Chapter 5
Context Aware and Adaptive Systems

Alan Colman, Mahmoud Hussein, Jun Han and Malinda Kapuruge

Abstract Context aware software systems and adaptive software systems *sense* changes in their environments, and *respond* by changing their behaviour and/or structure appropriately. The perspective of these two approaches, however, tends to differ. Context aware systems focus on modelling and reasoning about the relevant environmental context often with aid of formal ontologies The system, however, can only respond to an *anticipated* change of configuration setting or a change of application mode. Adaptive systems in contrast focus on how the *system responds* to an *unanticipated* environmental change. However, adaptive systems typically lack sophisticated models of context. This chapter analyses the differences and similarities between context-aware and adaptive systems. It then describes an approach and framework called ROAD that supports the development of context-aware applications whose structure and behavior can be altered at runtime. ROAD provides mechanisms to acquire and record context information and provision a central store of 'facts' which are evaluated in rules. These rules mediate operational messages or trigger adaptations to the structure of the application.

A. Colman (✉) · J. Han
School of Software and Electrical Engineering, Swinburne University of Technology,
John Street, Hawthorn, PO Box 218, Melbourne, Australia
e-mail: acolman@swin.edu.au

J. Han
e-mail: jhan@swin.edu.au

M. Hussein
Menofia University, Menofia, Egypt
e-mail: mahmoud.hussein@ci.menofia.edu.eg

M. Kapuruge
DiUS Computing Pty Ltd, Melbourne, Australia
e-mail: mkapuruge@dius.com.au

P. Brézillon, A. J. Gonzalez (eds.), *Context in Computing,*
DOI 10.1007/978-1-4939-1887-4_5

5.1 Introduction

Context-aware Software systems and Adaptive Software systems have many similarities. Both types of system *sense* changes in their environments, and *respond* by changing their behaviour and/or structure appropriately. The perspective of these two approaches, however, tends to differ. Context-aware systems focus on modelling and reasoning about the relevant environmental context often with aid of formal ontologies (e.g. Wang et al. 2004; Chen et al. 2003; Ranganathan et al. 2003; Bettini et al. 2010). While these context models may be rich and the reasoning complex, the context-aware system response typically tends to be simple—that is, an *anticipated* change of configuration setting or a change of application mode. Adaptive systems on the other hand focus on how the *system responds* to environmental context change or to changes in requirements. In adaptive systems the range of possible system configuration states is much larger. While some of these system states might be anticipated typically there being so many options available for configuration or regulation that not all configuration states can be anticipated. However, in general adaptive systems lack the sophisticated models of context apparent in context aware systems.

This chapter explores how the gap between context aware and adaptive systems, in particular adaptive service-based compositions, might be systematically bridged through a domain-specific language and framework that allows the rapid development of CAAS systems—Context-Aware Adaptive Systems (Nierstrasz et al. 2009; Hussein et al. 2011). To show how this gap can be bridged we will start from the perspective of adaptive systems, analyse the commonalities and differences between context-aware and adaptive systems, then discuss how richer models of context might be incorporated into such systems.

The structure of this chapter is as follows. In Sect. 5.2, we give an overview of approaches to adaptive software systems in terms of the goals of the adaptations, the environmental variables being monitored and modelled, the type of change enacted and how change is realised into the running system. In particular, we look at approaches that provide the potential to model more complex environments and user-focused adaptation. Section 5.3 presents a motivating scenario that shows the need for a system that is both context-aware and adaptive. Section 5.4 sets out the general requirements that we would need for models and a framework that can support the development of context-aware adaptive software systems—in particular adaptive service compositions. It also addresses some of the challenges faced in the development of such systems. Section 5.5 describes the approach for specifying the context, functional and adaptation properties of the system using the ROAD framework which is a model-driven approach to creating decoupled context-aware adaptive service compositions. We show how this approach meets the requirements identified in Sect. 5.4. Section 5.6 discusses related work in terms of our characterisation of context aware and adaptive systems, with Section 7 concluding the chapter.

5.2 Adaptive Systems

In the past decade or so there has been extensive research into how to create software systems that can change themselves in response to their environment (e.g. Cheng et al. 2008; Bradbury et al. 2004; Patikirikorala et al. 2012). Common with all these approaches is their aim to achieve some *goal* related to the system in its environment through definition of some form of loop whereby the environment and/or the system itself is *monitored*, the information gathered is *analysed*, a *decision* is taken as to what change is needed in the system in response, and these changes are then *enacted* in the system. For example, IBM in their vision of computer systems that behave 'autonomically' called this feedback loop a MAPE-K loop (Monitor, Analyze, Plan and Execute using a shared Knowledge base) (Kephart and Chess. 2003).

In this section, we will give a very brief overview of approaches to adaptive software systems based on the categories defined in a control loop (goal, monitoring and analysing the environment, making and enacting decisions) and contrast these approaches to work on context-aware systems.

5.2.1 Goal of Adaptation

Adaptive systems have some degree of self-management, for example self-configuring, self-optimizing, self-healing, and self-protecting—so-called *self-** properties. As well as varying goals, the definition varies as to scope of the 'system' that needs to be self-managed. In the case of IBM's autonomic initiative, the elements are typically regarded as assets within enterprise computing environment (e.g. servers, databases, network infrastructure, etc.) in order to reduce the amount of manual intervention required when components in the system fail or need to be changed. Other work has looked at lower levels of abstraction such as the allocation of server resources to optimise for energy efficiency. For example, cloud infrastructure providers continually need to automatically monitor and adapt the efficient provisioning of resources as user-demand and availability of servers change. Yet other work is focused on the software level, in particular managing service compositions in order to achieve service level objectives, or the changing availability/performance of constituent elements. In this case the systems goals and associated rules reflect real-world business relationships between service consumers and providers.

In contrast, in context-aware systems the goal has been typically to enable an application to adjust to its context of use or task, potentially taking into account user preferences (Dey et al. 2001). This adaptation may be restricted to the user interface level only, or may impact on the configuration of the functional system.

