

# Adaptive Embedded Services for Pervasive Computing

Thomas Cottenier, Tzilla Elrad  
Illinois Institute of Technology  
{cotttho, elrad}@iit.edu

**Abstract.** Today's embedded systems are pervasive, mobile, and in many cases, permanently connected. Yet, they are tightly integrated with their environment and barely coordinated. The next generation of distributed applications requires more decentralized and more dynamic interaction schemes, which the classic request/response communication paradigm can not accommodate.

The Executable Choreography Framework (ECF) is a middleware-level platform extension that enables on-demand deployment of peer-to-peer interactions between services and resources. The ECF platform extension combines transparent context propagation with non-invasive software composition techniques to dynamically refine the default control and data flow of service invocations.

The ECF provides a ground for experimentation with dynamic and distributed workflows, and a common specification language for defining advanced service composition schemes, such as adaptive choreographies, mobile agents and distributed Aspect-Oriented compositions in service-oriented environments.

## 1. Introduction

Today's embedded systems are pervasive, mobile, and in many cases, permanently connected to the network. Nevertheless, systems work in tightly integrated environments, where collaborative behavior between devices, if it exists at all, is anticipated before hand, and is built-in to the functionality of the devices. The interactions between components are barely reconfigurable. The pervasive computing vision requires more flexible and intelligent system coordination mechanisms to penetrate the area of embedded and distributed systems. It requires mechanisms and foundations that advance embedded distributed applications towards more decentralized and more dynamic interaction schemes, which the classic request/response communication paradigm can not accommodate. Device-to-device interactions might cross organization boundaries and thus, necessitates seamless coordination between domain controllers. Hence, sound mechanisms for establishing short-term interactions across organizational boundaries are of significant importance to the supporting middleware platforms.

Classic middleware systems such as CORBA bind systems and processes of several organizations into closely coordinated virtual organizations. Deployment must be coordinated across all participants. In practice, despite CORBA's cross-platform abilities, interoperability is restricted to low level services. Services such as security and transaction management are ORB-specific and depend on closely administered environment. They can therefore not accommodate open, ubiquitous environments, where all devices can potentially interact with each other, to provide added value services to the user.

## 2. Embedded Services

Web Services have been introduced to enable distributed applications to be deployed across organizations. Loose coupling is obtained by limiting the complexity of the service interfaces, which only encode generic semantics. All application-specific semantics have to be encoded in descriptive messages. Messages are written in an open standard format that is understood by all parties, generally XML. The ubiquitous nature of XML allows any Web Service-enabled device to be accessed and steered by any XML capable application connected to the network. The Web Service effort is heading towards standardizing the specifications of higher level middleware services such as security and transaction management so they can be deployed across heterogeneous domains and used to compose services and resources that are provided by different organization into higher level, added value applications.

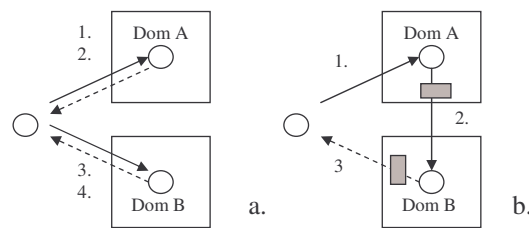
Nevertheless, the use of Web services in embedded systems remains challenging. It requires web service technologies that do not require applications to be integrated within a Web server. Despite the drawbacks of Web Services, such as the verbosity of XML, the industry is actively developing lightweight web service platforms that target embedded devices, such as the IBM Web Services Tool Kit for Mobile Devices [14] or the SUN J2ME Web Services client API [15].

### 3. Dynamic and Decentralized Service Composition

Current service composition mechanisms are limited in their flexibility. Simple composite services are obtained through service *orchestration*. Orchestration is specified in languages such as the Business Process Execution Language (BPEL) for Web Services [12]. Orchestration engines control the message exchanges between an organization and its partners. They are suited to implement centralized *value chain* relationships with long term partners.

Decentralized compositions can be implemented by coordinating BPEL processes running on different organizational domains, through a *choreography* description. Choreographies describe *value network* relationships with long term partners. Yet, Current choreography languages such as the Choreography Description Language [11] do not support dynamic and transient relationships, because their deployment must be coordinated and controlled across all participants.

