introduced. The CPTI team is composed of the heads of active services, such as *Fire_Fighting_Team* and *Police-Officer* and *Paramedic*. Additional forces have been reserved, so cardinality has changed to [1..n]. A new collaboration *CTPI_Member* defines the additional roles of *fireMember, policeMember* and *paramedicMember,* which can be played by existing handling members such as a *PoliceOfficer*. Finally, within the *Handle_Incident* participation, all police, ambulance and fire units on the ground communicate through a *CPTI_Member*, playing the role of a *coordinator*.

Coordination: At GRIP 1 the *Mayor* does not play an active role (there are no out-coming interactions), however he/she might get *informed by* the CTPI members. How information is exchanged and shared among members has also changed from a network to a hierarchical structure. Now every incident handler has an obligation to report directly to CTPI members. CTPI has also the right (permission) to delegate tasks to other non-CTPI members, whereas other non-CTPI members have the obligation (implement the interface which is accessible only to *CPTI_Members*) to perform the tasks delegated to them.

Services: At GRIP 1 two additional services *MayorService* and *CPTIService* are introduced. Previous services have also been altered in order to be consistent with the new coordination patterns. As a result, a *CPTIService* utilises the corresponding provided interfaces of *FireService*, *PoliceService* and *AmbulanceService* to delegate tasks as well as the *MayorService* to provide incident reports.

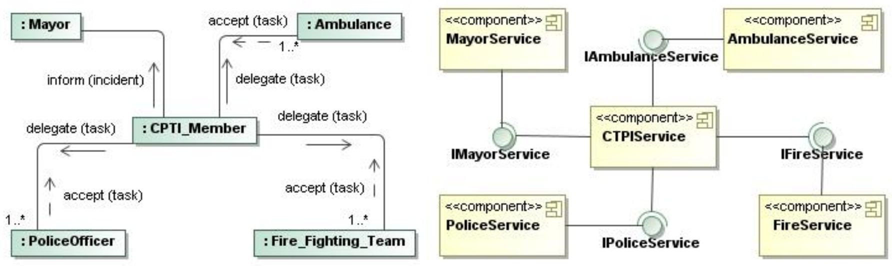


Fig. 6. After design-adaptation Coordination (left) and Service (right) models at GRIP 1

5 Other Related Work & Discussion

Another quite related approach to the concept of run-time models is that of executable UML [11]. Executable UML is based on rich diagrams that can produce executable models, which can then be translated directly to code. In this case a virtual machine interprets the UML models without any intermediate code generation step involved.

In our case the run-time models are represented by ordinary UML diagrams capturing the organisation and coordination of dynamic instances of an ALIVE scenario. Consistency among the service enactment (execution) and design models is maintained by the specification of significant events that are bound with specific state changes. Thus our approach does not provide a full bidirectional consistency among real execution (states) and dynamic models as the overhead is significant. Further, our ALIVE models are not executable; however they generate artefacts which can be executed via a transformation process. Specific markings are also used to identify the significant states requiring monitoring and operations that may trigger an adaptation process.

6 Conclusions

Providing mechanisms facilitating the dynamic adaptation of design models and run-time executions is an important property for systems that need to reflect the environmental and design changes. In this paper we have proposed a mutual monitoring mechanism for maintaining adaptations among design models and service enactment. The run-time adaptations are performed automatically, triggered by significant events, directed by adaptation patterns described at design-time and implemented via model transformations. In addition, we have shown how the multi-layers of model abstractions add significant complexity in the adaptation process, which also needs to be supported by the mechanism.

References

1. Jeffrey, O.K. and M.C. David, *The Vision of Autonomic Computing.* Computer, 2003. 36(1): p. 41-50.
2. ALIVE. *Coordination, Organisation and Model Driven Approaches for Dynamic, Flexible, Robust Software and Services Engineering*. European Commission Framework 7 ICT Project 2008; Available from: http://www.ist-alive.eu/.
3. Clarke, S., A. Staikopoulos, S. Saudrais, J. Vázquez-Salceda, V. Dignum, W. Vasconcelos, J. Paget, L. Ceccaroni, T. Quillinan, and C. Reed, *ALIVE: A Model Driven approach for the Coordination and Organisation for Services Engineering*, in *to appear on International Conference on Model Driven Engineering Languages and Systems (MODELS 08) Research Project Symposium*. 2008: Toulouse, France.
4. France, R. and B. Rumpe, *Model-driven Development of Complex Software: A Research Roadmap*, in *2007 Future of Software Engineering*. 2007, IEEE Computer Society.
5. Wooldridge, M. and N. Jennings, *Intelligent Agents: Theory and Practice.* In The Knowledge Engineering Review, 1995. 10(2): p. 115-152.
6. Splunter, S.v., T. Quillinan, K. Nieuwenhuis, and N. Wijngaards, *Alive Project: THALES Usecase - Crisis Management.* Technical Report ALIVE Project, 2008.
7. Aldewereld, H., F. Dignum, L. Penserini, and V. Dignum, *Norm Dynamics in Adaptive Organisations.* 3rd International Workshop on Normative Multiagent Systems (NorMAS 2008), 2008.
8. Bézivin, J., S. Hammoudi, D. Lopes, and F. Jouault, *An Experiment in Mapping Web Services to Implementation Platforms*, in *Technical report: 04.01*. 2004, LINA, University of Nantes: Nantes, France.
9. Bordbar, B. and A. Staikopoulos, *On Behavioural Model Transformation in Web Services.* Conceptual Modelling for Advanced Application Domain (eCOMO), 2004. LNCS 3289: p. 667-678.
10. Foundation, T.A.S. *Apache ODE (Orchestration Director Engine)* 2008; Available from: http://ode.apache.org/.
11. Stephen, J.M. and B. Marc, *Executable UML: A Foundation for Model-Driven Architectures*. 2002: Addison-Wesley Longman Publishing Co. 368.