# ARCHWARE: Architecting Evolvable Software

Flavio Oquendo[1], Brian Warboys[2], Ron Morrison[3], Régis Dindeleux[4], Ferdinando Gallo[5], Hubert Garavel[6], and Carmen Occhipinti[7]

[1] InterUnec/University of Savoie – France
http://www.arch-ware.org
[2] University of Manchester – UK
[3] University of St Andrews – UK
[4] Thésame – France
[5] Consorzio Pisa Ricerche – Italy
[6] INRIA – France
[7] Engineering Ingegneria Informatica – Italy

**Abstract.** This paper gives an overview of the ArchWare European Project[1]. The broad scope of ArchWare is to respond to the ever-present demand for software systems that are capable of accommodating change over their lifetime, and therefore are evolvable. In order to achieve this goal, ArchWare develops an integrated set of architecture-centric languages and tools for the model-driven engineering of evolvable software systems based on a persistent run-time framework. The ArchWare Integrated Development Environment comprises: (a) innovative formal architecture description, analysis, and refinement languages for describing the architecture of evolvable software systems, verifying their properties and expressing their refinements; (b) tools to support architecture description, analysis, and refinement as well as code generation; (c) enactable processes for supporting model-driven software engineering; (d) a persistent run-time framework including a virtual machine for process enactment. It has been developed using ArchWare itself and is available as Open Source Software.

## 1 Introduction

ArchWare applies an innovative approach to the architecture-centric model-driven engineering of software systems that sets the "ability to evolve" as its central characteristic. Evolution arises in response to changes to requirements as well as to run-time feedback. Within this focus, ArchWare comprises:

– the definition of formal languages (with textual and graphical notations) for modelling dynamic[2] architectures (including expression of run-time structure, behaviour, and semantic properties from a run-time component-and-connector viewpoint) and expressing their analyses and refinements;

[1] The ArchWare European Project is partially funded by the Commission of the European Union under contract No. IST-2001-32360 in the IST-V Framework Program.

[2] The architecture of a software system is dynamic if it can change during the course of its execution.

– the implementation of a customisable environment for architecture-centric software engineering, including processes and tools for supporting architecture description, analysis, refinement, code generation, and their evolution;
– the validation of ArchWare languages and environment through industrial business cases and its dissemination as Open Source Software.

The novelty of the ArchWare approach lies in its holistic view of software. This starts with the capturing of the key architectural decisions by the definition of architecture styles, which might also involve the specification of invariant properties of the software (w.r.t. aspects such as structure, behaviour and qualities). Such properties may then be checked/proved using analysis tools throughout the software life cycle. Since ArchWare accommodates both static and dynamic qualities, it is essential that the run-time system can provide feedback and evolve while preserving the defined properties according to any policy and constraint defined at the level of the architectural style.

ArchWare goes beyond existing software environments in supporting architecture-centric engineering of software systems. In ArchWare both the software engineering process and its products at every stage of the software life cycle, including specification, implementation, qualities and indeed architectural styles themselves, are run-time evolvable. While existing software environments have proposed and implemented mechanisms for particular evolutionary scenarios, ArchWare addresses the problem of evolution at all levels of abstraction throughout the lifecycle of a software system.

The engineering of evolvable software systems requires the capture of key design decisions about “how” the software is to function in terms of expected behaviours, “what” is its structure in terms of components and their connectors, and “which” qualities are to be guaranteed. Furthermore, an appropriate refinement process (describing “how to build” the software) is also to be effectively supported.

The core of ArchWare is its integrated set of architecture-centric languages: an architecture description language, an architecture analysis language, and an architecture refinement language. These integrated languages are supported by an integrated toolset including a UML-based visual editor, a hyper-code textual editor, a graphical animator, a property checker, a refiner, a virtual machine, and a code-generator synthesizer.

ArchWare provides a process-integrated software engineering environment. The ArchWare software engineering process is itself influenced by its architecture-centric nature, in which the software architecture description is used to organise development activities. This architecture-centric approach guarantees that compliant implementations conform to the architecture description (including structural, behavioural, and quality properties).

A further central aspect of ArchWare is the support for the dynamic evolvability of applications and embedded processes. ArchWare enables deployed software architectures to evolve, in a controlled manner, in order to address aspects such as run-time evolution of requirements and technology.

This paper gives an overview of ArchWare. The remainder of the paper is organised as follows. Section 2 introduces architecture-centric model-driven software engineering. Section 3 briefly presents the ArchWare architecture-centric languages: the description, analysis and refinement languages. Section 4 introduces the ArchWare architecture-centric integrated development environment. Section 5 addresses related work and concludes the paper.

## 2 Architecture-Centric Model-Driven Software Engineering

All forms of engineering rely on models to design real-world systems. Models are used in many ways: to understand specific system aspects, predict system qualities, reason about impact of changes, and communicate major system features to stakeholders.

Figure 1 shows the spectrum of modelling approaches in software engineering [13]. Each category identifies a particular use of models in assisting software engineers to create running applications (code) for a specific run-time platform as well as the relationship between the models and the code.