On-demand resource connectivity and service composition requires the ability to refine the default control and data flow of service invocations.



**Figure 1. Centralized vs Decentralized Composition**

Fig 1.a depicts a simple centralized composition, where 2 services A and B are invoked sequentially. An orchestration engine on the client side invokes the services using request/response based interactions. The invoked services do not have any knowledge about the orchestration they participate in. While this solution is very simple, it might not be optimal performance-wise. A centralized composition engine cannot always take full advantage of the parallelism in its message exchanges and the engine can become a performance bottleneck [1].

Fig 1.b illustrates a decentralized solution to the same composition. In this case, the client entity needs to distribute knowledge about the composition process over the domains. The control flow of the service execution depends on the activity the service invocation participates in. The domains therefore need the ability to discriminate request messages based on the composite activity they are part of. Composition logic then needs to be activated in reaction to some events of interest. In the case of the example, the response of service A needs to be redirected to service B, when a request occurs in the context of the composite activity.

The composition logic can be distributed over the domains in two ways. It can either be dynamically deployed on the domains before the composite service is invoked, or it can be piggybacked in the headers of the request messages and interpreted on the fly by the domains. The first approach is suitable when the interaction pattern is potentially reusable by other composite activities or if the composite service is to be exposed and its endpoint published. The second approach relates to Mobile Agents in the sense that the control logic migrates across domains, along with the service invocations. It is suited for context-specific transient compositions, or to refine existing compositions to context-specific needs. The two approaches are therefore complementary.

While the decentralized solutions provide more flexibility, domain controllers are reluctant to adopt mechanisms that allow peer-to-peer interactions to be deployed when needed because they don't want to execute control flow logic of processes that are initiated and steered by other entities. The author's position is that a better tradeoff between flexibility/expressiveness and control/safety can be found. Yet, flexible control delegation requires investigating new approaches to composition that go beyond current technologies.

## 4. Executable Choreography Framework

The Executable Choreography Framework (ECF) [5][6][7] introduces a language to specify executable choreographies and a platform extension to enable the deployment of executable choreographies on application servers.

### 4.1. Executable Choreography Language

The ECF enables the default flow of control of a service invocation to be refined in a context-sensitive way. Refinements are applied non-invasively – without having to manually modify the workflow of the target service. The Executable Choreography Language (ECL) is a XML-based language to define refinements on the default control flow of service invocation and execution. Given a choreography description, either provided as an activity diagram or a CDL specification, the ECF partitions the distributed workflow into ECL rules. ECL rules specify a set of actions to be performed when a message of interest is intercepted in the container. For clarity purposes, XML ECL rules are represented in a graphical notation, as shown in Fig. 2.

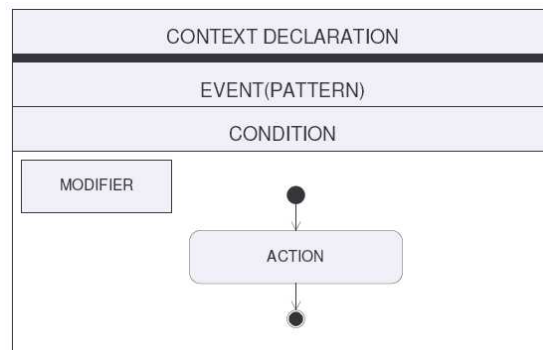


Fig. 2. Graphical representation of an ECL Rule

An ECL rule is similar to Event-Condition-Action (ECA) rule used in active databases [2]. It is composed of an activity context declaration, an event pattern, a set of conditions and an action:

1. **CONTEXT:** An Activity Context declaration uniquely identifies the rule. At runtime, the context encapsulates information that relates to a distributed activity. Activity contexts are propagated transparently from node to node, along the interactions of an activity.
2. **EVENT:** An Event expression defines when the rule should be applied. The ECF recognizes two types of events: sending a message and receiving a message. An event expression defines a pattern on the signature (PortType, Operation, Parameter types) of the messages to be intercepted.
3. **CONDITION:** The condition clauses identify the activities for which the refinement of the control flow should be applied. Interactions of an activity are discriminated based on their activity context information. Condition clauses take the form: `CONTEXT(<Activity-Context-Identifier>)`
4. **MODIFIER:** A modifier specifies when the rule behavior should be applied. The rule actions can execute either Before, Around (Instead) or After the events captured by the event pattern.
5. **ACTION:** The ECF takes control over the thread of execution of the service invocation or execution. The actions authorized by the ECF are:
  - Compound Action: Actions can be composed into compound actions, using the sequence, fork, join, decision and merge control flow operators.
  - Accept Request/Response Action: wait for a request or a response message from a remote service.
  - Send Request/Response Message Action: send a service request or response.
  - Invoke Service Action: invoke the functionality of a local service
  - Set Timer Action, Reset Timer Action and Accept Time Event Action: timing actions allow ECL rules to define timeouts and handle faults.
  - Data Mapping Action: The SOAP messages intercepted by the ECF can be transformed according to a XSL specification. SOAP messages can be aggregated and data can be consolidated.

## 4.2. Executable Choreography Platform Extension

ECF actions have unambiguous implementations in standard Application Servers. ECF-enabled platforms can interpret ECL rules, and deploy the corresponding control flow logic accordingly, in the native language of the platform. The ECF platform extension implements 3 distinct functionalities: message interception, transparent activity context propagation and dynamic rule deployment.

### 4.2.1. Message Interception

Incoming SOAP messages are intercepted before they are dispatched to a service provider. Likewise, outgoing messages are intercepted after serialization, before they flow out of the container. When a message matches the event pattern of an ECL rule within its activity context, the ECF platform extension takes control over the thread of the service request or response, and injects the rule behavior, before, after or instead of the intercepted event.

### 4.2.2. Activity Context Propagation

The ECF provides transparent context propagation within a distributed activity. Activity contexts are piggybacked in the headers of the intercepted messages. The ECF platform extension ensures that contexts are propagated from node to node, along the interactions of a same distributed activity. Context propagation helps managing the life-cycle of distributed activities. The ECF concept of activity context is derived from the Composite Application Framework (WS-CAF) specification [13]. WS-CAF is a standard to implement context-passing, coordination and transaction management in web-service based composite applications. As opposed the WS-CAF, the ECF propagates context transparently.