5.2.2 *Model of the Environment and Reflective Representation of the System*

In much of the work on adaptive systems, the aim is to change the behaviour and/or structure of the system in order to keep some monitored variable in line with a goal. These are typically performance variables such as response time, throughput, or reliability; or alternatively resource consumption variables such as memory used or energy consumed. As such, for adaptive systems the environmental context-in-focus is the *execution context* of the software application rather than just the *domain context* of the user (location, time, social situation etc.).The execution context can typically be well defined in terms of resource parameters, performance parameters, deployment or network topology, etc. In practice, it is therefore rare to see context ontologies used in adaptive systems as the environment is well defined in terms of what parameters are measured. Context-aware systems on the other hand often need to model and reason about the "messy" and perhaps uncertain domain context of the real world user.

While many software systems including context-aware systems exhibit various degrees of ability to adapt themselves, adaptive systems need to maintain a reflective runtime model (Bencomo 2009) of their own structure and/or behavior (Cheng et al. 2008). This reflective representation is used to reason about and trigger changes in the system. The 'self-awareness' of such systems means that changes can often be handled automatically compared with conventional systems that require off-line re-design, implementation and redeployment. The level of granularity of these models can vary greatly, from the code level (e.g. Wang et al. 2004) through to high level architectures or service compositions (e.g. Colman and Han 2007). Software architectural models are a course-grained view of the system as a set of components and connectors. Such models assume a closed computing environment. Service-oriented architectures, on the other hand, operate in much more open environments where the components or services that the application relies upon are not necessarily under the control of one organization. Such service-oriented compositions rely on the dynamic binding of services that are 'self-describing' using standards such as the Web Services Description Language (WSDL). The relationships between these 'loosely-coupled' services therefore need to be actively managed as requirements and service provisioning changes. The form of representation of models also varies from formal control models that model behavior (e.g. Patikirikorala et al. 2012) to structural models (e.g. Magee and Kramer 1996) that can automatically be composed, to more informal declarative representations (e.g.Bradbury et al. 2004; Garlan et al. 2004).

Context-aware systems in contrast do not necessarily maintain an explicit reflective model of the system. Rather, the system will have a number of predefined modes which are selected depending on the state of the sensed context. As context-aware systems become more adaptive their reflective models need to become commensurately more sophisticated.

5.2.3 Making and Enacting Decisions

Given a representation of the system and its environmental context, the nature of the control loop in adaptive systems also varies greatly between approaches, and depends on whether the system's behaviour is being *regulated*, or its structure *reconfigured*, or both. *Regulatory control* focuses on changing the *behavior* of the system assuming a fixed composition structure. Two predominant forms of regulatory control are control-theoretic and rule-based. *Control-theoretic* approaches create formal mathematical models of the behavior of the system and aim to maintain the system at some desired set-point in the face of environmental perturbations. Such approaches predominantly use blackbox feedback control where the perturbations are unmodelled and there is no explicit model of context (see Patikirikorala et al. 2012 for a comprehensive survey). *Rule-based regulation,* on the other hand, can explicitly reference contextual variables in the form of rule conditions that regulate interactions over the program or composition structure. Challenges for rule-based regulation include ensuring consistency of the rule-base both in their definition and in application, and ensuring that valid, non-oscillatory desirable behavior results from the application of complex rules sets (Cheng et al. 2008; Mannaert et al. 2012). *Adaptive reconfiguration* on the other hand focuses on maintaining a model of the architectural structure of the system, i.e., how the system is composed from components or services. Component based approaches (e.g. Magee and Kramer 1996) focus on the compatibility and controlled composition of required and provided interfaces (sometimes through connector components), whereas service-based compositions typically model a variable business process that manages two levels of indirection: (i) the relationships between abstract services and (ii) the binding of concrete services to those abstract services. Some of the key concerns that these compositional approaches need to address include ensuring the functional correctness of each architectural configuration; monitoring and analysing the relative performance of various configurations; coping with change in components/services bound to the composition; and safely transitioning between configurations at runtime. While some simple forms of reconfiguration control rely on selection of an appropriate configuration from a predefined set, more truly adaptive systems use rules, tactics, strategies or other planning techniques to enable effective change while ensuring that structural and behavioral constraints are not violated. The autonomic vision of adaptive systems (Kephart and Chess 2003) sees such systems as recursive compositions of self-managing systems that can communicate with each other on both functional and management levels.

In adaptive systems changes occur at runtime. These changes must be reflected both in the system model and in the runtime application itself. Many approaches use a model checking mechanisms to ensure any planned changes are consistent and beneficial, before any of the planned changes are committed to the runtime system. Changes in the runtime system (e.g. unavailability of a service) need to be reflected back in the model so that appropriate decisions can be made. Mechanisms are therefore required to keep the model and the runtime in sync. In contrast, the

number of states in context-aware systems is typically more limited with a set number of configurations as such system contains no reflective model that can be manipulated. As these configurations have been predetermined, their validity can be checked at design time. Decision-making is therefore often a matter of simple switching of modes based on some in-built logic.

Depending on the type of system, the "intelligence" required to make a control decision can either be built into the system itself (for example through either rule design or through some reinforcement learning mechanism), or can be exogenous to the system (Colman 2007; Colman and Han 2005). In the latter case the adaptive system only provides the runtime model of the flexible system and the mechanisms for adapting the system while the decision about *what* to adapt is made by others. Approaches such as (Garlan et al. 2004) and (Kapuruge et. al. 2014) do a combination of both.

5.2.4 Engineering Context-Aware and Adaptive Systems

It is clear from the above discussion that there is no sharp distinction between context-aware and adaptive systems. Both need to continually monitor their runtime context in order to make appropriate changes. It is also clear that adding context-awareness or adaptivity to a basic functional software system adds considerable complexity to the development task. Supporting the engineering of such systems with appropriate methodologies, architectures, frameworks and tools therefore becomes necessary. In context-aware systems, it has been long recognised that the acquisition and management of context should be treated as a separate concern from the underlying functional system (Dey et al. 2001; Henricksen and Indulska 2004). Likewise, adaptive software frameworks typically maintain the separation at both the conceptual and implementation levels between the management of the system and the system's functionality, albeit within the autonomic element (Colman 2007). Another common approach that assists the control of the complexity inherent in both context-aware and adaptive systems is to use model driven frameworks that enable such systems to be defined at a higher level of abstraction and then (semi-) automatically generated.