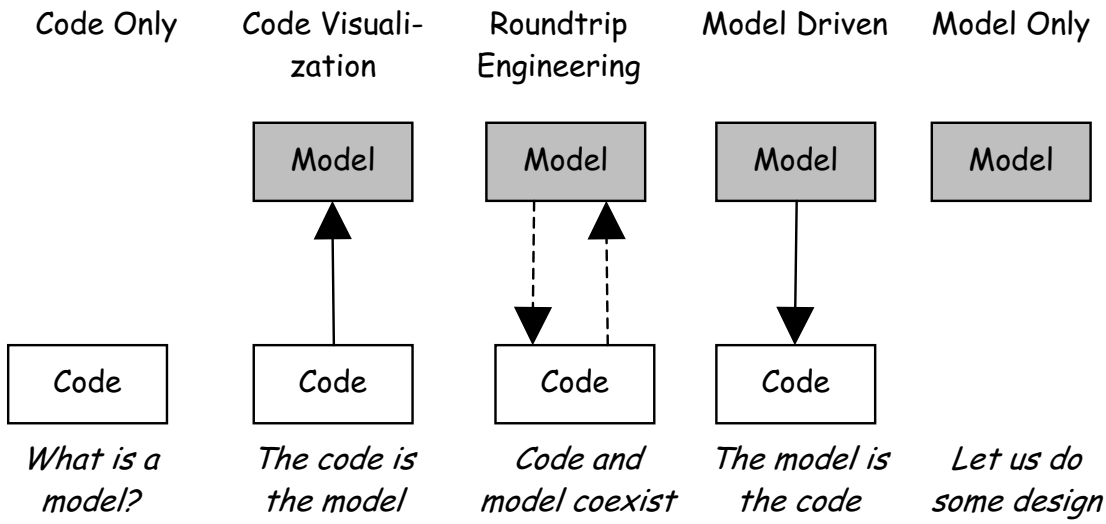


**Fig. 1.** The Modelling Spectrum.

Models are at the heart of the ArchWare approach. ArchWare provides a model-driven approach, i.e. the system models have sufficient detail to enable the generation of a full system implementation from the models themselves. Indeed, "the model is the code", i.e. the focus is on modelling and code is mechanically generated from models. In ArchWare, models are architecture-centric (run-time) models. They are executable and support analysis and refinement. This is illustrated in Figure 2 which shows a variety of architecture-centric model-driven activities and stakeholders that may be used when developing applications in ArchWare.

Typical architecture-centric activities are "define style", "define architecture", and "refine architecture" (the last refinement is code generation). Typical stakeholders are "style architect", "application architect", and "application engineer".

"Define style" activities, whose principal actors are the "style architects", represent the top level inception of a family of software architectures. An architecture style defines a domain specific architecture description "profile", including formal definitions of what an architectural element is, and what its invariant properties (including qualities) are, how elements can be combined, which constraints apply, and which processes can be applied to architecture elements and whole architecture descriptions (notably w.r.t. refinement and evolution).

"Define architecture" activities, whose principal actors are the "application architects", may use the domain specific styles defined by the style architect to describe a

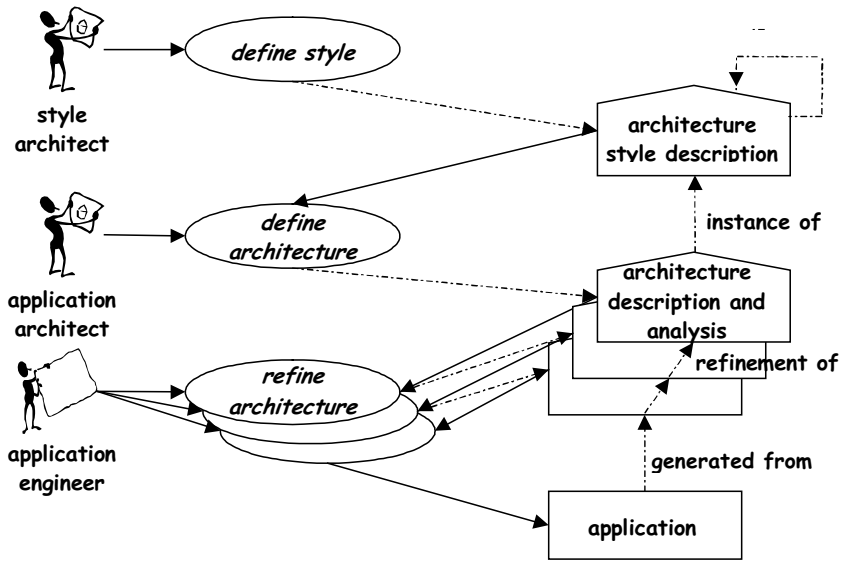


**Fig. 2.** ArchWare Architecture-centric Model-driven Software Engineering.

specific software architecture. In ArchWare an architecture description conforms to the properties and constraints defined by its style and can represent a system at various levels of abstractions (w.r.t. the detail of implementation decisions provided by the application engineer).

"Refine architecture" activities, whose principal actors are the "application engineer", support refinement transformations from abstract to more concrete architecture descriptions. Thus, an abstract – platform independent – architecture description can be refined to a concrete – platform specific – description of a specific application. The role of the application engineer is to derive that concrete description by applying correctness preserving refinements that conform to the constraints defined by the application architect and by the adopted architecture styles.

It is worth noting that this software engineering process model is not hard-coded in ArchWare. It is one possible architecture-centric software engineering process model that can be explicitly defined and enacted in ArchWare. ArchWare provides a library of software engineering process models that includes this one and they are all evolvable.