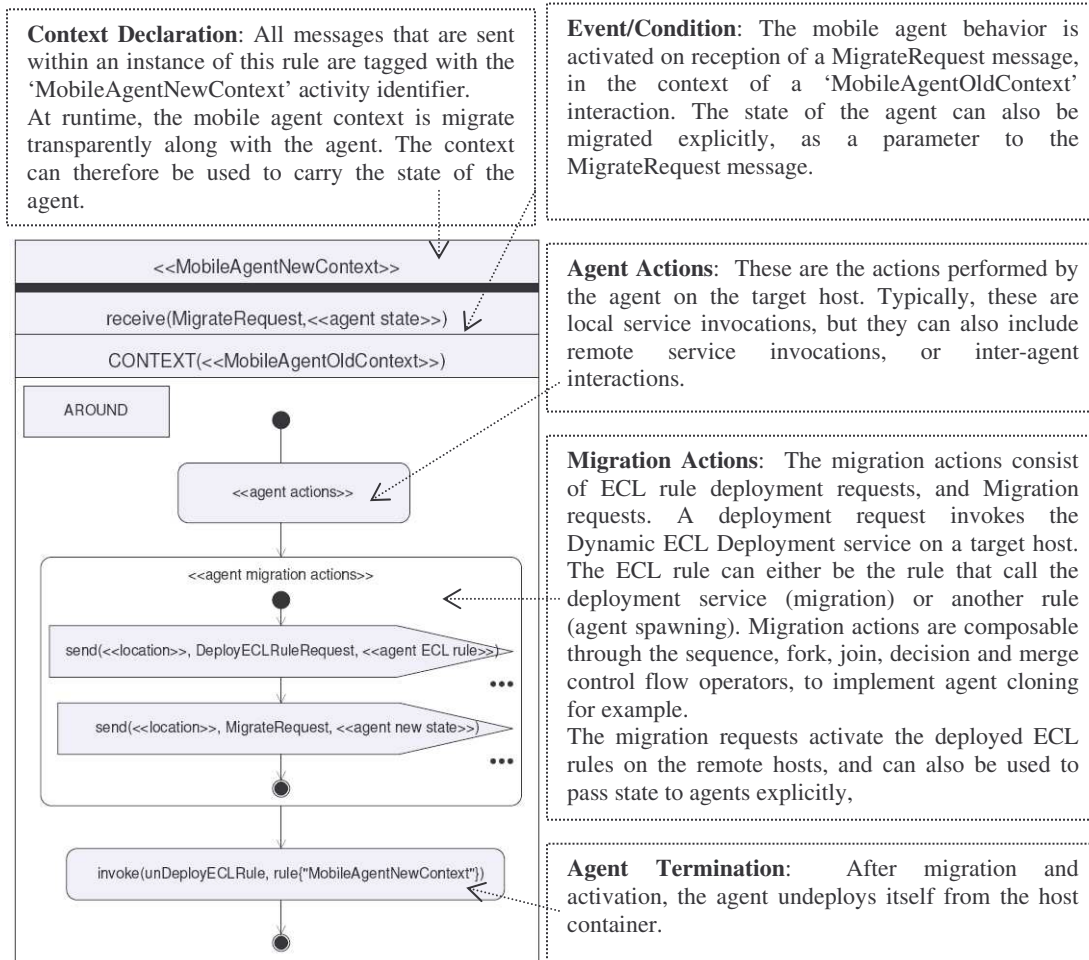


Fig.3. ECL rule pattern for simple mobile agents

### 3.2.3. Dynamic Deployment

Given a choreography description, the ECF partitions the distributed workflow into ECL rules. These rules can be deployed on remote containers on-demand. ECF-enabled platforms expose a choreography deployment Web Service, whose endpoint is published into a choreography repository.

## 5. Mobile Agent Implementation with the ECL

Simple Mobile agents are straightforward to implement with ECL rules. ECL rules conform to a XML Schema and can be interpreted by any ECF-enabled containers. Agent behavior can therefore be specified in a platform independent way, which is a fundamental requirement for implementing agents in web service environments. Most other proposals the authors are aware of [8][9][10] implement mobile agents for Web Service environments in a platform specific language such as java.

A simple mobile agent typically performs a series of invocations of local services according to some control flow logic and aggregates the result data. Once it performed its local operations, it migrates to another host, along with its control logic and data.

In the ECL, an agent is a regular rule which has the ability to deploy it self or other rules on a target host. Deploying a rule on a remote host is performed by invoking the Dynamic Deployment web service of the ECF. Fig3. presents and describes an ECL rule pattern for simple mobile agents.

## 6. Distributed Aspect-Oriented Programming with ECF

The ECF provides the basic building blocks for a distributed Aspect-Oriented Programming platform for Web-Service environments. In the Web Service context, the implementations of many of the middleware-level services such as transaction management or security are tightly coupled to the implementation of specific applications. These concerns are hard to cleanly modularize into separate Web Services, because they affect composite applications at many locations of their workflow specification.

Fig 10 illustrates a simple aspect that guides the behavior of agents when communication failures occur. Whenever an ECL rule deployment request or a migration request fails in the context of the 'FindRouteAgent' the aspect forces the agent to wait for 5 minutes, and try again. This type of behavior allows agents to keep functioning in environments where the network connection is unreliable, or offers irregular bandwidth.

Within the ECF, Aspect can be composed of multiple ECL rules that intercept many distributed events. Aspects can also intercept activities, as opposed to Send or Receive actions. An activity is spawned on one host, and may terminate on another host. An ECF choreography Aspect can inject behavior before, after or around an Activity. Distributed AOP is suited to implement Middleware-level services such as those provided by the Composite Application Framework (WS-CAF) specification.

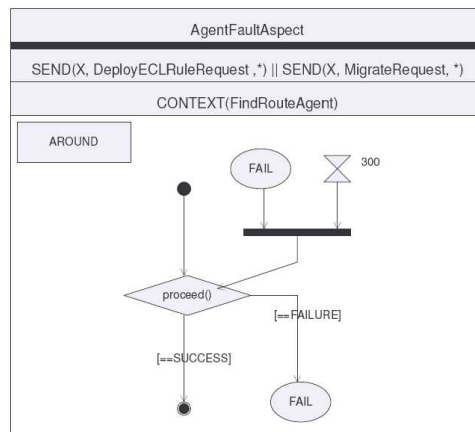


Fig. 4. An ECL rule for a simple Fault tolerance aspect for Mobile Agents

## 7. Conclusions

While Embedded and Distributed systems are pervasive and permanently connected, they are tightly integrated with their environment and poorly coordinated. The next generation of distributed applications requires more decentralized and more dynamic interaction schemes, which the classic request/response communication paradigm can hardly accommodate.