In this chapter we will show how adaptive systems, based on a rules-based declarative service-composition approach, can incorporate some of the more complex aspects of context apparent in context-aware systems. This approach maintains a separation between functional, management, and contextual requirements and is supported by a set of tools and framework that enables the ready development of context-aware adaptive service compositions.

In summary, the table below characterizes some of the prototypical differences between context-aware systems and adaptive systems. In Sect. 5.6 on related work we discuss the extent to which various adaptive and context-aware approaches take these characteristics into account.

	Context-aware systems	**Adaptive systems**
Goal	Present appropriate interface or functionality based on context of use. Abstraction at the application level	Maintain system objective in response to environmental perturbation or changing requirements. Abstraction possibly at many levels (application, network, resource, . . .)
Model and analysis of the environment/user	Complex model of the domain and user context. Need to reason about domain semantics and user preferences	Simple representation of the environment in the form of parameters in the domain environment or the software system infrastructure
Representation of the system	Operational view of the system modes	Explicit behavioural or structural model of the system
Decision making and enactment	Selection of pre-defined configuration modes based on rules and utility models	Tuning system operational parameters or altering composition of system structure based on (multiple) objectives
Engineering models and modularity	Separation of functional aspects from context acquisition	Separation of functional aspects from adaptation management

5.3 Motivating Scenario for CAAS

Let us consider a travel guide application service that composes a number of other services to create travel itineraries based on user preferences. These services provided both the functionality of the system (e.g. route planners, user profile services, etc.) and the *domain context* information the system needs in order to function (e.g. weather and traffic information, attraction finder services and so on). Even application-specific functions like the derivation of inferred context might be outsourced as a service. In this scenario all functionality is provided by services external to the composition. The role of the composition is to define a process that takes user requests (e.g. plan itinerary given a set of attraction-types, time available, preferred transport modes etc.), obtains relevant contextual information (e.g. weather and road conditions), sends this information to a service that recommends to the user a set of attractions, which on selection is sent to a route planning service that creates the final itinerary given the user's current location. This scenario of the service composition with its functional and context services is illustrated in Fig. 5.1.

This application needs to adapt in a couple of ways. Firstly, during runtime operation the composition needs to be aware of its *execution context*. In the case of a service-oriented system this includes the availability or otherwise of services it already knows about. For example, moving between regions the application may have to switch between alternative traffic information providers. This management capability might be realised by rules embedded in the composition, or the capability

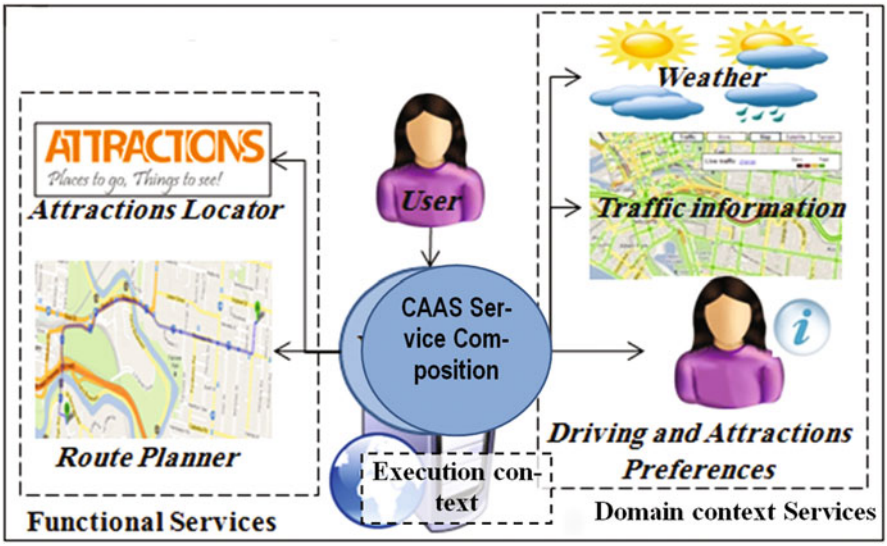

Fig. 5.1 Functional and context services

itself might be externalised in the service. Secondly, the provider of the travel guide service wants to ensure that they can readily incorporate new *types* of service into the composition (e.g. a transport disruptions service) without creating disruption to users of the composite service. The composition needs to therefore be modifiable without being taken off-line to go through another redesign/implementation-deploy cycle.

5.4 Requirements for a CAAS Framework

The above scenario suggests a number of general requirements that need to be met by any CAAS service composition framework. These include the ability to:

1. Mediate messages between functional services based on *domain context* information provided by context provider services.
2. Be able to alter the structure of the composition at runtime based on the *execution context*.
3. Be able to incorporate new types of behavior over a given structure by defining adaptable processes that can be changed at run time.
4. Readily incorporate not only new instances of services whose types are already known (service selection) but also incorporate new *types* of service (functional or context) into the composition without disruption to current process instances, and define the interactions between those new services and other services.

5. Incorporate new types of context information into the composition along with rules and make use of this information to mediate interactions or to handle changes in execution context.

As discussed in Sect. 5.2.4, given the complexity of CAAS systems, the engineering of such systems needs to be not only model-driven but maintain a reflective runtime model. On one hand, separation of concerns needs to be maintained between functional, context and management aspects while, on the other hand, facilitating the integration of these aspects into well-defined, deployable modules that have some degree of self-management.

