An Active Architecture Approach to Dynamic Systems Co-evolution

Ron Morrison[1], Dharini Balasubramaniam[1], Flavio Oquendo[2], Brian Warboys[3], and R. Mark Greenwood[3]

[1] University of St Andrews
St Andrews, KY16 9SX, UK
{ron,dharini}@cs.st-andrews.ac.uk
[2] University of South Brittany – Valoria
BP 573, 56017 Vannes Cedex, France
Flavio.Oquendo@univ-ubs.fr
[3] University of Manchester, Manchester, M13 9PL, UK
{brian,markg}@cs.man.ac.uk

Abstract. The term *co-evolution* describes the symbiotic relationship between dynamically changing business environments and the software that supports them. Business changes create pressures on the software to evolve, and at the same time technology changes create pressures on the business to evolve. More generally, we are concerned with systems where it is neither economically nor technologically feasible to suspend the operation of the system while it is being evolved. Typically these are long-lived systems in which *dynamic* co-evolution, whereby a system evolves as part of its own execution in reaction to both predicted and emergent events, is the only feasible option for change. Examples of such systems include continuously running business process models, sensor nets, grid applications, self-adapting/tuning systems, routing systems, control systems, autonomic systems, and pervasive computing applications.

Active architectures address both the structural and behavioural requirements of dynamic co-evolving software by modelling software architecture as part of the on-going computation, thereby allowing evolution during execution and formal checking that desired system properties are preserved through evolution. This invited paper presents results on active architectures from the Compliant System Architecture and ArchWare projects. We have designed and constructed the ArchWare-ADL, a formal, well-founded architecture description language, based on the higher-order typed π-calculus, which consists of a set of layers to address the requirements of active architectures. The ArchWare-ADL design principles, concepts and formal notations are presented together with its sophisticated reflective technologies for supporting active architectures and thereby dynamic co-evolution.

1 Introduction

Where software cannot change to keep up with the changing business goals, the business loses efficiency, and the perceived value of the software decreases [17].

F. Oquendo (Ed.): ECSA 2007, LNCS 4758, pp. 2 – 10, 2007.

Conversely, a business must continually intercept the potential of new technology to maintain its effectiveness and remain competitive. Thus business changes create pressures on the software to evolve, and at the same time software changes create pressures on the business to evolve [32]. We use the term co-evolution to describe the concept of the business and the software evolving sympathetically, but at potentially different rates, to satisfy changing requirements.

More generally, we are concerned with what Milner termed "wide informatic systems" [21] to describe systems, assembled from components, that co-evolve with their environment. Such systems recognise that the business effects of introducing, or changing, a software system are potentially emergent, especially in terms of non-functional requirements such as safety, reliability and performance. Typically these are long-lived applications in which dynamic co-evolution, whereby a system evolves as part of its own execution, is the only feasible option for change. Examples of such systems include continuously running business process models, sensor nets, grid applications, self-adapting/tuning systems, routing systems, control systems, and pervasive computing applications.

Active architectures address the structural and behavioural requirements of dynamic co-evolving software by modelling the software architecture as part of the on-going computation, thereby allowing evolution during execution. They facilitate the design and engineering of dynamic co-evolving systems in a way that preserves the flexibility to adapt to new requirements while ensuring that important properties can be continuously verified. Fundamentally, users and developers must be able to understand a system, and its interactions with the environment, to define the appropriate evolutions. In our approach active architectures manage the co-evolutionary process using: a formal description of the system's structure and behaviour; self-monitoring capabilities to provide a view of the system's dynamic behaviour; sophisticated evolutionary mechanisms including a model of the system itself at the appropriate level of abstraction and incremental checking tools for formally verifying that desired system properties are preserved over evolution.

We have designed and constructed the ArchWare-ADL [26][27], a formal, well-founded architecture description language based on the higher-order typed π-calculus [19], which consists of a set of layers to address the requirements of active architectures. The ArchWare-ADL design principles, concepts and formal notations are presented along with sophisticated reflective technologies for supporting active architectures and thereby dynamic co-evolution.

2 Active Software Architecture

A key aspect of the design of a software system is its architecture; the fundamental organisation of the system embodied in its components, their relationships to one another and to the environment. Two viewpoints are frequently used in describing software architectures: *structural* and *behavioural* [13].

The structural viewpoint may be specified in terms of:

- components – units of computation of a system,
- connectors – interconnections among components for supporting their interactions, and
- configurations – of components and connectors.

From a structural viewpoint, an architecture description should provide a formal specification of the architecture in terms of components and connectors, how they are composed, and how the architectural structure may change.