## 3 ARCHWARE Architecture-Centric Languages

ArchWare provides languages for describing dynamic architectures, analysing architecture structural and behavioural properties, and refining architecture descriptions. These languages are implemented using a software environment for supporting the evolution of architectural models, and this environment includes the processes used to develop the environment.

### 3.1 ARCHWARE Architecture Description Language

In ArchWare, architecture description encompasses two aspects: the expression and verification of architectural styles (typically carried out by style architects) and of software architectures themselves (typically carried out by application architects).

The ArchWare Architecture Description Language (ADL) [50][48] provides the core structure and behaviour constructs for describing dynamic software architectures. It is a formal specification language designed to be executable and to support automated analysis and refinement of dynamic architectures.

The ARCHWARE ADL has as formal foundation the higher-order typed π-calculus [54], a higher-order calculus for communicating and mobile systems. The ARCHWARE ADL is itself a formal language defined as a domain-specific extension of the higher-order typed π-calculus: it is a well-formed extension for defining a calculus of communicating and mobile architectural elements.

The ARCHWARE ADL takes its roots in previous work concerning the use of π-calculus as semantic foundation for architecture description languages [15][14]. Indeed, a natural candidate for expressing dynamic (run-time) behaviour would be the π-calculus as it is [43], which provides a general model of computation and is Turing-complete. This means that in π-calculus "every computation is possible but not necessarily easy to express". In fact, the classical π-calculus is not suitable as an architecture description language since it does not provide architecture-centric constructs to easily express architectures in particular w.r.t. architectural structures. Therefore, a language encompassing both structural and behavioural architecture-centric constructs is needed. The ARCHWARE ADL is this encompassing language, defined as a domain-specific extension of the higher-order typed π-calculus. It achieves Turing completeness and high architecture expressiveness with a simple formal notation.

The following general principles guided the design of ARCHWARE ADL:

- formality: ARCHWARE ADL is a formal language: it provides a formal system, at the mathematical sense, for describing dynamic software architectures and reasoning about them;
- run-time viewpoint: ARCHWARE ADL focuses on the formal description of software architectures from the run-time viewpoint: the (run-time) structure, the (run-time) behaviour, and how these may evolve over time;
- executability: ARCHWARE ADL is an executable language: a virtual machine runs specifications of software architectures;
- user-friendliness: ARCHWARE ADL supports different concrete syntaxes – textual [17][59] and graphical [6][7] (including UML-based) notations – to ease its use by architects and engineers.

Based on these general principles, the design of ARCHWARE ADL followed the following language design principles [47][57][58]:

- the principle of correspondence: the use of names are consistent within ARCHWARE ADL, in particular there is a one to one correspondence between the method of introducing names in declarations and parameter lists;
- the principle of abstraction: all major syntactic categories have abstractions defined over them (in ARCHWARE ADL, it includes abstractions over behaviours and abstractions over data),
- the principle of data type completeness: all data types are first-class without any restriction on their use.

In first-class citizenship, i.e. in addition to rights derived from type completeness (i.e. where a type may be used in a constructor, any type is legal without exception), there are properties possessed by all values of all types that constitute their civil rights in the language. In ARCHWARE ADL they are:

– the right to be declared,
– the right to be assigned,
– the right to have equality defined over them,
– the right to persist.

Additionally, ARCHWARE ADL provides an extension mechanism, i.e. new constructs can be defined on top of the language using user-defined mixfix abstractions. This extension mechanism provides the basis for providing style-based definitions.

In ArchWare, a style notation [16], built on the ARCHWARE ADL, provides the style constructs from which the base component-and-connector style and other derived styles can be defined. Conceptually, an architectural style includes:

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

ArchWare provides a novel ADL that is general-purpose and Turing-complete. The advantage w.r.t. other ADLs is that the ARCHWARE ADL supports user-defined architectural component-and-connector abstractions (instead of being obliged to use hard-coded abstractions provided by particular ADLs that very often do not meet architect needs). It can be seen as a second generation ADL: in first generation ADLs, languages were not complete and architectural (run-time) concepts were hard-coded as language constructs; in second generation, languages should be complete and architectural (run-time) concepts should be customisable.

### 3.2 ARCHWARE Architecture Analysis Language

In ArchWare, architecture analysis encompasses two aspects: the expression and verification of properties of architectural styles (typically carried out by style architects) and of software architectures themselves (typically carried out by application architects).

The ArchWare Architecture Analysis Language (AAL) [4] provides a uniform framework for specifying relevant properties of styles and architectures. These properties have different natures: they can be structural (e.g. cardinality of architectural elements, interconnection topology) or behavioural (e.g. safety, liveness, and fairness defined on actions of the system). The ARCHWARE AAL complements the ARCHWARE ADL with features allowing architects to express and verify properties of software architectures and styles in a natural way. Analysis is intended to be performed according to three approaches: model-checking, theorem proving and specific external tools.

The ARCHWARE AAL is a formal property expression language designed to support automated verification. Thereby, one can mechanically check whether an architecture described in ARCHWARE ADL satisfies a property expressed in ARCHWARE AAL.