5.5 A Model Driven Rules-based Approach to Implementing a CAAS Framework

To address the above requirements, in this chapter we propose a model-driven rules-based approach and framework for developing a CAAS applications. This approach is based on the clear separation between function, context, and management aspects as identified in Sect. 5.2, and incorporates these aspects into an integrated managed service composition. In a CAAS application, the structure and behaviour can be affected by not only changing state of the application but by changes in *domain* and *execution* contexts. What therefore is needed is a common format to represent this context information so that appropriate rules can be applied. To do this we adopt the event–condition–action (ECA) approach as:

- *Events* are generated as messages received by and passing through the composite. These messages can either be functional messages being mediated by the composite, or they may be messages indicating change of context which require updating of *facts* stored by the composite.
- *Conditions* are evaluated based on the stored facts. These facts can either be context acquired from external services, or be a reflection of the state of the process or composite itself.
- *Actions* arising from execution of rules can result in mediation of messages passing through the composite (e.g. message routing); generation of messages to services reflecting the state of the process or composite; generation of new facts either reflecting the internal state of the composite or its external context; firing of events which are then further evaluated by rules; automatic operational management actions (e.g. selection of an alternate to service based on availability); or generation of messages sent to management services/operators indicating need for re-configuration of the composite.

The following subsections describe how composite structure is defined, how the operational issues and adaptive behaviour is conditioned by context information, how this context information is acquired and provisioned, and how the development process is supported by a framework and tool chain.

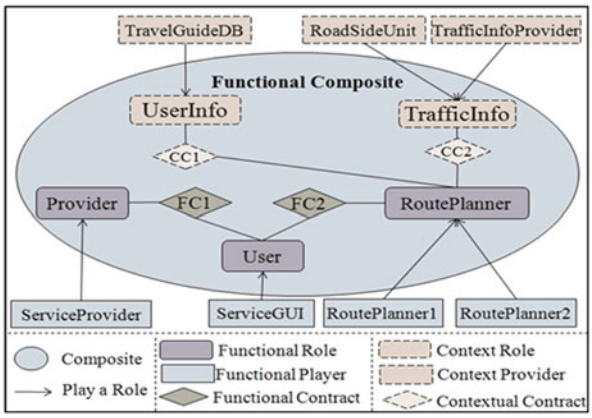

Fig. 5.2 Functional composite with domain context providers

5.5.1 Composite Structure

To create the service composition that is both context aware and adaptive, we extend our existing approach to creating adaptive service composites called ROAD (Role Oriented Adaptive Design) (Colman and Han 2007; Kapuruge et al. 2011b). This chapter will provide a very high level overview of the ROAD framework. A ROAD composite structure is defined in XML along with associated rule files. These descriptions are deployed to a ROAD4WS container which contains a component called ROADfactory that generates the run-time service. The interested reader is referred to (Kapuruge et. al 2014; Kapuruge et al. 2012; Kapuruge et al. 2013a, 2004; Talib et al. 2010; King and Colman 2009) for more details[1]. The purpose of this overview is to show how context facts are acquired and how they are used in the operation and adaptation of the service composite.

ROAD is based on an organisational paradigm which defines the service composite as a role structure. Roles represent an abstract service interfaces to which concrete services ("role players") are dynamically bound (Kapuruge et al. 2011b). Role players can be functional services, context provider services, or management services. Internally, the relationships between roles are represented by two types of contracts (i.e. functional and contextual) which define permissible interactions between roles. Figure 5.2 above illustrates a role structure based on our scenario with both external functional and context provider services attached to the composite roles.

For example, a functional contract "FC2" exists between the user (role A) and route planner (role B) roles as shown in Fig. 5.2. The contract has a set of permissible interactions between the contracted roles as shown in Table 5.1. Each interaction has (1) an identifier (e.g. i2); (2) an operation that needs to be performed by requesting that interaction and the operation has a name (e.g. PlanRoutes2) and a set of input

[1] The ROAD schemas and framework can be viewed and downloaded from https://github.com/road-framework.

Table 5.1 Part of the functional contract "FC2"

Functional Contract ID FC2: User_RoutePlanner
Parties: RoleA: User; **RoleB**: RoutePlanner;
Interaction Clauses:
i1: {PlanRoute1 (Destination, CurrentLocation), AtoB, Routes};
i2: {PlantRoute2 (Destination, CurrentLocation, TrafficInformation), AtoB, Routes};
...

Table 5.2 The contextual contract "CC1"

Contextual Contract ID CC1: TrafficInfo_RoutePlanner
Parties:
Context Source: TrafficInfo; **Context Consumer**: RoutePlanner;
Context Attributes:
a1:String: TrafficInformation;

parameters (e.g. destination, current location, and traffic information); (3) a direction to specify who is responsible for providing the operation included in that interaction; and (4) a return type (e.g. Routes).

Another type of contract is the contextual contract to define (represent) context information that is needed by the system roles (i.e. the context model). For example, the contract "CC1" shown in Table 5.2 specifies that the route planner role needs to know the live traffic information to calculate the routes effectively.

In addition to functional and context provider role interfaces, ROAD composites provide a management ("organiser") interface that allows the structure to be modified at run time. This interface provides a set of standard CRUD methods (a full list can be found in Appendix C of (Kapuruge et. al. 2014)) for monitoring and adapting the composite (e.g. adding and deleting roles and contracts, inject rules into contracts, etc.). Such methods enable the runtime adaptation of the context model by changing the system's contextual roles and contracts, and the system's functionality by adding, removing, and changing the functional services of the system (Hussein et al. 2013).