The behavioural viewpoint may be specified in terms of:

- actions a system executes or participates in,
- relations among actions to specify behaviours,
- behaviours of components and connectors, and how they interact and change,
- the state of the active system.

We define an active architecture as dynamic in that it can evolve during execution by creating, deleting, reconfiguring and moving components, connectors and their configurations at runtime. Importantly, active architectures require the structural and behavioural viewpoints to specify static and dynamic behoviour and properties. A challenge for an Architecture Description Language (ADL) is to support active architectures by finding the abstractions and technologies that accommodate the dynamic evolution of components, connectors and configurations through their actions, relationships and behaviours.

3 The Archware ADL

The ArchWare-ADL [26] is a novel architecture description language (ADL) designed to accommodate active architectures. It is:

- a formal, theoretically well-founded language based on the higher-order typed π-calculus;
- automated by tools, i.e. a specification and verification toolset providing support for automated checking; and
- supported by a set of reflective technologies for dynamic co-evolution.

The ArchWare-ADL takes its roots in previous work on the use of π-calculus as semantic foundation for architecture description languages [4][5]. When augmented with domain-specific architectural extensions it blends Turing completeness and high architecture expressiveness with a simple formal notation.

The novelty of ArchWare lies in its holistic view of formal development, encompassing languages for formal specification, (i.e. ArchWare ADL), formal verification [18], and formal transformation [28]. As such it goes beyond existing formal methods in supporting architecture-centric model-driven engineering of software systems.

Like formal methods such as B [1], Z [6], VDM [7], and FOCUS [31], ArchWare provides full support for formal description and development of software systems. Unlike these formal methods, ArchWare is based on a process algebra, the higher-order typed π-calculus, and provides architectural support for formal architecture-centric software engineering.

The ArchWare-ADL consists of a set of layers to address the structural and behavioural requirements of active architectures. The novelty is that it models the

architecture as part of the on-going computation, thereby allowing the evolution of the architecture in step with execution.

There are five layers in the ArchWare-ADL:

- The base layer defines a coordination language without data values corresponding to the monadic π-calculus.
- The first order layer adds data values and abstractions and corresponds to polyadic π-calculus.
- The higher order layer corresponds to higher-order polyadic π-calculus.
- The analysis layer enables the specification of constraints on the styles.
- The style layer allows the specification of components and connectors.

The style layer provides constructs from which the base component-and-connector style and other derived styles can be defined. Conceptually, an architectural style provides:

- a set of abstractions for architectural elements;
- a set of constraints (i.e. properties that must be satisfied) on architectural elements, including legal compositions; and,
- a set of additional analyses that can be performed on architecture descriptions constructed in the style.

Thus the ArchWare-ADL provides a framework for formalising software architectures based on the concepts of components and connectors. Styles are used to define families of architectures that have a common structure and satisfy the same properties.

The style layer makes use of both the underlying π-calculus layers and the analysis layer. Styles in ArchWare are defined as property-guarded abstractions that may be applied to yield instances of an architecture conforming to the style. The structure and behaviour of an architecture family are specified using the ArchWare-ADL [11] with the constraints on the family being specified using the analysis layer.

The essential property of dynamic co-evolutionary systems is the ability to (partially) stop and decompose a running system into its constituent components, and compose evolved or new components to form a new system, while preserving any state or shared data if desirable. Once a co-evolving system receives an internal or external stimulus for evolution it can: (partially) suspend the component(s) to be evolved; decompose them into constituent parts; modify the components separately; recompile the components and bind them into the executing system. The support technologies for this are a decomposition operator, linguistic reflection and a system representation capable of capturing closure, called hyper-code [14].

4 Dynamic Co-evolution

A dynamic co-evolving system is constantly in a state of flux as it evolves in reaction to external and internal stimuli. To understand and manage the co-evolutionary process, we introduce the concepts of incarnation, evolutionary step and locus [24].

A dynamic co-evolving system mutates from one incarnation to the next via an evolutionary step. Within a context we can observe a sub-system through a sequence of incarnations. Each incarnation includes a period of normal execution, followed by an

evolutionary step. The end of the evolutionary step defines the start of the next incarnation. Thus an incarnation can be considered as system execution between the end points of two subsequent evolutionary steps and consists of the code, data and meta-data not only for carrying out its present purpose, including architectural constraint checking, but also for a variety of possible future evolutionary steps. The change context, i.e. the set of entities that will change during an evolutionary step, is known as the locus. An evolutionary step is defined as a change to a locus and is made by an evolver internal to the locus. Thus it is the locus that evolves itself, in reaction to some external or internal stimulus. During normal execution, the locus continually monitors the incarnation to determine when an evolutionary step is necessary.