The ARCHWARE AAL has as formal foundation the modal μ-calculus [35], a calculus for expressing properties of labelled transition systems by using least and greatest fixed point operators. ARCHWARE AAL is itself a formal language defined as an ex-

tension of the μ-calculus: it is a well-formed extension for defining a calculus for expressing structural and behavioural properties of communicating and mobile architectural elements.

The ARCHWARE AAL takes its roots in previous work concerning the extension of modal operators with data-handling constructs [41], the use of regular expressions as specification formalism for value-passing process algebras [23], and the extension of fixed point operators with typed parameters [30].

Indeed, a natural candidate for "pure" behavioural properties would be the modal μ-calculus, which is a very expressive fixed point-based formalism subsuming virtually all temporal logics defined so far in the literature [55]. However, since ARCHWARE AAL must also provide features for expressing structural properties of architectures [5], the modal μ-calculus is not sufficient. Therefore, a formalism encompassing both the predicate calculus and the modal μ-calculus is needed. The ARCHWARE AAL is, thereby, this encompassing formalism, defined as a domain-specific extension of the μ-calculus.

The ARCHWARE AAL combines predicate logic with temporal logic in order to allow the specification of both structural and behavioural properties. It enables automated verification of property satisfaction by model checking (through on-the-fly model checking) or theorem proving (through deductive verification using tabled logic programming).

### 3.3 ARCHWARE Architecture Refinement Language

Software applications are usually developed in several steps. Indeed, the concrete architecture of a software system is often developed through vertical and horizontal refinements of related architectures that differ respectively w.r.t. abstraction and partition dimensions.

Vertical refinement steps add more and more details to abstract models until the concrete architectural model is described. A vertical refinement step typically leads to a more detailed architecture description that increases the determinism while implying properties of the abstract description. Generally, an abstract architecture is smaller and easier to understand and a concrete architecture reflects more implementation concerns.

Horizontal refinement is concerned with partitioning of an architecture. For instance, partitioning an abstract component in its parts at the same abstraction level.

ArchWare supports both vertical and horizontal refinement. In ArchWare, the underlying approach for architectural refinement is underspecification, i.e. at a high-level of abstraction, when specifying an architectural element, certain aspects can be left open. The decrease of this underspecification establishes a refinement relation for architectural elements.

A refinement relation in ArchWare, from an external or internal point of view, comprises four forms of refinement:

- behaviour refinement,
- port refinement,
- structure refinement,
- data refinement.

In behaviour refinement, the underspecification may concern the external (observable) behaviour or the internal behaviour of an architectural element. The external behaviour of an architectural element is the behaviour that its environment can observe, i.e. its behaviour from an external point of view. The internal behaviour concerns the internal expression of behaviour within the scope of the architectural element. The structure of an architectural element is its internal structure in terms of sub-architectural elements and their connected ports, i.e. the structure within the scope of the architectural element from an internal point of view. The ports of an architectural element provide the interaction points (i.e. connections) between the element and its environment, i.e. its ports from an external point of view.

The most fundamental notion of refinement in ArchWare is behaviour refinement. The other forms of refinement imply behaviour refinement modulo port, structure and data mappings.

In general, architectural refinement is a combination of the four forms of refinement. For instance, an architect can define an abstract architecture, then "data" refine that architecture in order to introduce base and constructed data types, then "port" refine the architecture to have ports with finer grain connections carrying data of different types, then "structure" refine its composite behaviour by adding new finer grain connectors, and so on.

The ArchWare Architecture Refinement Language (ARL) [49] provides constructs for defining refinements of the four forms cited so far, according to external or internal points of view. Composite refinements can be defined in terms of refinement primitives and composite refinements themselves. Refinement primitives comprise:

- adding, removing, replacing or transforming data type declarations of an architecture,
- adding, removing, replacing or transforming ports of an architecture,
- adding, removing, replacing or transforming output and input connections of ports of an architecture,
- transforming the behaviour of an architecture or the behaviour of a component or connector in an architecture,
- adding, removing, replacing or transforming components or connectors in an architecture,
- exploding or imploding components or connectors in an architecture,
- unifying or separating connections of ports in an architecture.

These primitives, applied step by step, allow the incremental transformation of an architecture description. These transformations are enforced to be refinements if preconditions of refinement primitives are satisfied and proof obligations discarded. A refinement engine based on rewriting logics [40][12] runs the refinement descriptions expressed in ARCHWARE ARL generating further refined architectures. Code is generated from refined (concrete) architectures.

The ARCHWARE ARL is a formal (executable) refinement language providing architecture-centric refinement primitives and supporting refinement compositions in both vertical and horizontal dimensions, from external or internal points of view. When applied, they refine architectural models described in ARCHWARE ADL outputting new refined architectural models also in ARCHWARE ADL.

ARCHWARE ARL provides the required key features for supporting architecture-centric model-driven formal development. By addressing software development as a set of architecture-centric model refinements, the refinements between models be-

come first class elements of the software engineering process. This is significant because a great deal of work takes places in defining these refinements, often requiring specialized knowledge on source and target abstraction levels, for instance knowledge on the source application logics and on the targeted implementation platforms. Efficiency and quality of software systems can be improved by capturing these refinements explicitly and reusing them consistently across developments. Thereby, user-defined refinement steps can be consistently defined, applied, validated, and mechanically automated.