5.5.2 Operational Behaviour

Each composite has a global repository of facts (a "fact tuple space" or FTS). The FTS stores facts related to both the internal state of the composite *and* to any relevant execution or domain context acquired via context roles. The composite also contains a number of points on the role-contract-role path at which a message may be mediated. These points have rule evaluation mechanisms (implemented in Drools[2]) which evaluate patterns of events/facts stored in a local "working memory". Events

[2] http://www.jboss.org/drools/

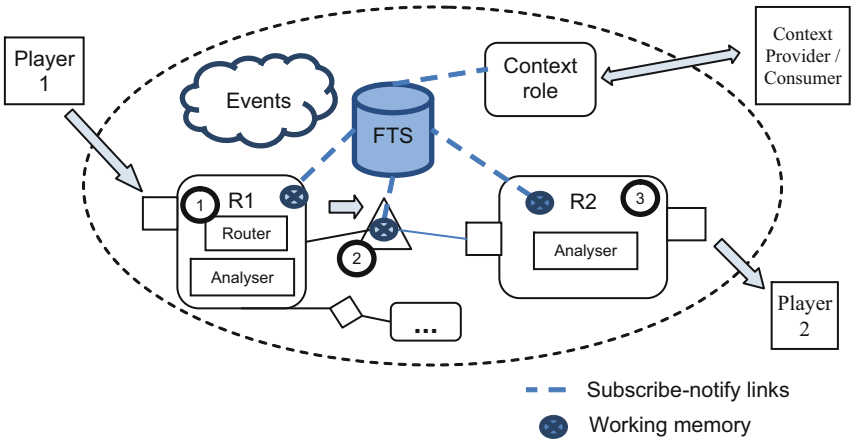

Fig. 5.3 Operational message flow

are triggered as messages pass through the composite. Facts in a working memory are obtained by it subscribing to relevant facts, including context facts, in the FTS. In the service composition behavioral mediation may be reactive (per message) or coordinated into a process. The flow of message through composite is illustrated in Fig. 5.3.

On receipt of a message at a role R1 ①, the message is transformed into the internal message format of the composite and routed to the relevant contract ②. This routing decision may be context dependent, for example a routing decision might depend on *execution-context* facts relating to the availability or loads on required services. A routing decision might be based on a *domain-context* rule that describes a particular user's preference for a service provider.

Likewise each contract contains a rule evaluation mechanism that can evaluate the messages against rules defined in the contract. These rules may be independent of context (e.g. is this type of message permissible) or maybe context-dependent (e.g. is the message permissable give the current location of the sender). Once the message has been processed by the contract it is passed to the outgoing role R2 ③ where it is transformed by the analyser object in that role to be sent to the player. This message transformation might (a) change the format/ordering of the message content and (b) incorporate extra information from other messages or facts from the FTS to make the message perceivable to the recipient.

Processes are implemented using Serendip (Kapuruge et al. 2012; Kapuruge et al. 2013b), which adds a coordination layer to the reactive message handling mechanisms of the ROAD framework. An example process is "plan route" shown in Fig. 5.4. Based on the live traffic information availability, a suitable route planning function is selected. Then, a set of routes are suggested to the user where she can select a route. In Serendip, processes can be viewed as event-process-chains (Kapuruge et al. 2013b) that compose units of behavior.

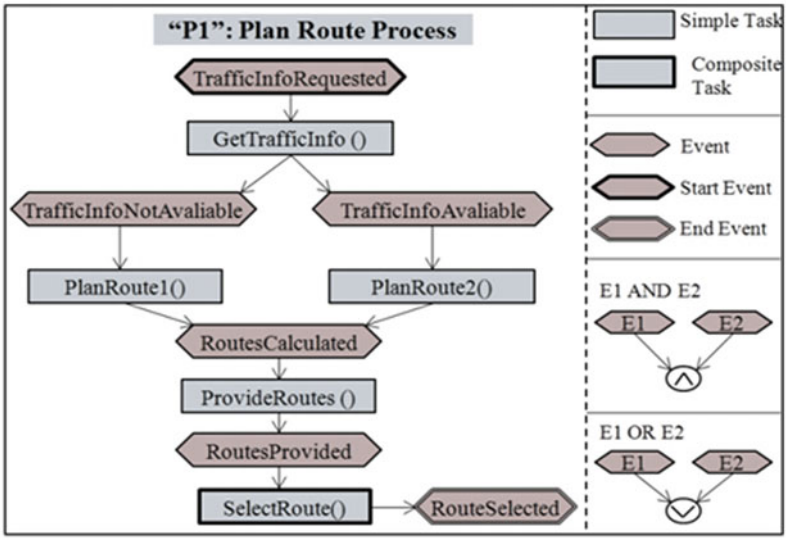

Fig. 5.4 An example behaviour process in the travel guide service

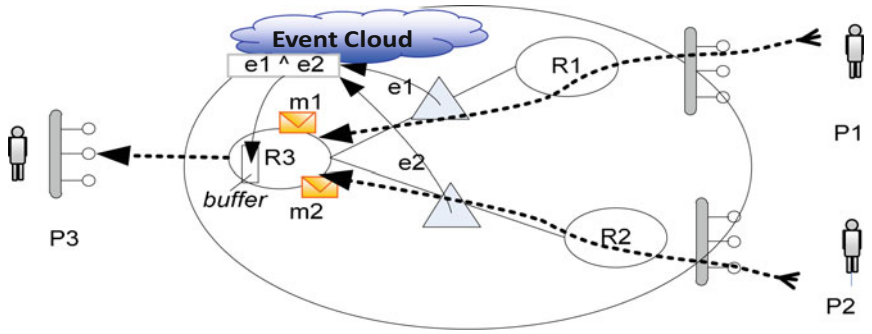

Fig. 5.5 Message coordination in an event driven process

At runtime, process instances are created from declarative process descriptions. The Serendip process engine enacts process instances in response to the events fed to the event-cloud. Typically, these events are published by the contracts into the event-cloud. Figure 5.5 illustrates a message being sent to a player P3 that requires prior receipt of messages from P1 and P2. Defining processes as a set of event-driven tasks with pre- and post-events not only makes the process much more readily adaptable but the evaluation of event conditions (e.g. $E1 \hat{} E2$) readily enables context state to be included in those conditions.