A co-evolving application may be constructed in terms of loci. The internal evolver produces new incarnations of the locus. The structuring of loci within an application is static where it is possible to predict the sets of loci and interactions that are subject to change. In such cases, an evolver produces a new incarnation of its locus and the application changes without changing the structure of the loci. Where it is not possible to predict the sets of loci and interactions that are subject to change, a new locus may be defined dynamically to undertake the desired evolutionary step. Thus the structuring of the loci is dynamic. In both cases the locus determines those factors that change and those that remain constant during the evolutionary step. The objective is to make explicit the distinction between what can change and what remains unaffected by each evolutionary step, even though this may not be completely statically defined.

We offer this initial set of intrinsic requirements for dynamic co-evolution [24]:

- a method for structuring dynamic co-evolving systems as loci;
- a technique for programming with incremental design;
- a method of monitoring the internal and external environment;
- a method of partially stopping and decomposing loci and their interactions;
- a method of dynamically reifying the state of the computation;
- a method of generating new loci using the reified state (IR6);
- and, a method of rebinding the new loci into the executing system.

5 Co-evolving Active Architecture

We have developed the following technologies that satisfy the needs of the intrinsic requirements of adaptation [3][23][33][34]

- **Decomposition:** a decompose operator to suspend execution and break up a locus into its constituent parts.
- **Reification:** a reify operator to yield a dynamic representation of the executing locus.
- **Generation:** a method of transforming the representation to create new representations.
- **Reflection:** a reflect operator to compile the resultant representations and bind them into the running computation [30].
- **Probes:** a software mechanism to provide feed forward and feedback stimuli [2].

- **A persistent** environment
- **Dynamic** property checking

Figure 1 illustrates the use of dynamic property checking to ensure that the execution and evolution of the active architecture preserves the essential properties of the system. The architectural style is expressed in terms on elements and constraints among these elements. These map on to the active architecture. The property checker monitors the execution and dynamically checks properties. When a violation occurs this is fed back to the executing architecture that then may form loci (ellipses) to evolve the architecture.

Importantly, since the property checker is part of the active system, it may itself evolve to check new properties or use new technology.

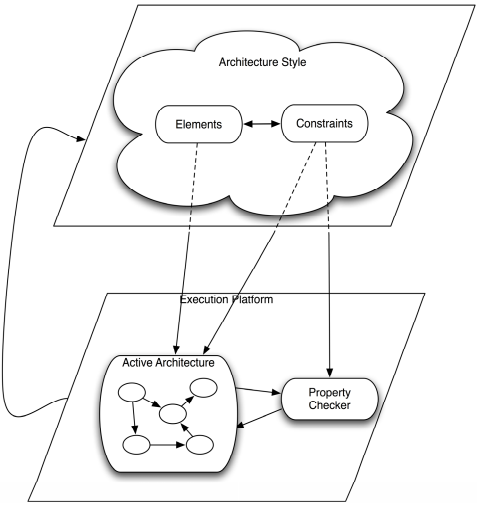

Fig. 1. A Dynamically Co-evolving System

6 Decomposition and Hyper-code

The two most unusual technologies supported by the Archware ADL are decomposition and hyper-code.

Decomposition suspends execution and breaks up a locus into its constituent parts. Unusually the semantics of decomposition are well suited for definition in the π-calculus by allowing each constituent part to run to its reduction limit, that is where it is waiting for an interaction, and then breaking the communication channel and stopping the execution.

Hyper-code is a representation of an active executing graph linking source code and existing values. It allows state and shared data to be preserved during evolution. Thus at any point during the computation the state of the execution may be inspected by viewing the hyper-code.

The importance of hyper-code is that it is rich enough to represent executing code since it captures the state of the computation. Since hyper-code can represent closure, it may be used to introspect an executing system, and thereby can be used to return the result of a decomposition operation. Together the facilities of hyper-code, decomposition, reflection and reification permit users to stop part of an executing system (while the rest of the system continues to execute), inspect its specification and state, evolve the part as necessary, and recompose the system. The generation may be performed by specifying a series of semantics preserving transformations using a programmable interface to the hyper-code graph such as supplied by the Metaprogramming Framework [19].

7 Conclusions

There are many complementary architectural approaches to system evolution [8][9] [10] [12][15][16][25][29][35]. Active architectures address the structural and behavioural requirements of dynamic co-evolving software by modelling the software architecture as part of the on-going computation, thereby allowing it to evolve during execution.

The judicious mixture of formality in the Archware-ADL and its sophisticated support technologies yield an innovative approach to implementing active architectures for dynamic co-evolution. The language can describe the system's specification, the executing software and the reflective evolutionary mechanisms within a single computational domain in which all three may evolve in tandem.