Despite the drawbacks of Web Services, such as the verbosity of XML, the industry is actively developing lightweight web service platforms that target embedded devices. Yet, current service composition mechanisms are limited in their flexibility. On-demand resource connectivity and service composition requires the ability to dynamically refine the default control and data flow of service invocations.

The Executable Choreography Framework (ECF) is a middleware-level framework that targets dynamic and decentralized service compositions. The ECF combines transparent context propagation with aspect-oriented software composition techniques to concisely encapsulate workflow adaptation policies.

The ECF provides a ground for experimentation with dynamic and distributed workflows, and a common specification language for defining advanced service composition schemes, such as adaptive choreographies, mobile agents and distributed Aspect-Oriented compositions in web service environments.

## Acknowledgement

This work is partially supported by CISE NSF grant No. 0137743.

## References

- [1] Chafle, G., Chandra, S., Mann, V., Nanda, M. G.: Decentralized Orchestration of Composite Web Services. Proceedings of the Thirteenth International World Wide Web Conference, New York, NY, USA, ACM Press (2004)
- [2] Norman W. Paton, Oscar Diaz, Active Database Systems, ACM Computing Surveys, New York, NY, USA, ACM Press (1999)
- [3] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. Proceedings of the European Conference on Object-Oriented Programming, Springer-Verlag (1997)
- [4] Filman, R., Friedman, D.: Aspect-oriented Programming is Quantification and Obliviousness. Workshop on Advanced Separation of Concerns, OOPSLA 2000 (2000)
- [5] Cottenier T., Elrad T, Prunicki, A.: Contextual Aspect-Sensitive Services, formal demonstration presented at the 4th International conference on Aspect-Oriented Software Development (AOSD'05), Chicago, USA (2005)
- [6] Cottenier, T., Elrad, T.: Dynamic and Decentralized Service Composition with Contextual Aspect-Sensitive Services, First International Conference on Web Information Systems and Technologies, Miami, USA (2005)
- [7] Cottenier, T., Elrad, T.: Validation of Aspect-Oriented Adaptations to Components. Ninth International Workshop on Component-Oriented Programming as part of ECOOP'04, Oslo, Norway (2004)
- [8] Zakaria Maamar, Quan Z. Sheng, and Boualem Benatallah. Interleaving web services composition and execution using software agents and delegation. In AAMAS'2003 Workshop on Web Services and Agent-based Engineering (2003)
- [9] Amir Padovitz, Shonali Krishnaswamy, and Seng Wai Loke. Toward efficient and smart selection of web service. In AAMAS'2003 Workshop on Web Services and Agent-based Engineering (2003).
- [10] Fuyuki Ishikawa, Nobukazu Yoshioka, Yasuyuki Tahara, and Shinichi Honiden, Toward Synthesis of Web Services and Mobile Agents, AAMAS'2004 Workshop on Web Services and Agent-based Engineering (2004)
- [11] Web Services Choreography Description Language (WS-CDL) Version 1.0, W3C Working Draft 17, <http://www.w3.org/TR/ws-cdl-10/> (2004)
- [12] BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems, Business Process Execution Language for Web Services specification  
<http://www-128.ibm.com/developerworks/library/ws-bpel>
- [13] Arjuna Technologies Ltd., Fujitsu Limited, IONA Technologies Ltd., Oracle Corporation, and Sun Microsystems, Inc, Web Services Composite Application Framework (WS-CAF) specification  
<http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf> (2003)
- [14] IBM Web Services Tool Kit for Mobile Devices homepage  
<http://www.alphaworks.ibm.com/tech/wstkMD>
- [15] SUN J2ME Web Services APIs (WSA, JSR 172) homepage  
<http://java.sun.com/products/wsa/>