Table 5.3 A rule to cope with the unavailability of the traffic information

```
Rule "AdaptationRule1": {
When ValueChanges (TrafficInfoAvailability);
if TrafficInfoAvailability == False;
do RemoveContract("CC2"), Bind("RoutePlanner", "RoutePlanner2"),
        RemoveInteraction("FC2","i2"), RemoveRole("TrafficInfo"),
        RemoveTask ("P1", "GetTrafficInfo"), RemoveTask("P1", "PlanRoute2"),
        RemoveEvent("P1", "TrafficInfo Available")};
```

5.5.3 Adaptive Behaviour

As can be seen by the description of operational behaviour in the previous section, both message flow and process are sensitive to rules that evaluate, among other things, arbitrary context. Rules can also be defined at the global composite level which respond to anticipated changes in context to enact actions such as activation of role-player bindings, termination of a process instance, exception handling, generation of operational management messages to players, etc. An example adaptation rule from our scenario is given in Table 5.3. This rule is activated (i.e. event) when the traffic information is not available (i.e. condition). In response to this change, the service is adapted (i.e. action) by removing the contextual contract "CC2", binding the route planner role with the player "RoutePlanner2", etc.

For more complex decision-making potentially involving unanticipated situations, an external management player bound to the organizer role can subscribe to events and facts stored in the FTS. This player, who may be a program or a human controller, takes the appropriate adaptation decision based on information available, and then manipulates the composite as mentioned in section 5.5.1. This manipulation might be as simple as resetting the state of the system fact or as complex as the wholesale transformation of the composite structure. It is through this mechanism that the composite is also adapted to changing requirements.

5.5.4 Acquiring and Providing Context Information

Given that both operational and adaptive behavior can be conditioned by context information stored in the FTS, it remains to be described how such context information is acquired from external context providers. Or in the case where the composite self is a provider of context information to other services, how this information is made available to those services.

From an external point of view context roles and functional roles (as shown in Fig. 5.3 above) are identical. Both define provided and/or required service interfaces. The key differences between a context role and a functional role are that, firstly, context roles read and write from the FTS rather than passing a message to a functional contract. Secondly, the context role defines acquisition and provisioning regimes to either pull or push context information to the partner context provider/consumer

service. These acquisition/provisioning regimes may be either periodic (e.g. update this context fact every 30 s) or event driven (e.g. notify this context consumer when this fact changes). It should be noted however that a single role can have both functional and context aspects given that its player may be sending/receiving both functional and context messages.

While it is possible to do some simple reasoning to derive/infer context using the rule mechanisms within the composite, a better strategy in terms of maintaining a clear separation of concerns and modularity of design is to externalise the reasoning about context to a separate computational entity/ service that is attached to the composite. Such entities subscribe to context facts using a standard context role, infer further facts from this information and return this derived context to the composite. Such inference might be as simple as calculating statistical information from facts obtained. More complex inference mechanisms using ontologies might also be implemented in such external entities, for example, inferring the situation (i.e. domain context) of a user based on facts about their interactions over the composite[3]. If this external entity is itself implemented as a ROAD composite then it can aggregate context information from multiple external sources.

5.5.5 *Engineering CAAS Applications Using the ROAD Framework*

Our approach has two main phases: development and runtime adaptation. The development phase is illustrated in Fig. 5.6. The service requirements are used for designing the service model using the ROADdesigner Eclipse plugin. The design transformed to an XML document and rules files following the ROAD schema. This model captures the service's functionality, context, and adaptive behavior. The service model is then transformed to an executable service using the ROADfactory component in ROAD4WS (Kapuruge 2011b; Kapuruge et al. 2013b). In particular, the generated runtime artifacts of the executable service are engineered to change at runtime (Step 2).

In the second phase, if there is a need to make unanticipated changes at runtime then the service's runtime model is adapted (Step 3). The differences between the running service's model and its adapted model are then computed. These differences are then used to generate a set of adaptation actions which are applied to the running composite service (Step 4). A more detailed description of this dynamic adaption process can be found in (Hussein et al. 2013).

The ROAD framework has applied in a number domains including adaptive business processes (Kapuruge et al. 2013b Kapuruge et al. 2011), personalised mobile phone call handling based on social context information obtained from social networks (Kabir et al. 2012) (see Chapter 19 of this book), context-aware access control

[3] See Chapter 19 "Socially-aware applications" for an example of such an approach.

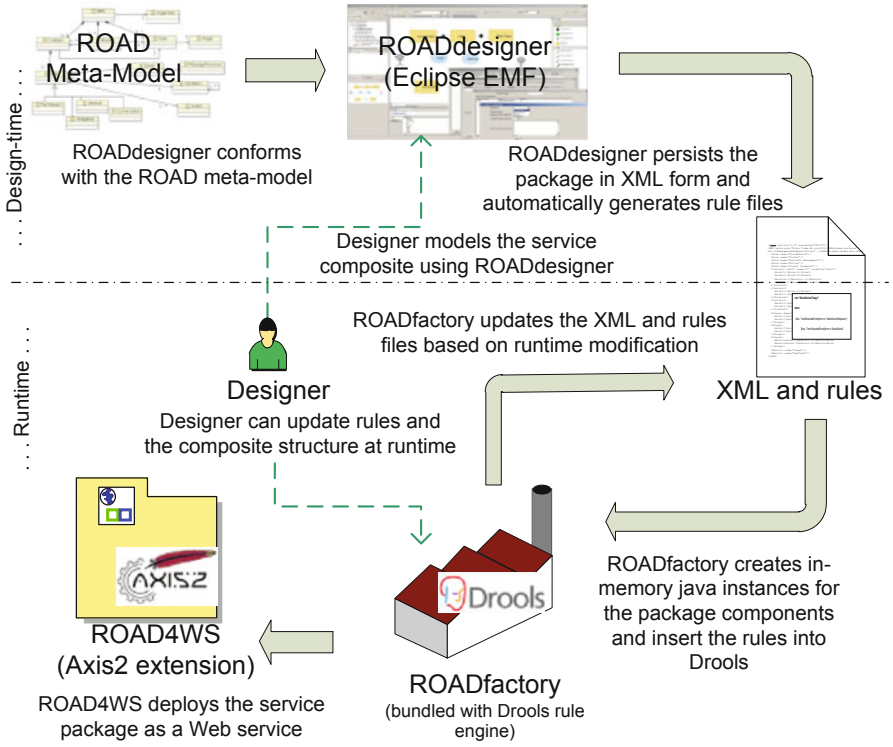

Fig. 5.6 ROAD framework tool chain

(Kayes et al. 2014), and multi-tenanted cloud applications (Kumara et al. 2013). Domain specific evaluations can be found in the above.