Acknowledgements

The ArchWare Project was partially funded by the Commission of the European Union under contract No. IST-2001-32360 in the IST-V Framework. The work reported here also builds on earlier EPSRC-funded work in Compliant Systems Architectures [22] (GR/M88938 & GR/M88945).

References

[1] Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)

[2] Balasubramaniam, D., Morrison, R., Mickan, K., Kirby, G.N.C., Warboys, B.C., Robertson, I., Snowdon, R., Greenwood, R.M., Seet, W.: Support for feedback and change in self-adaptive systems. In: WOSS'04. Proc. ACM SIGSOFT Workshop on Self-Managing Systems, Newport Beach, CA, USA, pp. 18–22. ACM Press, New York (2004)

[3] Balasubramaniam, D., Morrison, R., Kirby, G.N.C., Mickan, K., Warboys, B.C., Robertson, I., Snowdon, B., Greenwood, R.M., Seet, W.: A software architecture approach for structuring autonomic systems. In: DEAS 2005. Proc. ICSE Workshop on the Design and Evolution of Autonomic Application Software, St Louis, MO, USA, pp. 59–65 (2005)

[4] Chaudet, C., Greenwood, M., Oquendo, F., Warboys, B.: Architecture-Driven Software Engineering: Specifying, Generating, and Evolving Component-Based Software Systems. IEE Journal: Software Engineering 147(6) (December 2000)

[5] Chaudet, C., Oquendo, F.: A Formal Architecture Description Language Based on Process Algebra for Evolving Software Systems. In: ASE'00. Proceedings of the 15th IEEE International Conference on Automated Software Engineering, Grenoble, September 2000, IEEE Computer Society, Los Alamitos (2000)
[6] Davies, J., Woodcock, J.: Using Z: Specification, Refinement and Proof. Prentice Hall International Series in Computer Science (1996)
[7] Fitzgerald, J., Larsen, P.: Modelling Systems: Practical Tools and Techniques for Software Development. Cambridge University Press, Cambridge (1998)
[8] Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-based self-adaptation with reusable infrastructure. IEEE Computer 37(10), 46–54 (2004)
[9] Godfrey, M.W., Tu, Q.: Evolution in Open Source Software: A Case Study. In: ICSM'00. Proceedings of the International Conference on Software Maintenance, Washington, DC, October 11 - 14, 2000, pp. 131–142. IEEE Computer Society, Los Alamitos (2000)
[10] Gorlick, M.M., Razouk, R.R.: Using Weaves for Software Construction and Analysis. In: Proc. 13th International Conference on Software Engineering, Austin, Texas, United States, pp. 23–34. IEEE Computer Society Press, Los Alamitos (1991)
[11] Greenwood, M., Balasubramaniam, D., Cimpan, S., Kirby, N.C., Mickan, K., Morrison, R., Oquendo, F., Robertson, I., Seet, W., Snowdon, R., Warboys, B., Zirintsis, E.: Process Support for Evolving Active Architectures. In: Oquendo, F. (ed.) EWSPT 2003. LNCS, vol. 2786, Springer, Heidelberg (2003)
[12] Groenewegen, L.P.J., de Vink, E.P.: Evolution-On-The-Fly with Paradigm. In: Ciancarini, P., Wiklicky, H. (eds.) COORDINATION 2006. LNCS, vol. 4038, pp. 97–122. Springer, Heidelberg (2006)
[13] IEEE Std 1471-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (October 2000)
[14] Kirby, G.N.C., Connor, R.C.H., Cutts, Q.I., Dearle, A., Farkas, A.M., Morrison, R.: Persistent hyper-programs. In: Albano, A., Morrison, R. (eds.) Persistent Object Systems, 1992. Proc. 5th International Conference on Persistent Object Systems, Italy, pp. 86–106. Springer, Heidelberg (1993)
[15] Kramer, J., Magee, J.: The Evolving Philosophers Problem: Dynamic Change Management. IEEE Trans. on Software Engineering 16(11), 1293–1306 (1990)
[16] Kramer, J., Magee, J.: Analysing Dynamic Change in Software Architectures: A Case Study. In: CDS 98. Proc. IEEE 4th Int. Conference on Configurable Distributed Systems, Annapolis, USA, May 1998, pp. 91–100. IEEE Computer Society Press, Los Alamitos (1998)
[17] Lehman, M.M.: Laws of software evolution revisited. In: Montangero, C. (ed.) EWSPT 1996. LNCS, vol. 1149, pp. 108–124. Springer, Heidelberg (1996)
[18] Mateescu, R., Oquendo, F.: pi-AAL: An Architecture Analysis Language for Formally Specifying and Verifying Structural and Behavioural Properties of Software Architectures. ACM Software Engineering Notes 31(2) (March 2006)
[19] Mickan, K.: A Meta-Programming Framework for Software Evolution. Ph.D. Thesis, University of St Andrews (2006)
[20] Milner, R.: Communicating and Mobile Systems: The Pi-Calculus. Cambridge University Press, Cambridge (1999)
[21] Milner, R.: Computing in space. In: 17th International Congress on Computer Assisted Radiology and Surgery (CARS2003) (2003), http://www.cl.cam.ac.uk/users/rm135/
[22] Morrison, R., Balasubramaniam, D., Greenwood, R.M., Kirby, G.N.C., Mayes, K., Munro, D., Warboys, B.C.: A Compliant Persistent Architecture. Software, Practice & Experience 30, 1–24 (2000)
[23] Morrison, R., Kirby, G.N.C., Balasubramaniam, D., Mickan, K., Oquendo, F., Cîmpan, S., Warboys, B.C., Snowdon, B., Greenwood, R.M.: Support for evolving software