## 4 ARCHWARE Integrated Development Environment

ArchWare implements a software development environment, i.e. the ArchWare Integrated Development Environment (IDE), to support the application of architecture-centric processes. The ARCHWARE IDE is composed of:

- The ArchWare Core Environment which provides the ARCHWARE ADL Compiler and Virtual Machine that supports the enactment of architecture descriptions.
- The ArchWare Core Meta-Process Models which provide the support for software processes that are used to build and evolve software applications.
- The ArchWare Environment Components which provide the ArchWare tools that support architecture description, analysis, and refinement processes.

The ARCHWARE ADL Virtual Machine provides the basic support for an evolvable persistent store used to both control the execution of applications and the processes used for their development and subsequent evolution [28]. Architecture descriptions expressed in ARCHWARE ADL will be enacted after compiling the ADL into the Virtual Machine executable code and thus will be preserved (and, more significantly, can be dynamically evolved) in the target persistent store.

The main ArchWare Core Meta-Process Model is the Process for Process Evolution (P2E) [29]. This provides a recursive mechanism for selecting a process abstraction (written in ARCHWARE ADL) from a library of existing abstractions or for producing a new (or evolved) abstraction if a suitable one is not available. The recursion means that abstractions for abstraction production are developed (and evolved) in the same way. This recursion can apply to any depth.

The ARCHWARE ADL Virtual Machine persistent store provides a repository for storing, retrieving and refining architectural models. Basic support is provided through the concept of an architecture "node" which is a reference to a element description written in ARCHWARE ADL. Nodes are related through a graph structure where the directional arcs represent different dimensions of refinement supported by ArchWare. The ARCHWARE IDE provides operations to support the creation and evolution of this graph structure. There is a refine operation which allows for one node to be related to another in terms of the child node being a more concrete vertical refinement of its parent. There is a partition operation which allows for nodes to be related by horizontal refinement in terms of the child(ren) being parts explosion of the parent. Methods of vertical and horizontal refinements are written in ARCHWARE ARL. There is also a satisfy operation to support verification of properties written in ARCHWARE AAL. The nodes (and graph structure) are evolved using the P2E meta-process cited so far.

ArchWare Environment Components are implemented either as ADL enactable descriptions (and hence wholly stored in the ADL Virtual Machine persistent store) or as COTS[3]-like components integrated by wrapping them using ARCHWARE ADL as with other application software. Wrapped application components are essentially the end-user components of an ArchWare-based application in operation.

From the viewpoint of ArchWare users, the establishment of an integrated persistent environment that supports higher-order code yields a new software development paradigm, hyper-code [61], in which source code may include direct links to data and code values that exist in the persistent environment. Hyper-code unifies source code and executable code. The result is that the distinction between them is completely removed: the software engineer sees only a single code representation form throughout the software engineering process, during software construction, execution, debugging, and viewing existing code and data.

The integration of hyper-code into the ARCHWARE ADL supports this unified view [8]: an architecture can be described, then compiled to be executed, afterwards during execution composed with other descriptions via hyper-code, and so on.

The ARCHWARE IDE is itself an example of an evolutionary software system. ArchWare adopts the methods and principles it itself is developing in the development of the ArchWare Engineering Environment.

## 5 Related Work and Concluding Remarks

Several architecture description languages (ADLs) have been proposed in the literature, including: ACME/Dynamic-ACME [26][27], AESOP [25], AML [60], ARMANI [44], CHAM-ADL [32][33], DARWIN [39], META-H [11], PADL [9], RAPIDE [52][38], SADL [45][46], σπ-SPACE [15][36], UNICON-2 [19], and WRIGHT/Dynamic-WRIGHT [2][3].

ADLs provide both a concrete syntax and a formal, or semi-formal, semantics. Typically, they embody a conceptual framework reflecting characteristics of the domain for which the ADL is intended and/or an architectural style [42].

The focus of ArchWare is on languages and tools for the formal modelling of dynamic software architectures (evolvable at design and run-time), and for the computer-aided formal analysis and refinement of these models. In a broad sense the ARCHWARE ADL is composed of a set of integrated notations: the architecture description notation (the core), the style definition notation, the property expression notation, and the refinement definition notation. No other ADL provides a comprehensive set of notations for describing dynamic architectures.

But how does ArchWare compare with related work? Comparing ADLs objectively is a difficult task because their focuses are quite different. Most ADLs essentially provide a component-and-connector built-in model of architecture description and formalise topological constraints. The reason for this is probably that structure is certainly the most understandable and visible part of an architecture. But behavioural and quality aspects are not completely neglected. They are often taken into account (even if partially) in most ADLs. They are certainly an essential part of architecture description.

---

[3] Commercial off-the-shelf (COTS).

ARCHWARE ADL is the most general among studied ADLs. Instead of hard coding a specific component-and-connector viewpoint model, it is a general-purpose ADL that can be used to define, in a compliant way, different user-defined component-and-connector viewpoint models.

ARCHWARE AAL provides the notation to express properties of architectures described in ARCHWARE ADL.

ARCHWARE AAL combines predicate logic with temporal logic in order to allow the specification of both structural properties and behavioural properties concerning architecture descriptions obtained by instantiating a given style.