5.6 Related Work

A number of approaches support the development of context-aware adaptive software systems from self-adaptive and context-aware perspectives. In this section, we briefly analyze approaches in relation to the requirements we have identified.

Separation of Concerns Existing approaches follow one of two ways for system modeling. Some separate system functionality from management but consider the context representation implicitly as found in self-adaptive systems research (Salehie and Tahvildari 2009). Other approaches have an explicit context representation but hard-code the system management with its functionality, as found in context-aware systems research (Baldauf et al. 2007). As such, they limit the system's runtime adaptation capability. In our approach, we separate the three aspects and keep them integrated from modeling to implementation and to runtime execution by capturing

the system-context relationships explicitly (see Section 5). As such, we can clearly capture and manage the system model, the context model, and their relationships.

Runtime Changes of the Context Model The context model needs to be changed at runtime to cope with unanticipated context changes such as new context information or changes in the number of context element instances unknown at design time, and to reduce the monitoring overhead by only selecting the context model elements that are needed by the functional system. Most of the existing approaches have only a design time context model (e.g. Bettini et al. 2010), and even those approaches that have a runtime context representation do not provide a method for dynamically managing the context model (Taconet et al. 2009; Sheng et al. 2009; Reichle et al. 2008). This makes it more difficult to deal with unanticipated runtime context changes. For example, in the MUSIC project (Rouvoy et al. 2009), the context model elements are represented at runtime and when an element is needed it is activated. But, they do not provide a method of managing the context model at the application level. Our approach has a runtime representation of the context model and its management (i.e. the organizer interface introduced in Section 5.1) enables its runtime changes.

Two Types of Contexts There are two types of context information that need to be considered: (1) the *domain* context, which is the environment information that affects the system operation; (2) the *execution* context, which is the system states that the system management needs to know to initiate the adaptation process if needed. Current research considers either the domain context (Henricksen and Indulska 2004; Sheng et al. 2009; Gu et al. 2005; Mohyeldin et al. 2005; Serral et al. 2010), or the *execution* context (Garlan et al. 2004; Rouvoy et al. 2009). Our approach handles both in a generic and consistent way (see Section 5.2).

System-context Relationships They can be classified into (1) operational relationships, where the system needs to know certain facts about its context to continue its operation; (2) management relationships, where the system needs to adapt itself in response to the context changes. Most of existing approaches consider these relationships implicitly (e.g. Garlan et al. 2004; Morin et al. 2009). Existing approaches do not maintain a runtime representation of the system-context relationships, and as such they cannot be changed at runtime. In our approach, we represent the two types of relationships explicitly and separately (as discussed in Sect. 5.1). Furthermore, we have a runtime representation of these relationships to enable their runtime change.

System realization Many adaptive architectural approaches are based on dynamic component models that explicitly connect the required and provided functional interfaces of component (e.g. Acme Garlan et al. 1997, Darwin Magee et al. 1995). Any process is implicit in the behaviour of those components. In contrast, the approach described here provides an added level of indirection and mediation to the service composition. The downside of this mediated approach is that it requires message transformation that may be inherently more inefficient. The upside is that it allows a much greater the degree of flexibility in the definition of process and allows arbitrary mediators to be defined. In the case of ROAD this allows not only context and other business domain rules to be readily injected into the composition, but context acquisition and provisioning to be dynamically altered.

5.7 Conclusion

ROAD is an adaptive service composition framework that readily enables the incorporation of context information to facilitate both functional and management operations. It does this by providing mechanisms to acquire, record and provision a central store of 'facts' which are evaluated in rules that mediate operational messages or result in adaptations to the behavioral structure of the composite. These facts can include facts about the composite's domain and execution context.

Facts can be sourced either internally or from external context providers. The framework provides a standard way to generate role interfaces from declarative descriptions that can be injected dynamically into the composite. These interfaces can be for functional, context or management services. It also provides a way to inject new fact types and rules to adapt the behavior of the composite at run time.

This approach assists in the development of CAAS applications that integrate the explicit/sophisticated/separate context models of context aware system with the ability of adaptive systems to manage unanticipated change in their environments and requirements.

References

Baldauf, M. et al.: A survey on context-aware systems. Int. J. Ad. Hoc. Ubiquitous. Comput. **2**, 263–277 (2007)

Bencomo, N.: On the use of software models during software execution. In: Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering, pp. 62–67. IEEE Computer Society, Washington DC (2009)

Bettini, C. et al.: A survey of context modelling and reasoning techniques. Pervasive Mob. Comput. **6**, 161–180 (2010)

Bradbury, J.S. et al.: A survey of self-management in dynamic software architecture specifications. In: Proceedings of the 1st ACM SIGSOFT Workshop on Self-Managed Systems, Newport Beach, CA. ACM, New York (2004)

Chen, H., Finin, T., Joshi A.: An ontology for context-aware pervasive computing environments. Knowl. Eng. Rev. **18**(3), 197–207 (2003)

Cheng, B.H.C. et al.: Software engineering for self-adaptive systems: a research road map. In: Software Engineering for Self-Adaptive Systems. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2008)

Colman, A.: Exogenous management in autonomic service compositions. In: Proceedings of the Third International Conference on Autonomic and Autonomous Systems 2007 (ICAS 2007). IEEE Computer Society Press, Athens (2007)

Colman, A., Han J.: On the autonomy of software entities and modes of organisation. In: Proceedings of the 1st International Workshop on Coordination and Organisation (CoOrg 2005), Namur, Belgium (2005)

Colman, A., Han J.: Using Role-based Coordination to Achieve Software Adaptability. Sci. Comput. Program. **64**(2), pp. 223–245 (2007)