architectures in the ArchWare ADL. In: WICSA 4. Proc. 4th Working IEEE/IFIP Conference on Software Architecture, Oslo, Norway, pp. 69–78 (2004)
[24] Morrison, R., Balasubramaniam, D., Kirby, G.N.C., Warboys, B.C., Greenwood, R.M.: A Framework for Supporting Dynamic Systems Co-evolution. Journal of Automated Software Engineering (accepted for publication, 2007)
[25] Oreizy, P., Medvidovic, N., Taylor, R.N.: Architecture-Based Runtime Software Evolution. In: Proc. 20th International Conference on Software Engineering, Kyoto, Japan, pp. 177–186. IEEE Computer Society, Los Alamitos (1998)
[26] Oquendo, F., Warboys, B., Morrison, R., Dindeleux, R., Gallo, F., Garavel, H., Occhipinti, C.: ArchWare: Architecting Evolvable Software. In: Oquendo, F., Warboys, B.C., Morrison, R. (eds.) EWSA 2004. LNCS, vol. 3047, Springer, Heidelberg (2004)
[27] Oquendo, F.: π-ADL: an architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. ACM Software Engineering Notes 29(3) (2004)
[28] Oquendo, F.: pi-ARL: An Architecture Refinement Language for Formally Modelling the Stepwise Refinement of Software Architectures. ACM Software Engineering Notes 29(5) (September 2004)
[29] Schmerl, B., Garlan, D.: Exploiting architectural design knowledge to support self-repairing systems. In: SEKE '02. Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July 15-19, 2002, vol. 27, pp. 241–248. ACM Press, New York (2002)
[30] Stemple, D., Fegaras, L., Stanton, R.B., Sheard, T., Philbrow, P., Cooper, R.L., Atkinson, M.P., Morrison, R., Kirby, G.N.C., Connor, R.C.H., Alagic, S.: Type-safe linguistic reflection: a generator technology. In: Atkinson, M.P., Welland, R. (eds.) Fully Integrated Data Environments, pp. 158–188. Springer, Heidelberg (1999)
[31] Stolen, K., Broy, M.: Specification and Development of Interactive Systems. Springer, Heidelberg (2001)
[32] Warboys, B.C., Kawalek, P., Robertson, I., Greenwood, R.M.: Business Information Systems: A Process Approach. McGraw-Hill, New York (1999)
[33] Warboys, B.C., Balasubramaniam, D., Greenwood, R.M., Kirby, G.N.C., Mayes, K., Morrison, R., Munro, D.S.: Collaboration and Composition: Issues for a Second Generation Process Language. In: Nierstrasz, O., Lemoine, M. (eds.) Software Engineering - ESEC/FSE '99. LNCS, vol. 1687, pp. 75–91. Springer, Heidelberg (1999)
[34] Warboys, B.C., Greenwood, R.M., Robertson, I., Morrison, R., Balasubramaniam, D., Kirby, G.N.C., Mickan, K.: The ArchWare Tower: The Implementation of an Active Software Engineering Environment using a π-calculus based Architecture Description Language. In: Morrison, R., Oquendo, F. (eds.) EWSA 2005. LNCS, vol. 3527, pp. 30–40. Springer, Heidelberg (2005)
[35] Zhang, J., Cheng, B.H.: Model-based development of dynamically adaptive software. In: ICSE '06. Proc. of the 28th international Conference on Software Engineering, Shanghai, China, May 20-28, 2006, pp. 371–380. ACM Press, New York (2006)