Regarding behavioural properties, the choice of modal μ-calculus as the underlying formalism provides a significant expressive power. Moreover, the extension of μ-calculus modalities with higher level constructs such as regular formulas inspired from early dynamic logics like PDL [21] facilitates the specification task of the practitioners, by allowing a more natural and concise description of properties involving complex sequences of actions. The extension of fixed point operators with data parameters also provides a significant increase of the practical expressive power, and is naturally adapted for specifying behavioural properties of value-passing languages such as the ARCHWARE ADL.

In the context of software architectures, several attempts at using classical process algebras and generic model-checking technology have been reported in the literature. In [31], various architectural styles (e.g., repository, pipe-and-filter, and event-action) are described in LOTOS, by using specific communication patterns and constraints on the form of components, and verified using the CADP toolbox [20][24]. In [53], several variants of the pipe-and-filter style are described in LOTOS and analysed using CADP. In [34], the transformation of software architectures specified in LOTOS and their verification using the XTL model-checker [41] of CADP are presented. Finally, an approach for checking deadlock freedom of software architectures described using a variant of CCS is described in [10]. All these works provide rather ad-hoc solutions for a class of software architectures limited to static communication between architectural elements, and can be subsumed by the more general framework provided by ARCHWARE ADL/AAL and verification tools.

Regarding structural analysis, ACME, ARMANI, CHAM-ADL, DARWIN, RAPIDE, SADL, WRIGHT and others addressed mainly properties like completeness and consistency of software architectures. Most of those approaches propose a less or more sophisticated language for describing properties to analyse.

The main limitation of most of these approaches with regard to ArchWare objectives is that they address either structural or behavioural properties, but not both as in ArchWare.

As regards architecture refinement, with the exception of a variant of FOCUS [56], i.e. FOCUS/DFA [51], RAPIDE and SADL, there is no proposal for a rigorous calculus based on architectural terms. In the case of SADL the refinement is only structural. In the case of RAPIDE it is only behavioural (supported by simulations). In both cases, clear architectural primitives for refining architectures are not provided and the refinement supported is only partial. ARCHWARE ARL, like the B [1] and Z [18] formal methods, provides operations to transform specifications. However, unlike FOCUS, B and Z, ARCHWARE ARL has been specially designed to deal with architectural elements of any architectural style. Unlike SADL, ARCHWARE ARL supports underspecification. In FOCUS/DFA, refinement is essentially logical implication. In SADL, it is restricted by faithful interpretation. In RAPIDE, it is defined by

simulations. In ArchWare, it is based on property implication, where the properties to be preserved are defined by the architect.

The ARCHWARE ARL provides a novel language that on the one side has been specifically designed for architectural refinement taking into account refinement of behaviour, port, structure, and data from an architectural perspective and on the other side is based on preservation of properties. The core of ARCHWARE ARL is a set of architecture transformation primitives that support refinement of architecture descriptions. Transformations are refinements when they preserve properties of the more abstract architecture. Core properties are built-in. Style-specific or architecture-specific properties are user defined. The underlying foundation for architected behaviours is the higher-order typed π-calculus. Satisfying proof obligations in ARCHWARE ARL is supported by the ArchWare analysis tools, which comprises a model checker, a prover and specific evaluators.

A detailed positioning of ARCHWARE ADL, AAL and ARL w.r.t. the state-of-art is given in [22][37][49].

ARCHWARE languages have been applied in several realistic case studies and industrial business cases at Thésame (France) and Engineering Ingegneria Informatica (Italy). The pilot project at Thésame aims to develop and evolve agile integrated industrial process systems. The pilot project at Engineering Ingegneria Informatica aims to develop and evolve federated knowledge management systems. ARCHWARE languages have also been used by the CERN (Switzerland) for architecting human computer interfaces for monitoring particle accelerator restart. Ongoing work focuses on customisations of the ARCHWARE IDE for automating engineering processes in these applications.

Besides providing languages, frameworks, processes, and tools for architecture-centric model-driven engineering, ArchWare enforces a novel relationship between software systems and their development environments. Indeed, in ArchWare, software systems and the software development environments that support their engineering are both evolving artefacts; the keystone of ArchWare is recognising that compositional, architecture-centric evolutionary approaches (supported by adequate formal languages, frameworks, processes and tools) are needed to effectively construct and operate both kinds of systems. Thus, ArchWare solutions are applied consistently to the software systems being engineered as well as to the software engineering process and environment themselves. Jointly, they exploit the many advantages of the ArchWare architecture-centric compositional and evolutionary framework. This gives the possibility of an ongoing link between software systems and their software engineering processes in order to support continuous evolution, thereby accommodating change over their lifetime.

## References


1. Abrial J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996.
2. Allen R.: A Formal Approach to Software Architectures. PhD Thesis, Carnegie Mellon University, 1997.
3. Allen R., Douence R., Garlan D.: Specifying and Analyzing Dynamic Software Architectures. In Fundamental Approaches to Software Engineering, LNCS 1382, Springer Verlag, 1998.