Dey, A.K., Abowd G.D., Salber D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Hum. Comput. Interact. **16**(2–4), 97–166 (2001)

Garlan, D., Monroe R., Wile D.: Acme: an architecture description interchange language. In: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research, p. 7. IBM Press, Toronto (1997)

Garlan, D. et al.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer **37**, 46–54 (2004)

Gu T.: A service-oriented middleware for building context-aware services. J. Netw. Comput. Appl. **28**, 1–18 (2005)

Henricksen, K., Indulska J.: A software engineering framework for context-aware pervasive computing. In: The Second IEEE Annual Conference on Pervasive Computing and Communications (PERCOM 2004). IEEE Press, New York (2004)

Hussein, M., Han, J., Yu, J., Colman, A.: An approach to model-based development of context-aware adaptive systems. In: Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference (2011)

Hussein, M., Han, J., Yu, J.; Colman, A.: Enabling runtime evolution of context-aware adaptive services. In 10th International Conference on Services Computing (SCC), pp. 248–255. IEEE, Santa Clara (2013)

Kabir, M.A., Han, J., Yu, J., Colman, A.: SCIMS: a social context information management system for socially-aware applications. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (ed.) Advanced Information Systems Engineering. Lecture Notes in Computer Science, vol. 7328, pp. 301–317 (2012)

Kapuruge, M., Han, J., Colman, A. "Service Orchestration as Organization;; Morgan Kaufmann (2014)

Kapuruge, M., Colman A., Han J.: Achieving multi-tenanted business processes in SaaS applications. In: Web Information System Engineering (WISE). Springer, Sydney (2011a)

Kapuruge, M., Colman A., King J.: ROAD4WS—extending Apache Axis2 for adaptive service compositions. In: IEEE International Conference on Enterprise Distributed Object Computing (EDOC). IEEE Computer Soc. Helsinki (2011b)

Kapuruge, M., Han J., Colman A.: Representing service-relationships as first class entities in service orchestrations. In: International Conference on Web Information System Engineering (WISE), Cyprus. Springer, Berlin (2012)

Kapuruge, M. et al.: Enabling ad-hoc adaptations through event-driven task decoupling. In: International Conference on Advanced Information Systems Engineering (CAiSE). Springer, Valencia (2013a)

Kapuruge, M. et al.: ROAD4SaaS: scalable business service-based SaaS Applications, In: Salinesi, C., Norrie, M., Pastor, Ó. (eds.), Advanced Information Systems Engineering, pp. 338–352. Springer, Berlin (2013b)

Kayes, A.S.M., Jun H., Colman A.: A context-aware access control framework for software services. In: Alessio Lomuscio S.N., Patrizi F., Benatallah B., Brandi I. (eds.) Lecture Notes in Computer Science: International Conference on Service-oriented Computing (ICSOC), pp. 569–577. Springer, Cham (2014)

Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer. **36**(1), 41–50 (2003)

King, J. Colman, A: A multi faceted management interface for web services. Australian software engineering conference, pp. 191–199. IEEE Computer Society, Los Alamitos (2009)

Kumara, I. et al.: Sharing with a Difference: realizing service-based SaaS applications with runtime sharing and variation in dynamic software product lines. In: Proceedings of the 2013 IEEE International Conference on Services Computing, pp. 567–574. IEEE Computer Society, Los Alamitos (2013)

Magee, J., Kramer, J.: Dynamic structure in software architectures. In: Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 3–14. ACM Press, San Francisco (1996)

Magee, J. et al.: Specifying distributed software architectures. Software Engineering—ESEC'95, pp. 137–153. Springer, London (1995)

Mannaert, H., Jan, V., Kris V, Towards evolvable software architectures based on systems theoretic stability. Softw. Pract. Exp. **42**(1) 89–116 (2012)

Mohyeldin, E. et. al.: A generic framework for context aware and adaptation behaviour of reconfigurable systems. In: Personal, Indoor and Mobile Radio Communications (PIMRC), pp. 1957–1963. IEEE, Piscataway (2005)

Morin, B. et. al.: Taming dynamically adaptive systems using models and aspects. In: Proceedings of the 31st International Conference on Software Engineering (2009)

Nierstrasz, O., Denker M., Renggli L.: Model-centric, context-aware software adaptation. Softw. Eng. Self-Adapt. Syst. **5525**, 128–145 (2009)

Patikirikorala, T. et al.: A systematic survey on the design of self-adaptive software systems using control engineering approaches. In: Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Zurich, Switzerland, 04–05 June 2012, pp. 33–42 (2012)

Ranganathan, A. et al.: Use of Ontologies in a Pervasive Computing Environment. Knowl. Eng. Rev. **18**(3), 209–220 (2003)

Reichle, R. et. al.: A comprehensive context modeling framework for pervasive computing systems. In: Meier, R., Terzis, S. (eds.) Distributed Applications and Interoperable Systems. Lecture Notes in Computer Science, vol. **5053**, pp. 281–295. (2008)

Rouvoy, R.: MUSIC: middleware support for self-adaptation in ubiquitous and service-oriented environments. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems, pp. 164–182. Springer, Berlin (2009)

Salehie, M., Tahvildari, L.: Self-adaptive software: landscape and landscape and research challenges. ACM Trans. Auton. Adapt. Syst. **4**, 1–42 (2009)

Serral, E. et al.: Towards the model driven development of contextaware pervasive systems. Pervasive Mob. Comput. **6**, 254–280 (2010)

Sheng, Q.Z. et al.: ContextServ: A platform for rapid and flexible development of context-aware Web services. In: Proceedings of the 31st International Conference on Software Engineering. (2009)

Taconet, C., et al.: CA3M: a runtime model and a middleware for dynamic context management. In: Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE. Vilamoura, Portugal (2009)

Talib, M.A. et al.: A service packaging platform for delivering services. In: IEEE International Conference on Services Computing (SCC), pp. 202–209. IEEE Computer Society, Los Alamitos. (2010)

Wang, X.H. et al.: Ontology based context modeling and reasoning using OWL. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW'04). IEEE, Orlando (2004)