4. Alloui I., Garavel H., Mateescu R., Oquendo F.: The ArchWare Architecture Analysis Language: Syntax and Semantics. Deliverable D3.1b, ArchWare European RTD Project, IST-2001-32360, January 2003.
5. Alloui I., Oquendo F.: Supporting Decentralised Software-intensive Processes using ZETA Component-based Architecture Description Language. Enterprise Information Systems, Joaquim Filipe (Ed.), Kluwer Academic Publishers, 2002.
6. Alloui I., Oquendo F.: The ArchWare Architecture Description Language: UML Profile for Architecting with ArchWare ADL. Deliverable D1.4b, ArchWare European RTD Project, IST-2001-32360, June 2003.
7. Alloui I., Oquendo F.: Describing Software-intensive Process Architectures using a UML-based ADL, Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS'04), Porto, Portugal, April 2004.
8. Balasubramaniam D., Morrison R., Kirby G., Mickan K.: Integration of Hyper-code and Structural Reflection into ArchWare ADL. Deliverable D1.5, ArchWare European RTD Project, IST-2001-32360, February 2003.
9. Bernardo M., Ciancarini P., Donatiello L.: Architecting Systems with Process Algebras. Technical Report UBLCS-2001-7, July 2001.
10. Bernardo M., Ciancarini P., Donatiello L.: Detecting Architectural Mismatches in Process Algebraic Descriptions of Software Systems, Proceedings of the 2nd Working IEEE/IFIP Conference on Software Architecture, Amsterdam, IEEE-CS Press, August 2001.
11. Binns P., Engelhart M., Jackson M., Vestal S.: Domain-Specific Software Architectures for Guidance, Navigation, and Control. International Journal of Software Engineering and Knowledge Engineering. 1996.
12. Bolusset T., Oquendo F.: Formal Refinement of Software Architectures Based on Rewriting Logic, ZB2002 International Workshop on Refinement of Critical Systems: Methods, Tools and Experience, Grenoble, Janvier 2002.
13. Brown A.W.: An Introduction to Model Driven Architecture – Part I: MDA and Today's Systems. The Rational Edge, February 2004.
14. Chaudet C., Greenwood M., Oquendo F., Warboys B.: Architecture-Driven Software Engineering: Specifying, Generating, and Evolving Component-Based Software Systems. IEE Journal: Software Engineering, Vol. 147, No. 6, UK, December 2000.
15. Chaudet C., Oquendo F.: A Formal Architecture Description Language Based on Process Algebra for Evolving Software Systems. Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'00). IEEE Computer Society, Grenoble, September 2000.
16. Cimpan S., Leymonerie F., Oquendo F.: The ArchWare Foundation Styles Library. Report R1.3-1, ArchWare European RTD Project, IST-2001-32360, June 2003.
17. Cimpan S., Oquendo F., Balasubramaniam D., Kirby G., Morrison R.: The ArchWare Architecture Description Language: Textual Concrete Syntax. Deliverable D1.2b, ArchWare European RTD Project, IST-2001-32360, December 2002.
18. Davies J., Woodcock J.: Using Z: Specification, Refinement and Proof. Prentice Hall International Series in Computer Science, 1996.
19. DeLine R.: Toward User-Defined Element Types and Architectural Styles. Proceedings of the 2nd International Software Architecture Workshop, San Francisco, 1996.
20. Fernandez J-C., Garavel H., Kerbrat A., Mateescu R., Mounier L., Sighireanu M.: CADP (CAESAR/ALDEBARAN Development Package) – A Protocol Validation and Verification Toolbox, Proceedings of the 8th International Conference on Computer-Aided Verification, New Brunswick, USA, LNCS 1102, Springer Verlag, August 1996.
21. Fischer M.J., Ladner R.E.: Propositional Dynamic Logic of Regular Programs. Journal of Computer and System Sciences Vol. 18, 1979.
22. Gallo F. (Ed.): Annual Report: Project Achievements in 2002. Appendix B: Survey of State-of-the-Art and Typical Usage Scenario for ArchWare ADL and AAL. Deliverable D0.4.1, ArchWare European RTD Project, IST-2001-32360, February 2003.

23. Garavel H.: Compilation et Vérification de Programmes LOTOS. Thèse de Doctorat, Univ. Joseph Fourier (Grenoble), November 1989. Chapter 9: Vérification (In French).
24. Garavel H., Lang F., Mateescu R.: An Overview of CADP 2001. European Association for Software Science and Technology (EASST) Newsletter, Vol. 4, August 2002.
25. Garlan D., Allen R., Ockerbloom J.: Exploiting Style in Architectural Design Environments. Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering, New Orleans, 1994.
26. Garlan D., Monroe R., Wile D.: ACME: An Architecture Description Interchange Language. Proceedings of CASCON'97, Toronto, November 1997.
27. Garlan D., Monroe, R., Wile D.: ACME: Architectural Description of Component-Based Systems. Foundations of Component-Based Systems, Leavens G.T, and Sitaraman M. (Eds.), Cambridge University Press, 2000.
28. 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, Proceedings of the 9th European Workshop on Software Process Technology, LNCS 2786, Springer Verlag, Helsinki, September 2003.
29. Greenwood M., Robertson I., Seet W., Snowdon R., Warboys B.: Evolution Meta-Process Model. Deliverable D5.3, ArchWare European RTD Project, IST-2001-32360, December 2003.
30. Groote J. F., Mateescu R.: Verification of Temporal Properties of Processes in a Setting with Data. Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology, Amazonia, Brazil, LNCS 1548, January 1999.
31. Heisel M., Levy N.: Using LOTOS Patterns to Characterize Architectural Styles, Proceedings of the International Conference on Theory and Practice of Software Development, LNCS 1214, Springer Verlag, 1997.
32. Inverardi P., Wolf A.: Formal Specification an Analysis of Software Architectures using the Chemical Abstract Machine Model. IEEE Transactions on Software Engineering, Vol. 21, No. 4, April 1995.
33. Inverardi P., Wolf A., Yankelevich D.: Static Checking of System Behaviors using Derived Component Assumptions. ACM Transactions on Software Engineering and Methodology, Vol. 9, No. 3, July 2000.
34. Kerschbaumer A.: Non-Refinement Transformation of Software Architectures. Proceedings of the ZB2002 International Workshop on Refinement of Critical Systems: Methods, Tools and Experience, Grenoble, Janvier 2002.
35. Kozen D.: Results on the Propositional μ-Calculus. Theoretical Computer Science 27:333-354, 1983.
36. Leymonerie F., Cimpan S., Oquendo F. : Extension d'un langage de description architecturale pour la prise en compte des styles architecturaux : application à J2EE. Proceedings of the 14th International Conference on Software and Systems Engineering and their Applications. Paris, December 2001 (In French).
37. Leymonerie F., Cimpan S., Oquendo F., "État de l'art sur les styles architecturaux : classification et comparaison des langages de description d'architectures logicielles", Revue Génie Logiciel, No. 62, September 2002 (In French).
38. Luckham D.C., Kenney J.J., Augustin L.M., Vera J., Bryan D., Mann W.: Specification and Analysis of System Architecture Using RAPIDE. IEEE Transactions on Software Engineering, Vol. 21, No. 4, April 1995.
39. Magee J., Dulay N., Eisenbach S., Kramer J.: Specifying Distributed Software Architectures. Proceedings of the 5th European Software Engineering Conference, Sitges, Spain, September 1995.
40. Martí-Oliet N., Meseguer J.: Rewriting Logic: Roadmap and Bibliography. Theoretical Computer Science, 2001.
41. Mateescu R., Garavel H.: XTL: A Meta-Language and Tool for Temporal Logic Model-Checking. Proceedings of the 1st International Workshop on Software Tools for Technology Transfer, Aalborg, Denmark, July 1998.

42. Medvidovic N., Taylor R.: A Classification and Comparison Framework for Architecture Description Languages. Technical Report UCI-ICS-97-02, Department of Information and Computer Science, University of California. Irvine, February 1997.
43. Milner R.: Communicating and Mobile Systems: The Pi-Calculus. Cambridge University Press, 1999.
44. Monroe R.: Capturing Software Architecture Design Expertise with ARMANI. Technical Report CMU-CS-98-163, Carnegie Mellon University, January 2001.
45. Moriconi M., Qian X., Riemenschneider R.A.: Correct Architecture Refinement. IEEE Transactions on Software Engineering, Vol. 21, No. 4, April 1995.
46. Moriconi M., Riemenschneider R.A.: Introduction to SADL 1.0: A Language for Specifying Software Architecture Hierarchies. Computer Science Laboratory, SRI International, Technical Report SRI-CSL-97-01, March 1997.
47. Morrison R.: On the Development of S-algol. PhD Thesis, University of St Andrews, 1979.
48. Oquendo F.: The ArchWare Architecture Description Language: Tutorial. Report R1.1-1, ArchWare European RTD Project, IST-2001-32360, March 2003.
49. Oquendo F.: The ArchWare Architecture Refinement Language. Deliverable D6.1b, ArchWare European RTD Project, IST-2001-32360, December 2003.
50. Oquendo F., Alloui I., Cimpan S., Verjus H.: The ArchWare Architecture Description Language: Abstract Syntax and Formal Semantics. Deliverable D1.1b, ArchWare European RTD Project, IST-2001-32360, December 2002.
51. Philipps J., Rumpe B.: Refinement of Pipe and Filter Architectures. Proceedings of FM'99, LNCS 1708, 1999.
52. RAPIDE Design Team: Guide to the RAPIDE 1.0. Language Reference Manuals, Stanford University, July 1997.
53. Rongviriyapanish S., Levy N.: Variations sur le Style Architectural Pipe and Filter. Actes du Colloque sur les Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'00), Grenoble, France, January 2000.
54. Sangiorgi, D., Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD Thesis, University of Edinburgh, 1992.
55. Stirling C.: Modal and Temporal Properties of Processes. Springer Verlag, 2001.
56. Stolen K., Broy M.: Specification and Development of Interactive Systems. Springer Verlag, 2001.
57. Strachey C.: Fundamental Concepts in Programming Languages. Oxford University Press, Oxford, 1967.
58. Tennent R.D.: Language Design Methods based on Semantic Principles. Acta Informatica 8, 1977.
59. Verjus H., Oquendo F.: The ArchWare Architecture Description Language: XML Concrete Syntax. Deliverable D1.3b, ArchWare European RTD Project, IST-2001-32360, June 2003.
60. Wile D.: AML: An Architecture Meta Language. Proceedings of the 14th International Conference on Automated Software Engineering, pp. 183-190. Cocoa Beach. October 1999.
61. Zirintsis, E.: Towards Simplification of the Software Development Process: The Hypercode Abstraction. PhD Thesis, University of St Andrews, 2000.